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 planners and controllers are integrated in many of the approaches for solving complex manipulation tasks. Consider, for example, a robot operating in an unstructured and dynamic environment that, e.g. places an object onto a shelf, or drilling during pedicle screw fixation in surgery (see Fig. 1). In such cases, a planner and controller must account for objectives/constraints like bi-manual coordination, contact constraints between robot-object and object-environment, and be robust to disturbances. Efficient motion planning and fast controllers are an effective way of enabling robots to perform these tasks subject to motion constraints, system dynamics, and changing task objectives.

(a) (b)

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

Sampling-based planners [1] are effective, however, they typically require considerable post-processing (e.g. trajectory smoothing). Optimal planners (i.e. that are provably asymptotically optimal, e.g. RRT*) are promising but inefficient (in terms of computation duration) for solving high-dimensional problems [2].

Gradient-based trajectory optimization (TO) is a key approach in optimal control, and has also been utilized for motion planning. This approach underpins many recent works in robotics for planning and control, e.g. [3], [4], [5], [6], [7], [8], [9], [10]. Given an initialization, optimization finds a locally optimal trajectory, comprised of a stream of state and control commands subject to motion constraints and system dynamics (i.e. equations of motion).

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

To fill this gap, this paper proposes OpTaS, a user-friendly task-specification library for rapid development and deployment of nonlinear optimization-based planning and control approaches such as Model Predictive Control (MPC). The library leverages the symbolic framework of CasADi [15], enabling function derivatives to arbitrary order via automatic differentiation. This is important since some solvers (e.g. SNOPT) utilize the Jacobian and Hessian.

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 work was supported by core funding from the Wellcome/EPSRC [WT203148/Z/16/Z; NS/A000049/1]. 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.

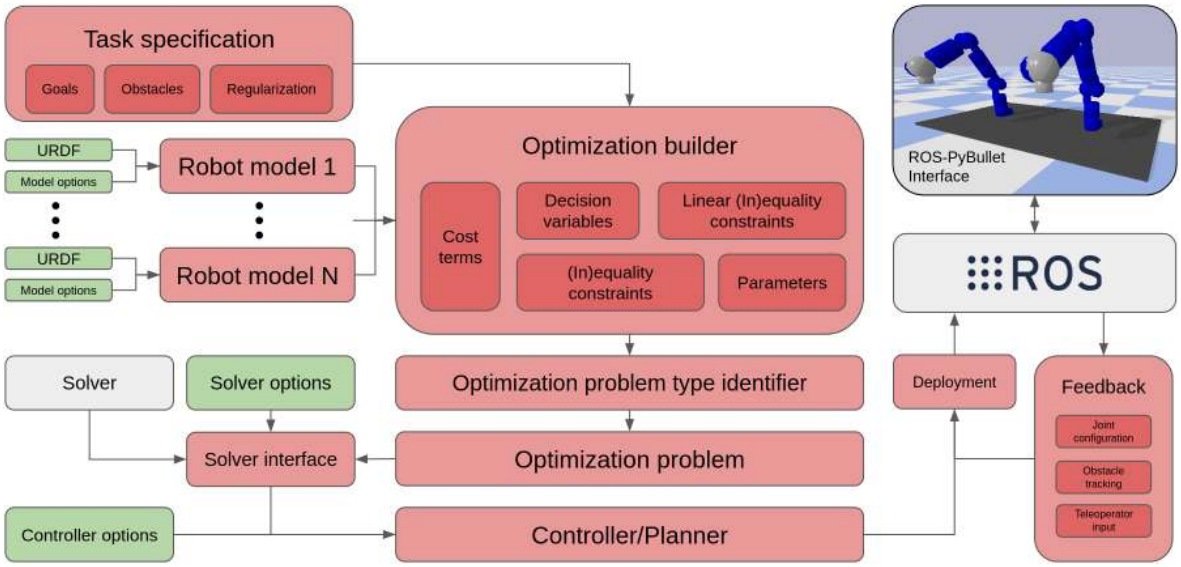


Fig. 2: System overview for the proposed OpTaS library. **Red** highlights the main features of the proposed library. **Green** shows configuration parameter input. **Grey** shows third-party frameworks/libraries. Finally, the image in the top-right corner shows integration with the ROS-PyBullet Interface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

A. Related work

In this section, we review popular optimization solvers and their interfaces. Next, we describe works similar (in formulation) to our proposed library. Finally, we summarize the key differences and highlight our contributions. Table I summarizes alternatives and how they compare to OpTaS.

