2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

OpTaS: An Optimization-based Task Specification Library for Trajectory Optimization and Model Predictive Control

Christopher E. Mower, João Moura, Nazanin Zamani Behabadi,
Sethu Vijayakumar, Tom Vercauteren*, Christos Bergeles*

***Abstract*— This paper presents OpTaS, a task specification Python library for Trajectory Optimization (TO) and Model Predictive Control (MPC) in robotics. Both TO and MPC are increasingly receiving interest in optimal control and in particular handling dynamic environments. While a flurry of software libraries exists to handle such problems, they either provide interfaces that are limited to a specific problem formulation (e.g. TracIK, CHOMP), or are large and statically specify the problem in configuration files (e.g. EXOTica, eTaSL). OpTaS, on the other hand, allows a user to specify custom nonlinear constrained problem formulations in a single Python script allowing the controller parameters to be modified during execution. The library provides interface to several open source and commercial solvers (e.g. IPOPT, SNOPT, KNITRO, SciPy) to facilitate integration with established workflows in robotics. Further benefits of OpTaS are highlighted through a thorough comparison with common libraries. An additional key advantage of OpTaS is the ability to define optimal control tasks in the joint-space, task-space, or indeed simultaneously. The code for OpTaS is easily installed via `pip`, and the source code with examples can be found at github.com/cmower/optas.**

I. Introduction

High-dimensional motion planning and control are essential for complex manipulation tasks in unstructured and dynamic environments, such as placing objects on shelves or surgical procedures like pedicle screw fixation (see Fig. 1). The planner and controller must account for bi-manual coordination, contact constraints, and robustness to disturbances. Efficient motion planning and fast controllers enable robots to perform these tasks while considering motion constraints, system dynamics, and changing task objectives.

Sampling-based planners [1] are effective but usually need significant post-processing, such as trajectory smoothing.

C. E. Mower, C. Bergeles and T. Vercauteren are with the School of Biomedical Engineering & Imaging Sciences, King's College London, UK. J. Moura and S. Vijaykumar are with School of Informatics, University of Edinburgh, UK. Correspondence: christopher.mower@kcl.ac.uk.

This research received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 101016985 (FAROS). Further, this work was supported by core funding from the Wellcome/EPSRC [WT203148/Z/16/Z; NS/A000049/1]. T. Vercauteren is supported by a Medtronic / RAEng Research Chair [RCSRF1819\7\34], and C. Bergeles by an ERC Starting Grant [714562]. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017008, Enhancing Healthcare with Assistive Robotic Mobile Manipulation (HARMONY). This research is supported by Kawada Robotics Corporation, Japan and the Alan Turing Institute, UK.

*C. Bergeles and T. Vercauteren equally contributed to the work.

(a) (b)

Fig. 1: Examples of contact-rich manipulation including (a) a robot placing an item on a shelf, and (b) a human interacting with a robot during a drilling task for pedicle screw fixation. Image credit: University Hospital Balgrist, Daniel Hager Photography & Film GmbH.

Optimal planners like RRT*, that are provably asymptotically optimal, are promising but inefficient for solving high-dimensional problems [2].

Gradient-based trajectory optimization (TO) is a key approach in optimal control and motion planning in robotics, as seen in recent works such as [3], [4], [5], [6], [7], [8], [9], [10]. Optimization starts with an initialization and finds a locally optimal state and control commands subject to motion constraints and system dynamics (i.e. equations of motion).

Several open-source and commercial optimization solvers, such as IPOPT [11], KNITRO [12], and SNOPT [13], are reliable for solving TO problems. However, there is a lack of libraries that allow for fast development and prototyping of optimization-based approaches for multi-robot setups that easily interface with these solvers, despite the success of optimization approaches in literature and motion planning frameworks like MoveIt [14].

This paper proposes OpTaS, a user-friendly task-specification library for rapid development and deployment of nonlinear optimization-based planning and control approaches, including Model Predictive Control (MPC). OpTaS leverages the symbolic framework provided by CasADi [15], enabling function derivatives to an arbitrary order, which is crucial for solvers like SNOPT that require Jacobian and Hessian of the objective function and constraints.

A. Related work

In this section, we review popular optimization solvers and their interfaces, similar works to our proposed library, and summarize key differences. Table I provides a summary of alternatives and how they compare to OpTaS.

