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

International Conference on Robotics and Automation (ICRA) 2023

Christopher E. Mower[1], João Moura[2], Nazanin Zamani Behabadi, Sethu Vijayakumar[2,3], Tom Vercauteren[1], Christos Bergeles[1],

[1]School of Biomedical Engineering & Imaging Sciences, King's College London, London, UK, [2]School of Informatics, The University of Edinburgh, Edinburgh, UK, [3]The Alan Turing Institute, London, UK

Try out OpTaS

Hosted on GitHub

Easily installed using pip.

SCAN ME

Code snippet

Goal: (i) find a joint space trajectory, (ii) from a known initial configuration, (iii) where the end-effector must avoid an obstacle, and (iv) reach a given position (v) with minimal joint velocity.

```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
dQ = b.get_model_states(n, time_deriv=1) # jnt vel traj
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
dur = b.add_parameter("dur")  # traj duration
dt = dur / float(T - 1)  # time step

p = r.get_global_link_position(tip, qT)  # FK
b.add_cost_term("goal", optas.sumsqr(p - pg))
b.add_cost_term("min_vel", 0.01*optas.sumsqr(dQ))
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")

# Solver is setup using solver.reset_initial_seed(..)
# and solver.reset_parameters(..). The solver is
# called using solution = solver.solve().
```

Overview

(1) Task specification

User provides task model through user-friendly syntax in Python

$$x^*, u^* = \arg\min_{x,u} \sum_i \text{cost}(x, u)$$

$$\text{subject to} \begin{cases} \dot{x} = f(x, u) \\ x \in \mathbb{X} \\ u \in \mathbb{U} \end{cases}$$

(2) Optimization builder

The task model is converted to a compatible format with multiple solvers.

$$X^* = \arg\min_X f(X)$$

$$\text{subject to} \begin{cases} MX + c \geq 0 \\ g(X) \geq 0 \\ h(X) = 0 \end{cases}$$

(3) Solver interface

Optimization problem is interfaced with open-source/commercial solvers for quadratic/nonlinear optimal control, such as