There are several capable open-source and commercial optimization solvers. First considering quadratic programming, the OSQP method provides a general purpose solver based on the alternating direction method of multipliers [17]. Alternatively, CVXOPT implements a custom interior-point solver [18]. IPOPT implements an interior-point solver for constrained nonlinear optimization. SNOPT provides an interface to an SQP algorithm [13]. KNITRO also solves general mixed-integer programs [12]. Please note that SNOPT and KNITRO are proprietary.

These solvers are often implemented in low-level programming languages such as C, C++, or FORTRAN. However, there are also many interfaces to these methods via higher level languages, such as Python, to make implementation and adoption easier. The SciPy library contains the `optimize` module [19] to interface with low-level routines, e.g. conjugate gradient and BFGS algorithm [20], the Simplex method [21], COBYLA [22], and SLSQP [23]. A requirement when using optimization-based methods is the need for function gradients. Several popular software packages implement automatic differentiation [24], [15], [25]. We leverage the CasADi framework [15] for deriving gradients. Our choice for CasADI is based on the fact that it comes readily integrated with common solvers for optimal control. To the best of our knowledge, JAX and PyTorch are not currently integrated with constrained nonlinear optimization solvers.

Similar to our proposed library are the following packages. The MoveIt package provides the user with specific IK/planning formulations and provides interfaces to solvers for the particular problem [14]. The eTaSL library [26] allows the user to specify custom tasks specifications, but only supports problems formulated as quadratic programs. The CASCLIK library uses CasADi and provides support for constraint-based inverse kinematic controllers [27], to the best of our knowledge they allow optimization in the joint space. We provide joint space, task space optimization and also the ability to simultaneously optimize in the joint/task space. Furthermore, our framework supports optimization of several robots in a single formulation. The EXOTica library allows the user to specify a problem formulation from an XML file [28]. The package, however, requires the user to supply analytic gradients for additional sub-task models.

TABLE I: Comparison between OpTaS and common alternatives in literature.

	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 library, in Python, for rapid development/deployment of TO approaches for multi-robot setups.
- Modeling of the robot kinematics (forward kinematics, geometric Jacobian, etc.), to arbitrary derivative order, given a URDF specification.

[1]Enabled with external pluggins.

- 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 performance of the library (i.e. solver convergence, solution quality) versus existing software packages. Further demonstrations highlight the ease in which nonlinear constrained optimization problems can be set up and deployed in realistic settings.

II. Problem Formulation

We can write an optimal control formulation of a TO or planning problems as

$$\min_{x,u} \text{cost}(x, u; T) \quad \text{subject to} \quad \begin{cases} \dot{x} = f(x, u) \\ x \in \mathbb{X} \\ u \in \mathbb{U} \end{cases} \tag{1}$$

where t denotes time, and $x = x(t) \in \mathbb{R}^{n_x}$ and $u = u(t) \in \mathbb{R}^{n_u}$ denote the states and controls, with T being the time-horizon for the planned trajectory. The scalar function cost : $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ represents the cost function (typically a weighted sum of terms each modeling a certain sub-task), the dot notation denotes a derivative with respect to time (i.e. $\dot{x} \equiv \frac{dx}{dt}$), f represents the system dynamics (equations of motion), 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). Direct optimal control, optimizes for the controls u for a discrete set of time instances, using numerical methods (e.g. Euler or Runge-Kutta), to integrate the system dynamics over the time horizon T [29]. Given an initialization $x^{\text{init}}, u^{\text{init}}$, a locally optimal trajectory x^*, u^* is found by solving (1).

As discussed in Sec. I, many works propose optimization-based approaches for planning and control. These can all be formulated under the same framework, i.e. a TO problem as in (1). The goal of our work is to deliver a library that allows a user to quickly develop and prototype constrained nonlinear TO for multi-robot problems, and deploy them for motion generation. The library includes two types of problems, IK and task-sace TO, and indeed both simultaneously. Common steps, such as transcription that transforms the problem's task-level description into a form accepted by numerical optimization solver routines, should be automated and thus not burden the user. 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

In this section, we describe the main features of the proposed library shown in Fig. 2. The library is completely implemented in the Python programming language. We chose Python because it is simple for beginners but also versatile with many well-developed libraries, and it easily facilitates fast prototyping.