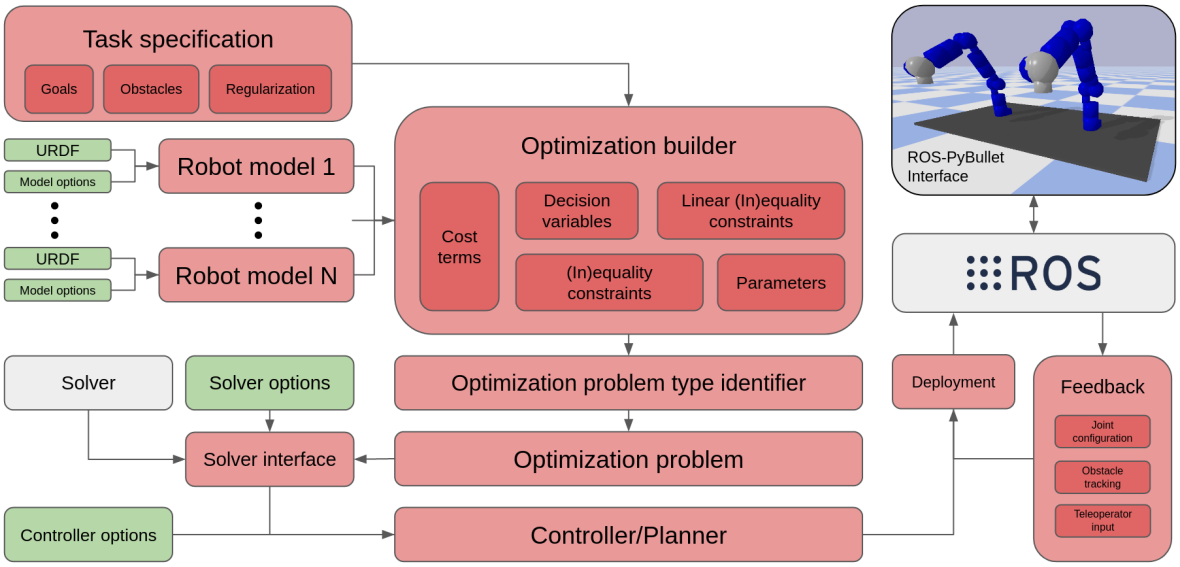

Fig. 2: System overview for the proposed OpTaS library. **Red** highlights the main features of the proposed library. **Green** shows configuration parameter input. **Grey** shows third-party frameworks/libraries. Finally, the image in the top-right corner shows integration with the ROS-PyBullet Interface [16].

Several open-source and commercial optimization solvers are available. For quadratic programming, OSQP provides a general-purpose solver based on the alternating direction method of multipliers [17]. Alternatively, CVXOPT uses a custom interior-point solver [18]. IPOPT is an interior-point solver for constrained nonlinear optimization. SNOPT has an interface to an SQP algorithm [13], while KNITRO solves general mixed-integer programs [12]. Note that SNOPT and KNITRO are proprietary.

Optimization solvers such as OSQP, CVXOPT, IPOPT, SNOPT, and KNITRO are often implemented in low-level programming languages such as C, C++, or FORTRAN, but there are also many interfaces via higher-level languages like Python to make implementation easier. SciPy's `optimize` module provides interfaces for low-level routines such as conjugate gradient and the BFGS algorithm [19], the Simplex method [20], COBYLA [21], and SLSQP [22]. Function gradients are necessary for many optimization-based methods. Automatic differentiation is implemented in various popular software packages such as JAX [23], PyTorch [24], and CasADi [15]. We chose CasADi because it is readily integrated with common solvers for optimal control, whereas JAX and PyTorch are not currently integrated with constrained nonlinear optimization solvers.

Similar packages to our proposed library are as follows: MoveIt [14] provides specific IK/planning formulations with interfaces to particular solvers; eTaSL [25] supports custom task specifications but only for problems formulated as quadratic programs; CASCLIK [26] uses CasADi for constraint-based inverse kinematic controllers but only allows optimization in the joint-space; and EXOTica [27] allows the user to specify a problem formulation from an XML file however requires the user to supply gradients analytically (otherwise the gradients are estimated using the finite differences method that can be less efficient and can be affected by roundoff errors [19]). Our framework supports joint- and task-space optimization for multiple robots in a single formulation, and leverages CasADi for automatic differentiation. Table I summarizes the key differences between these packages and OpTaS.

TABLE I: Comparison of OpTaS with common alternatives.