A. Robot model

The robot model (`RobotModel`) provides the kinematic modeling and specifies the time derivative orders required for the optimization problem. The only requirement is a URDF to instantiate the object[2]. A key feature is that we can include several robots in the TO, which is useful for dual arm and whole-body optimization. Additional base frames and end-effector links can be added programatically (for example, when several robots are included the optimization their base frames should be registered within a global coordinate frame).

The `RobotModel` class allows access to data such as: the number of degrees of freedom, the names of the actuated joints, the upper and lower actuated joint limits, and the kinematics model. Furthermore, we provide methods to compute the forward kinematics and geometric Jacobian in any given reference frame. Several methods modeling the kinematics are supplied, given a specification from the user for the base frame and end-effector frame. These methods include: the 4×4 homogeneous transformation matrix, translation position, rotational representations (e.g. Euler angles, quaternions), the geometric and analytical Jacobian. Each of the methods above depend on a joint state (supplied as either a Python list, NumPy array, or CasADi symbolic array).

B. Task model

Several works optimize robot motion in the task space and then compute the IK as a secondary step, e.g. [8], [9]. The task model (`TaskModel`) provides a representation for any arbitrary trajectory. For example, the three dimensional position trajectory of an end-effector. In the same way as the robot model, the time derivatives can be specified in the interface an arbitrary order.

C. Optimization builder

This section introduces and describes the optimization builder class (`OptimizationBuilder`). The purpose of this class is to aid the user to easily setup a TO problem, and then automatically build an optimization problem model (Sec. III-D) that interfaces with a solver interface (Sec. III-E). The development cycle consists in specifying the task (i.e. decision variables, parameters, cost function, and constraints) using intuitive syntax and symbolic variables. Then, the builder creates an optimization problem class, which interfaces with several solvers.

D. Optimization problem model

The standard TO is stated in (1). This task/problem is specified by the optimization builder class in intuitive syntax for the user. Transcribing the problem to a form that can be solved by off-the-shelf solvers is non-trivial. The output of the optimization builder method `build` is an optimization problem model that allows us to interface with several solvers.

[2]http://wiki.ros.org/urdf

The most general optimization problem that is modeled by OpTaS is given by

$$X^* = \arg\min_X \ f(X;P) \tag{2a}$$

subject to

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

$$a(X;P) = A(P)X + b(P) = 0 \tag{2c}$$

$$g(X;P) \geq 0 \tag{2d}$$

$$h(X;P) = 0 \tag{2e}$$

where $X = [vec(x)^T, vec(u)^T]^T \in \mathbb{R}^{nx}$ is the decision variable array such that x, u are as defined in (1) and $vec(\cdot)$ is a function that returns its input as a 1-dimensional vector, $P \in \mathbb{R}^{nP}$ is the vectorized parameters, $f : \mathbb{R}^{nx} \to \mathbb{R}$ denotes the objective function, $k : \mathbb{R}^{nx} \to \mathbb{R}^{nk}$ denotes the linear inequality constraints, $a : \mathbb{R}^{nx} \to \mathbb{R}^{na}$ denotes the linear equality constraints, $g : \mathbb{R}^{nx} \to \mathbb{R}^{ng}$ denotes the nonlinear inequality constraints, and $h : \mathbb{R}^{nx} \to \mathbb{R}^{nh}$ denotes the nonlinear equality constraints. The decision variables X are all the joint states and other variables specified by the user stacked into a single vector. Similarly for the parameters, cost terms, and constraints. Vectorization is made possible by the `SXContainer` data structure implemented in the `sx_container` module. This data structure enables automatic transcription of the TO problem specified in (1) into the form (2).

Of course, not all task specifications will require definitions for each of the functions in (2). Depending on the structure of the objective function and constraints, the required time budget, and accuracy, some solvers will be more appropriate for solving (2). For example, a quadratic programming solver that only handles linear constraints (e.g. OSQP [17]) is unsuitable for solving a problem with nonlinear objective function and nonlinear constraints. The build process automatically identifies the optimization problem type, exposing only the relevant solvers. Several problem types are available to the user: unconstrained quadratic cost, linearly constrained with quadratic cost, nonlinear constrained with quadratic cost, unconstrained with nonlinear cost, linearly constrained with nonlinear cost, nonlinear cost and constraints.