	Languages	End-pose	Traj.	MPC	Solver	AutoDiff	ROS	Re-form
OpTaS	Python	✓	✓	✓	QP/NLP	✓	✓	✓
EXOTica	Python/C++	✓	✓	✗	QP/NLP	✗	✓	✓
MoveIt	Python/C++	✓	✓	✗	QP	✗	✓	✗
TracIK	Python/C++	✓	✗	✗	QP	✗	✓	✗
RBDL	Python/C++	✓	✗	✗	QP	✗	✗	✗
eTaSL	C++	✓	✗	✗	QP	✓	✗[1]	✓
OpenRAVE	Python	✗	✓	✗	QP	✗	✓	✗

B. *Contributions*

This paper makes the following contributions:

- A task-specification Python library for rapid development/deployment of TO approaches for multi-robot setups.
- Given a URDF file, modeling of the robot kinematics to arbitrary derivative order.
- An interface that allows a user to easily reformulate an optimal control problem, and define parameterized constraints for online modification of the optimization problem.
- Analysis comparing the libraries performance (i.e. solver convergence, solution quality) versus existing packages. Further demonstrations highlight the ease in which nonlinear constrained optimization problems can be deployed in realistic settings.

[1]Enabled with external pluggins.

II. Problem Formulation

We can write an trajectory optimization problem as

$$\min_{x,u} \text{cost}(x, u; T) \quad \text{s.t.} \quad \dot{x} = f(x, u), x \in \mathbb{X}, u \in \mathbb{U} \quad (1)$$

where t denotes time, and $x = x(t) \in \mathbb{R}^{n_x}$ and $u = u(t) \in \mathbb{R}^{n_u}$ denote states and controls, T is the time-horizon, the scalar cost : $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ function models task objectives (typically a weighted sum of terms), the dot notation represents a derivative with respect to time t (i.e. $\dot{x} \equiv \frac{dx}{dt}$), f is the system dynamics (equations of motion) represented by several equality constraints, and $\mathbb{X} \subseteq \mathbb{R}^{n_x}$ and $\mathbb{U} \subseteq \mathbb{R}^{n_u}$ are feasible regions for the states and controls respectively (modeled by a set of equality and inequality constraints). Optimal control finds optimal (with respect to the cost function) states and controls for a discrete set of time steps. Typically numerical methods (e.g. Euler or Runge-Kutta) are used to estimate the integral of the system dynamics over the time horizon [28]. Given an initialization $x^{\text{init}}, u^{\text{init}}$, a locally optimal trajectory x^*, u^* is found by solving (1).

In Sec.I, we discussed optimization-based approaches for planning and control, which can be formulated under a TO problem as in (1). Our goal is to provide a library that enables quick development and prototyping of constrained nonlinear TO for multi-robot problems, with support for both IK and task-space TO. The library automates common steps such as transcription that transforms the problems task specification into a form accepted by optimization solver routines. Furthermore, many works in practice require the ability to adapt constraints dynamically to handle changes in the environment (e.g. MPC). This motivates a constraint parameterization feature.

III. Proposed Framework

Here, we present features of the proposed library (Fig. 2), implemented in Python. Python was chosen for its simplicity for beginners, versatility, and support for fast prototyping.

A. Robot model

The `RobotModel` provides kinematic modeling and specifies time derivative orders needed for the optimization problem. It can be instantiated with a URDF filename, string, or xacro filename. Additional base frames and end-effector links can be added programmatically, which is helpful when optimizing multiple robots and localizing their base frames within a global coordinate frame. Several robot model objects can be initialized to handle multi-robot setups (e.g. Fig. 3).

The `RobotModel` class provides access to data such as the number of degrees of freedom, actuated joint names, and limits. It also offers methods to compute forward kinematics and geometric Jacobian in any reference frame. Various kinematics modeling methods are provided, including the 4×4 homogeneous transformation matrix, rotational representations (e.g. Euler angles, quaternions), and geometric/analytical Jacobian. Each method requires a joint state, which can be supplied as a Python list, NumPy array, or CasADi symbolic array.

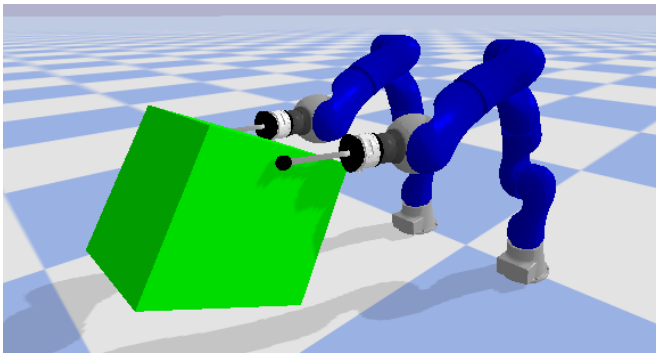

Fig. 3: Dual arm robot example. The two KUKA LWR arms collaborate to pick up a box. The plan is found by a single trajectory optimization problem. See the `example/` directory in the main OpTaS repository.

B. Task model

Some works optimize robot motion in the task-space and compute the IK as a secondary step, e.g. [8], [9]. The `TaskModel` represents any arbitrary trajectory, e.g. a 3D position trajectory. Time derivatives can also be specified similar to the `RobotModel`.

C. Optimization builder

The `OptimizationBuilder` class simplifies the process of setting up a TO problem and building an optimization problem model. The development cycle involves specifying the task (i.e. decision variables, parameters, cost function, and constraints) using intuitive syntax and symbolic variables. Several robot and task models (as described previously) can be passed to the builder that creates the corresponding decision variables for the states (e.g. positions, velocities, etc.). The builder then creates an optimization problem class that interfaces with several solvers.

1) Optimization problem model: The optimization builder class allows the user to define a TO problem in a user-friendly syntax. However, transcribing the problem into a form that can be solved by off-the-shelf solvers is not straightforward. The `build` method of the optimization builder class produces an optimization problem model enabling us to interface with multiple solvers.

The most general problem modeled by OpTaS is given by

$$X^* = \arg\min_X f(X; P) \quad (2a)$$

s.t.

$$k(X; P) = M(P)X + c(P) \geq 0 \quad (2b)$$

$$a(X; P) = A(P)X + b(P) = 0 \quad (2c)$$

$$g(X; P) \geq 0 \quad (2d)$$

$$h(X; P) = 0 \quad (2e)$$

where $X = [vec(x)^T, vec(u)^T]^T \in \mathbb{R}^{nX}$ is the decision variable array, $vec(\cdot)$ is a function that returns the input m-by-n array as a mn-by-1 vector, $P \in \mathbb{R}^{n_P}$ is the vectorized parameters, $f : \mathbb{R}^{n_X} \to \mathbb{R}$ is the objective function, k :

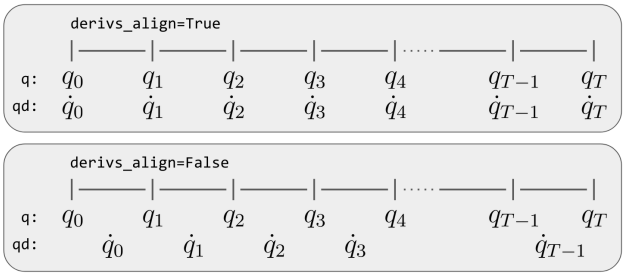

Fig. 4: Joint state alignment with time. The `derivs_align` flag specifies how the time derivatives should be aligned.

$\mathbb{R}^{nx} \to \mathbb{R}^{n_k}$ are linear inequality constraints, $a : \mathbb{R}^{nx} \to \mathbb{R}^{n_a}$ are linear equality constraints, $g : \mathbb{R}^{nx} \to \mathbb{R}^{n_g}$ are nonlinear inequality constraints, and $h : \mathbb{R}^{nx} \to \mathbb{R}^{n_h}$ are nonlinear equality constraints. The decision variables X represent robot and task model states and any other variables specified by the user stacked into a single vector. Similarly for the parameters and constraints. Vectorization is made possible by the `SXContainer` data structure implemented in the `sx_container` module. We can automatically transcribe the TO problem specified in (1) into the form (2) using this data structure.

Not all task specifications require defining all the functions in (2). The choice of solver for (2) depends on various factors, such as the structure of the objective function and constraints, time budget, and desired accuracy. For instance, a solver that only handles linear constraints (e.g. OSQP [17]) is not suitable for problems with nonlinear objective functions and constraints. The optimization builder automatically identifies the problem type and presents relevant solvers to the user. Available problem types include problems with either quadratic or nonlinear cost functions and those that are unconstrained, linearly constrained, or constrained by nonlinear functions.

2) Initialization: Upon initialization of the optimization builder class we can specify **(i)** the number of time steps in the trajectory, **(ii)** several robot and task models (given a unique name for each), **(iii)** the joint states (positions and required time-derivatives) that integrate the decision variable array, **(iv)** task-space labels, dimensions, and derivatives to also integrate the decision variable array, and **(v)** a Boolean describing the alignment of the derivatives (Fig. 4).