1) 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, **(v)** a Boolean describing the alignment of the derivatives (Fig. 3), and **(vi)** a Boolean indicating whether to optimize time steps.

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. 3.

In addition, the user can also optimize the time-steps between each state. The time derivatives can be integrated

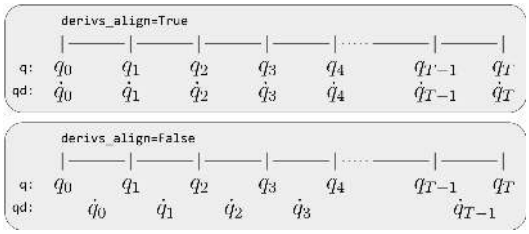


Fig. 3: Joint state alignment with time. User supplies `derivs_align` that specifies how joint state time derivatives should be aligned.

over time, e.g. $q_{t+1} = q_t + \delta\tau_t \dot{q}_t$, where $\delta\tau_t$ is an increment in time. When `optimize_time=True`, then each $\delta\tau_t$ is included as decision variables in the optimal control problem.

2) Decision variables and parameters: Decision variables are specified in the optimization builder class interface for the joint space, task space, and time steps. Each group of variables is given a unique label and can be retrieved using the `get_model_state` method. States are retrieved by specifying a robot name or task name, the required time index, and the time derivative order required. Additional decision variables can be included in the problem by using the `add_decision_variables` method given a unique name and dimension.

Parameters for the problem (e.g. safe distances) can be specified using the `add_parameter` method. To specify a new parameter, a unique name and dimension is required.

3) Cost and constraint functions: The cost function in (1) is assumed to be made up of several cost terms, i.e.

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

where $c_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is an individual cost term modeling a specific sub-task. For example, let us define the cost terms $c_0 = \|\psi(x_T) - \psi^*\|^2$ and $c_1 = \lambda \int_0^T \ \|u\|^2 \ dt$ (note, discretization is implicit in this formulation) where $\psi : \mathbb{R}^{n_x} \to \mathbb{R}^3$ is a function for the forward kinematics position (note, this can be provided by the robot model class as described in Sec. III-A), $\psi^* \in \mathbb{R}^3$ is a goal task space position, and $0 < \lambda \in \mathbb{R}$ is a scaling term used to weight the relative importance of one constraint against the other. Thus, c_0 describes an ideal state for the final state, and c_1 encourages trajectories with minimal control signals (e.g. minimize joint velocities). Each cost term is added to the problem using the `add_cost_term` method; the `build` sequence ensures each term is added to the objective function.

Several constraints can be added to the optimization problem by using the `add_equality_constraint` and `add_leq_inequality_constraint` methods that add equality and inequality constraints respectively. When the constraints are added to the problem, they are first checked to see if they are linear constraints with respect to the decision variables. This functionality allows the library to differentiate between linear and nonlinear constraints.

Additionally, OpTaS offers several methods that provide an implementation for common constraints, as, for example,

joint position/velocity limits and time-integration for the system dynamics f (e.g joint velocities can be integrated to positions).

E. Solver interface

OpTaS provides interfaces to solvers (open-source and commercial) that interface with CasADi [15] (such as IPOPT [11]), SNOPT [13], KNITRO [12], and Gurobi [30]), the Scipy `minimize` method [19], OSQP [17], and CVXOPT [18].

1) Initialization of solver: When the solver is initialized, several variables are setup and the optimization problem object is set as a class attribute. The user must then call the `setup` method - that itself is an interface to the solver initialization that the user has chosen. The requirement of this method is to setup the interface for the specific solver; relevant solver parameters are passed to the interface at this stage.

2) Resetting the interface: When using the solver as a controller, it is expected that the solver should be called more than once. In the case for feedback controllers or controllers with parameterized constraints (e.g. obstacles), this requires a way to reset the problem parameters. Furthermore, the initial seed for the optimizer is often required to be reset at each control loop cycle. To reset the initial seed and problem parameters the user calls `reset_initial_seed`, and `reset_parameters`, respectively. Both the initial seed and parameters are initialized by giving the name of the variables. The required vectorization is internally performed by the solver utilizing features of the `SXContainer` data structure. Note, if any decision variables or parameters are not specified in the reset methods then they automatically default to zero. This enables warm-starting the optimization routine, e.g. with the solution of the previous time-step problem.

3) Solving an optimization problem: The optimization problem is solved by calling the `solve` method. This method passes the optimization problem to the desired solver. The resulting data from the solver is collected and transformed back into the state trajectory for each robot. A method is provided, named `interpolate`, is used to interpolate the computed trajectories across time. Additionally, the method `stats` retrieves available optimization statistics (e.g. number of iterations).

4) Extensible solver interface: The solver interface has been implemented to allow for extensibility, i.e. additional optimization solvers can be easily integrated into the framework. When a user would like to include a new solver interface, they must create a new class that inherits from the `Solver` class. In their sub-class definition they must implement three methods: (i) `setup` which (as described above) initializes the solver interface, (ii) `_solve` that calls the solver and returns the optimized variable X^*, and (iii) `stats` that returns any statistics from the solver.

F. Additional features

Support for integration with ROS [31] is provided out-of-the-box. The ROS node provided is integrated with the ROS-PyBullet Interface [16] so the publishers/subscribers can connect a robot in the optimization problem with a robot simulated in PyBullet.

In addition, we provide a port of the `spatialmath` library by Corke [32] that supports CasADi variables. This library defines methods for manipulating homogeneous transformation matrices, quaternions, Euler angles, etc. using CasADi symbolic variables.

```python
import optas

# Setup robot and optimization builder
T = 100 # number of time steps in trajectory
urdf = '/path/to/robot.urdf'
r = optas.RobotModel(urdf, time_deriv=[0, 1])
n = r.get_name()
b = optas.OptimizationBuilder(T=T, robots=[r])

# Retrieve variables and setup parameters
q0 = b.get_model_state(n, t=0)
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.
r = b.add_parameter('r')  # obstacle radius
dt = b.add_parameter('dt') # time step

# Forward kinematics
p = r.get_global_link_position(tip, qT)

# Cost and constraints
b.add_cost_term('c', optas.sumsqr(p - pg))
b.integrate_model_states(
    n, time_deriv=1, dt=dt)
b.add_equality_constraint('init', q0, qc)
for t in range(T):
    b.add_leq_inequality_constraint(
        optas.sumsqr(p - o), r**2)

# Build optimization problem and setup solver
solver = optas.CasADiSolver(
    b.build()).setup('ipopt')
```

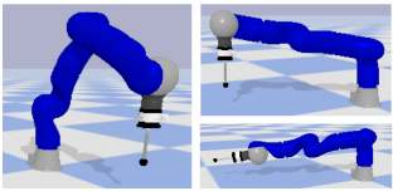
Fig. 4: Example code for TO described in Section IV.

IV. Code Example

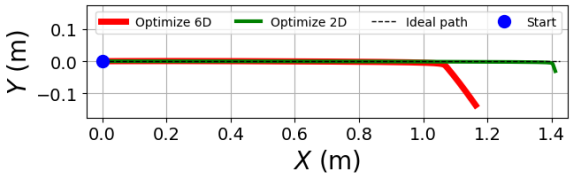
In this section, we describe a common TO problem and give the code that models the problem. We aim to highlight how straightforward it is to setup a problem.

Consider a serial link manipulator, and goal to find a collision-free plan over time horizon T to a goal end-effector position p_g given a starting configuration q_c. A single spherical collision is represented by a position o and radius r. The robot configuration q_t represent states, and the velocities $\dot{q}_t$ are controls.

The cost function is given by $\|p(q_T) - p_g\|^2$ where p is the position of the end-effector given by the forward kinematics. We solve the problem by minimizing the cost function subject to the constraints: (i) initial configuration, $q_0 = q_c$, (ii) joint limits $q^- \leq q_t \leq q^+$, and (iii) obstacle avoidance, $\|p(q_t) - o\|^2 \geq r^2$. The system dynamics is represented by several equality constraints $q_{t+1} = q_t + \delta t \dot{q}_t$ that can be specified by methods already in-built into OpTaS. The code for the TO problem above, is shown in Fig. 4.



(a)



(b)

Fig. 5: Comparison of end-effector task space trajectories computed using two different formulations. (a) Shows the start (left), and final configurations (right) for the robot under each approach. (b) Plots the end-effector position trajectory two dimensions.