The alignment of time-derivatives can be specified in two ways. Each derivative is aligned with its corresponding state (alignement), or otherwise. This is specified by the `derivs_align` flag in the optimization builder interface and shown diagramatically in Fig. 4.

3) Decision variables and parameters: Decision variables for joint- and task-space trajectories are labeled and defined by the optimization builder interface. They can be retrieved using the `get_model_state` method, which requires specifying a robot or task name, time index, and derivative order. Extra decision variables can be added using the `add_decision_variables` method with a unique name and dimension. The `add_parameter` method is used to specify problem parameters (e.g. safe distances), requiring a unique name and dimension.

4) Cost and constraint functions: The cost function in (1) consists of multiple terms

$$\text{cost}(x, u; T) = \sum_i c_i(x, u; T) \tag{3}$$

where $c_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is an individual term modeling a particular sub-task. For example, define $c_0 = \|\psi(x_T) - \psi^*\|^2$ and $c_1 = \lambda \int_0^T \|u\|^2 \, dt$ (discretization is implicit in this formulation) where $\psi : \mathbb{R}^{n_x} \to \mathbb{R}^3$ is a function for the end-effector position (provided by the robot model, see Sec. III-A), $\psi^* \in \mathbb{R}^3$ is a goal position, and $0 < \lambda \in \mathbb{R}$ is a weight representing the relative importance of c_1. The c_0 term models ideal final states, and c_1 encourages control signals with minimal magnitudes (e.g. minimize joint velocities). Each cost term is added to the problem using the `add_cost_term` method; the `build` sequence ensures the terms are summed as the objective function.

Constraints can be added to the optimization problem using `add_equality_constraint` and `add_leq_inequality_constraint` for equality and inequality constraints respectively. The library differentiates between linear and nonlinear constraints by first checking if they are linear with respect to the decision variables. Additionally, OpTaS provides pre-implemented methods for common constraints, such as joint position/velocity limits and time-integration of the system dynamics f, enabling integration of velocities to positions.

D. Solver interface

OpTaS can interface with solvers via CasADi and also several others (both open-source and commercial) using the `Solver` class. Available solvers include IPOPT [11], SNOPT [13], KNITRO [12], Gurobi [29], the Scipy minimize method [30], OSQP [17], and CVXOPT [18]

1) Initialization of solver: When initializing the solver, variables are set up and the optimization problem object becomes a class attribute. The user then calls the `setup` method, which acts as an interface to the chosen solver's initialization. This method sets up the solver's interface and passes relevant solver parameters.

2) Resetting the interface: The solver may be called multiple times as a controller, requiring the problem parameters and initial seed to be reset. To reset the initial seed and problem parameters, the user calls `reset_initial_seed` and `reset_parameters`, respectively. The required vectorization is handled internally by the solver using the `SXContainer` data structure so that the user can specify the variables/parameters in the dimensions that are most intuitive to them (e.g. a position trajectory with a 3-by-n NumPy array); any unspecified values default to zero. This is particularly useful for feedback controllers or controllers with parameterized constraints, such as obstacles. The naming conventions for task/robot models are chosen such that warm-starting the optimization with the previous solution is easy; i.e. the user simply passes the solution object returned by the `solve` method.

```python
import optas

T = 100 # number of time steps in trajectory
tip = "ee_name" # name of end-effector in URDF
urdf = '/path/to/robot.urdf'
r = optas.RobotModel(urdf, time_derivs=[0, 1])
n = r.get_name()
b = optas.OptimizationBuilder(T, robots=[r])

qT = b.get_model_state(n, t=-1)  # final state
pg = b.add_parameter("pg", 3)  # goal pos.
qc = b.add_parameter("qc", r.ndof)  # init q
o = b.add_parameter("o", 3)  # obstacle pos.
s = b.add_parameter("s")  # obstacle radius
dt = b.add_parameter("dt")  # time step

p = r.get_global_link_position(tip, qT)  # FK
b.add_cost_term("goal", optas.sumsqr(p - pg))
b.integrate_model_states(
    n, time_deriv=1, dt=dt)
b.initial_configuration(n, qc)
for t in range(T):
    qt = b.get_model_state(n, t=t)
    pt = r.get_global_link_position(tip, qt)
    b.add_geq_inequality_constraint(
        f"obs_avoid_{t}",
        optas.sumsqr(pt - o), s**2)

solver = optas.CasADiSolver(
    b.build()).setup("ipopt")
```

Fig. 5: Example code for TO described in Section IV.

3) Solving an optimization problem: The `solve` method passes the problem to the solver and collects the resulting data, which is transformed into the state trajectory for each robot/task. The `interpolate` method can be used to interpolate trajectories across time, and the `stats` method retrieves optimization statistics such as the number of iterations – exactly what is returned is specific to each solver.

4) Extensible solver interface: The solver interface is designed for easy extensibility, allowing for integration of additional solvers. To include a new solver, the user creates a sub-class that inherits from the `Solver` class and implements three methods: (i) `setup` for initializing the solver interface, (ii) `_solve` to call the solver and return the optimized variable X^*, and (iii) `stats` to return any solver statistics.

E. Additional features

Integration with ROS [31] is provided via the dedicated `optas_ros` repository[2]. Additionally, OpTaS can interfaced with the ROS-PyBullet Interface [16].

In addition, we provide a partial port of the `spatialmath` library [32] supporting CasADi variables. The user can manipulate objects such as homogeneous transformation matrices, quaternions, Euler angles, etc. with symbolic variables.

IV. Code Example

This section demonstrates how easy it is to set up a TO problem using the example of finding an end-effector position trajectory. While we use position as an example, goals and constraints can also be set for other dimensions, including orientation

[2]github.com/cmower/optas_ros

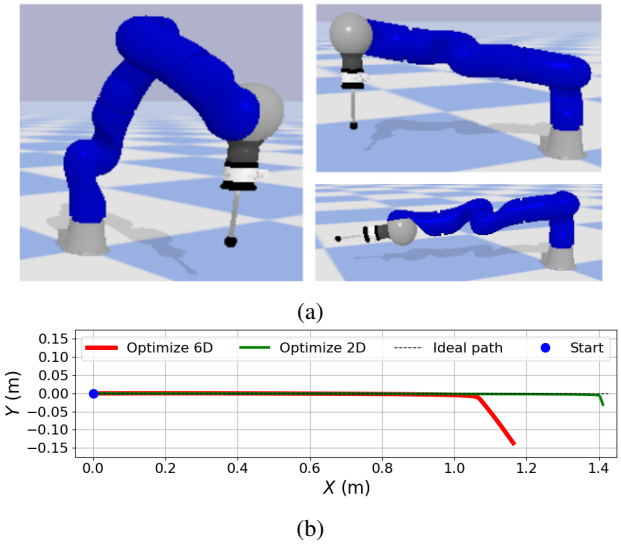

Fig. 6: Comparison of two end-effector task-space trajectory formulations. (a) Displays start (left) and final configurations for each approach (right). (b) Shows the 2D end-effector position trajectory plot.

Given a serial link manipulator, the objective is to find a collision-free plan from starting configuration q_c to a goal end-effector position p_g over a time horizon of T. A spherical collision is represented by position o and radius r. The robot's configuration at time t is denoted as q_t and its velocities $\dot{q}_t$, which act as controls.

The cost function, $\|p(q_T) - p_g\|^2$, is minimized subject to constraints: initial configuration $q_0 = q_c$, joint limits $q^- \leq q_t \leq q^+$, and obstacle avoidance $\|p(q_t) - o\|^2 \geq r^2$. The system dynamics are represented by equality constraints $q_{t+1} = q_t + \delta t \dot{q}_t$, which can be specified using the built-in method `integrate_model_states` in the `OptimizationBuilder` object. Fig. 5 shows the code for solving this TO problem.

V. Experiments

This section presents our experiments and results comparing OpTaS with alternative libraries and highlighting its beneficial features. We compare OpTaS against TracIK [33], which solves a specific IK problem formulation, and EXOTica [27], a library that offers similar functionality as OpTaS but, as mentioned in Sec. I-A, lacks automatic differentiation integration (requiring analytical derivative supply or finite differencing approximation).

A. Optimization along custom dimensions

TracIK [33], a commonly used solver, requires a 6D pose as the task-space goal. While suitable for various robotics problems (e.g. pick-and-place), optimizing every task-space dimension may be unnecessary (e.g. spraying applications do not require roll angular optimization). Moreover, optimizing more dimensions than necessary can be disadvantageous.