V. Experiments

A. Optimization along custom dimensions

Popular solvers, such as TracIK [33], require the user to provide a 6D pose as the task space goal. Whilst this is applicable to several robotics problems (e.g. pick-and-place) it may not be necessary to optimize each task space dimension (e.g. spraying applications does not require optimization in the roll angular direction). Furthermore, optimizing in more dimensions than necessary may be disadvantageous.

OpTaS can optimize or neglect any desired task space dimension. This can have certain advantages, for example increasing the robot workspace. Consider a non-prehensile pushing task along the plane, optimizing the full 6D pose may not be ideal since the task is two dimensional. By optimizing in the two dimensional plane and specifying boundary constraints on the third linear spatial dimension, increases the robots workspace.

We setup a tracking experiment in OpTaS using a simulated Kuka LWR robot arm to compare the two cases: (i) optimize the full 6D pose, and (ii) optimize 2D linear position. The robot is given an initial configuration (Fig. 5a left) and the task is to move the end-effector with velocity of constant magnitude and direction in the 2D plane. The end configuration for each approach is shown in Fig. 5a right and the end-effector trajectories are shown in Fig. 5b. We see that the 2D optimization problem is able to reach a greater distance, highlighting that the robot workspace is increased.

B. Performance comparison

In this section, we demonstrate that OpTaS can formulate similar problems and compare its performance to alternatives. First, we model, with OpTaS, the same problem as used in TracIK [33] and in addition we also model the problem using EXOTica [28]. The Scipy SLSQP solver [23] was used for OpTaS and EXOTica. With same Kuka LWR robot arm in the previous experiment, we setup a task where the robot must track a figure-of-eight motion in task space (Fig. 6) and record the CPU time for the solver duration at each control loop cycle. The results are shown in Fig. 7a. TracIK is the fastest (0.049 ± 0.035ms), which is expected since it is optimized for a specific problem formulation. We see that OpTaS (2.608 ± 0.239ms) is faster than EXOTica (3.694 ± 0.300ms)

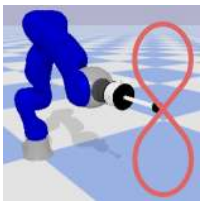
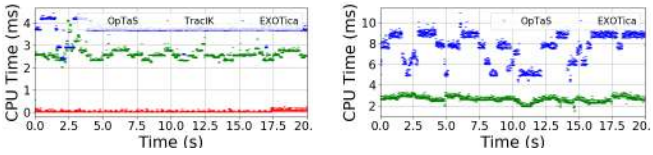


Fig. 6: Figure-of-eight trajectory tracked by the Kuka LWR.



(a) (b)

Fig. 7: 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.

A second experiment, using the same setup as before, was performed comparing the performance of OpTaS against EXOTica with an additional cost term to maximize manipulability [34]. The results are shown in Fig. 7b. Despite using the same formulation and solver, OpTaS (2.650 ± 0.270ms) achieved better performance than EXOTica (7.640 ± 1.404ms). Without extensive profiling it is difficult to precisely explain this difference. However, EXOTica requires the user to supply analytical gradients for sub-tasks (called *task maps* in the EXOTica documentation). EXOTica does not provide the gradients for the manipulability task, and thus falls-back to using the finite difference method to estimate the gradient - this can can be slow to compute.

VI. Conclusions

In this paper, we have proposed OpTaS: an optimization-based task tpecification Python library for TO and MPC. OpTaS allows a user to setup a constrained nonlinear programs for custom problem formulations and has been shown to perform well against alternatives. Parameterization enables programs to act as feedback controllers, motion planners, and benchmark problem formulations and solvers.

We hope OpTaS will be used by researchers, students, and industry to facilitate the development of control and motion planning algorithms. The code base is easily installed via `pip` and has been made open-source under the Apache 2 license: https://github.com/cmower/optas.

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.

[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.

[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.

[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. [Online]. Available: https://doi.org/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.

[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. [Online]. Available: https://doi.org/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. [Online]. Available: https://doi.org/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.

[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.

[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 *[to appear] 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. [Online]. Available: https://doi.org/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] 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.

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

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

[22] 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. [Online]. Available: https://doi.org/10.1007/978-94-015-8330-5_4

[23] 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.

[24] 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

[25] 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.

[26] 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.

[27] 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

[28] 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. [Online]. Available: https://doi.org/10.1007/978-3-319-91590-6_7

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

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

[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.

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