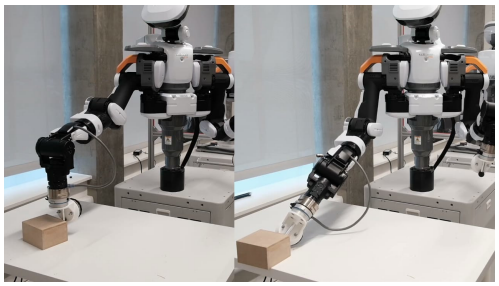

Fig. 7: The Kawada Nextage robot performing non-prehensile manipulation by pushing an object along a 2D plane running OpTaS. The left image shows the maximum reach when optimizing the full 6D end-effector pose, whereas the right image shows that the robot can reach further by optimizing only in the 2D plane.

OpTaS allows users to optimize or ignore any desired joint- or task-space dimension, which can have advantages such as expanding the robot's workspace. For instance, in a non-prehensile pushing task on a plane (Fig. 7), optimizing the full 6D pose may be suboptimal as the task is two-dimensional. By optimizing in the 2D plane and imposing boundary constraints on the third linear spatial dimension, the robot's workspace can be increased.

We used OpTaS to set up a tracking experiment with a simulated Kuka LWR robot arm to compare two cases: (i) optimizing the full 6D pose and (ii) optimizing 2D linear position. The task was to move the end-effector with a constant magnitude and direction velocity in the 2D plane. The initial configuration is shown in Fig.6a (left), and the end configurations for each approach are shown in Fig.6a (right), with the end-effector trajectories displayed in Fig. 6b. The 2D optimization problem achieved a greater distance, demonstrating an increased robot workspace. OpTaS was also setup on the Kawada Nextage robot and similarly we demonstration the improved workspace in Figure 7.

B. Performance comparison

In this section, we compare OpTaS to alternatives by formulating similar problems. We model the same problem used in TracIK [33] and EXOTica [27], using the Scipy SLSQP solver [22]. With the Kuka LWR robot arm, we setup a task where the robot tracks a figure-of-eight motion in task-space and record the CPU time for the solver duration at each control loop cycle (Fig. 8). TracIK is the fastest (0.049 ± 0.035ms), OpTaS is faster than EXOTica (2.608 ± 0.239ms and 3.694 ± 0.300ms, respectively).

In a second experiment, we compared OpTaS against EXOTica with an additional manipulability maximization cost term [34]. Using the same setup as before, the results shown in Fig. 9b indicate that OpTaS (2.650 ± 0.270ms) outperforms EXOTica (7.640 ± 1.404ms). EXOTica requires the user to supply analytical gradients for sub-tasks, and when analytical gradients are not available for a task such as manipulability maximization, it resorts to using finite difference method which can be slow to compute, likely contributing to the difference in performance.

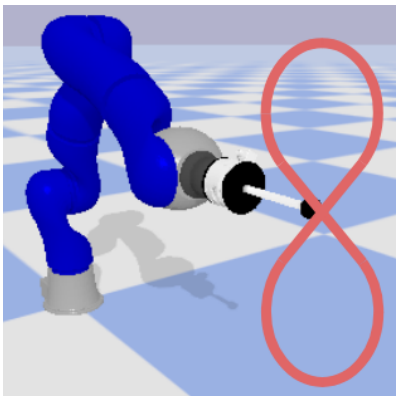

Fig. 8: Figure-of-eight trajectory tracked by the Kuka LWR.

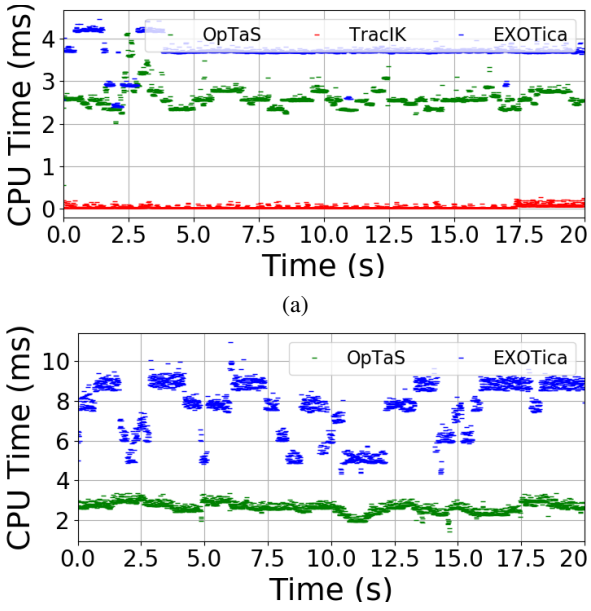

Fig. 9: Solver duration comparisons for figure of eight motion. (a) Compares an IK tracking approach described in Section V, (b) is a similar comparison that includes a maximization term for manipulability. Green is OpTaS, red is TracIK, and blue is EXOTica.

VI. Conclusions

This paper introduced OpTaS, a Python library for constrained nonlinear program optimization for TO and MPC. OpTaS allows users to set up custom problem formulations and has been demonstrated to perform well against alternatives. Parameterization enables easy implementation of feedback controllers, motion planners, and benchmarking problem formulations and solvers.

We hope OpTaS will be used by researchers, students, and industry to facilitate the development of control/planning algorithms. The code is easily installed via `pip` and has been made open-source under the Apache 2 license: https://github.com/cmower/optas. Several examples are included such as optimization-based planning, differential inverse kinematics, multi-robot optimization, and MPC.

REFERENCES

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

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[3] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494, DOI: 10.1109/ROBOT.2009.5152817.

[4] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014, DOI: 10.1177/0278364914528132.

[5] 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, DOI: 10.1177/0278364913506757.

[6] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, Mar 2016, DOI: 10.1007/s10514-015-9479-3.

[7] T. Stouraitis, I. Chatzinikolaidis, M. Gienger, and S. Vijayakumar, "Online hybrid motion planning for dyadic collaborative manipulation via bilevel optimization," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1452–1471, 2020.

[8] C. E. Mower, J. Moura, and S. Vijayakumar, "Skill-based shared control," in *Robotics: Science and Systems*, 2021.

[9] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile planar manipulation via trajectory optimization with complementarity constraints," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 970–976.

[10] M. Toussaint, J. Harris, J.-S. Ha, D. Driess, and W. Hönig, "Sequence-of-constraints mpc: Reactive timing-optimal control of sequential manipulation," 2022. [Online]. Available: https://arxiv.org/abs/2203.05390

[11] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar 2006, DOI: 10.1007/s10107-004-0559-y.

[12] R. H. Byrd, J. Nocedal, and R. A. Waltz, "KNITRO: An integrated package for nonlinear optimization," in *Large-scale nonlinear optimization*. Springer, 2006, pp. 35–59.

[13] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An sqp algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, p. 99–131, jan 2005, DOI: 10.1137/S0036144504446096.

[14] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a MoveIt! case study," 2014. [Online]. Available: https://arxiv.org/abs/1404.3785

[15] 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, DOI: 10.1007/s12532-018-0139-4.

[16] C. E. Mower, T. Stouraitis, J. Moura, C. Rauch, L. Yan, N. Z. Behabadi, M. Gienger, T. Vercauteren, C. Bergeles, and S. Vijayakumar, "ROS-PyBullet Interface: A framework for reliable contact simulation and human-robot interaction," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research. PMLR, 2022.

[17] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020, DOI: 10.1007/s12532-020-00179-2.

[18] M. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT: Convex optimization," *Astrophysics Source Code Library*, pp. ascl–2008, 2020.

[19] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[20] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[21] M. J. D. Powell, *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*. Dordrecht: Springer Netherlands, 1994, pp. 51–67, DOI: 10.1007/978-94-015-8330-5_4.

[22] D. Kraft, "A software package for sequential quadratic programming," *Tech. Rep. DFVLR-FB 88-28, DLR German Aerospace Center – Institute for Flight Mechanics, Koln, Germany*, 1988.

[23] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[25] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1540–1546, DOI: 10.1109/IROS.2014.6942760.

[26] M. H. Arbo, E. I. Grøtli, and J. T. Gravdahl, "CASCLIK: Casadi-based closed-loop inverse kinematics," 2019. [Online]. Available: https://arxiv.org/abs/1901.06713

[27] V. Ivan, Y. Yang, W. Merkt, M. P. Camilleri, and S. Vijayakumar, *EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control*. Cham: Springer International Publishing, 2019, pp. 211–240, DOI: 10.1007/978-3-319-91590-6_7.

[28] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[29] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022. [Online]. Available: https://www.gurobi.com

[30] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, DOI: 10.1038/s41592-019-0686-2.

[31] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

[32] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*, 2nd ed. Springer, 2017, iSBN 978-3-319-54413-7.

[33] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 928–935, DOI: 10.1109/HUMANOIDS.2015.7363472.

[34] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985, DOI: 10.1177/027836498500400201.