SNOPT	IPOPT	KNITRO
Gurobi	SciPy	OSQP
BOMIN	CVXOPT	qpOASES

Proposed framework

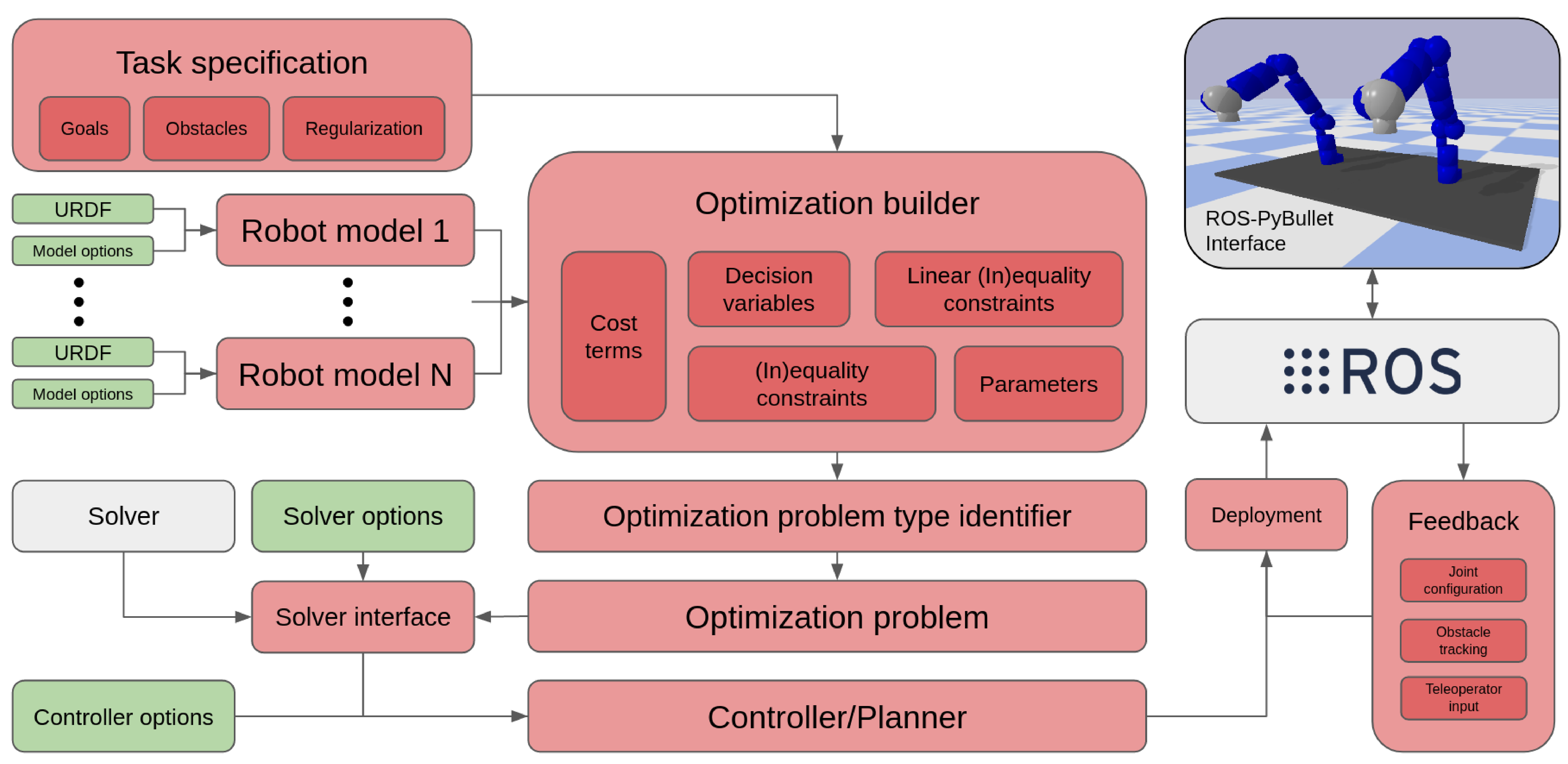

Figure: System overview for the proposed OpTaS library. **Red** highlights the main features of the proposed library. **Green** shows configuration parameter input. **Grey** shows third-party frameworks/libraries. Finally, the image in the top-right corner shows integration with the ROS-PyBullet Interface.

Contributions

- A **task-specification Python library** for rapid development/deployment of trajectory optimization approaches for **multi-robot setups**.
- **Modeling of the robot kinematics** (end-effector transform, unit-quaternion, Geometric/Analytical Jacobian in any base frame) to **arbitrary derivative order**.
- **Easily reformulate optimal control problems**, optimize in **specific** task/joint dimensions, and define **parameterized constraints** for online modification of the optimization problem.
- Analysis comparing the performance (i.e. solver convergence, solution quality) versus existing packages. Several demonstrations highlight the ease in which NLP problems can be deployed in realistic settings.

Alternatives?

	Language	EndPose	Traj	MPC	Solver	AutoDiff	ROS	RF
OpTaS	Py	✓	✓	✓	QP/NLP	✓	✓	✓
EXOTica	Py/C++	✓	✓	✗	QP/NLP	✗	✓	✓
MoveIt	Py/C++	✓	✓	✗	QP	✗	✓	✗
TracIK	Py/C++	✓	✗	✗	QP	✗	✓	✗
RBDL	Py/C++	✓	✗	✗	QP	✗	✗	✗
eTaSL	C++	✓	✗	✗	QP	✓	✗*	✓
OpenRAVE	Py	✗	✓	✗	QP	✗	✓	✗

*Enabled through external plugins.

Key

MPC: Model Predictive Control
ROS: Robot Operating System
AutoDiff: Automatic differentiation
Py: Python
Traj: Trajectory
RF: Reformulation

Performance comparison:

- Comparable performance in terms of CPU time for solver duration compared against alternatives.
- Since OpTaS enables optimization in specific dimensions, we show this increases the robot workspace.

Connect with us

Code: github.com/cmower/optas
Email: christopher.mower@kcl.ac.uk
RViM: rvim.online
SLMC: web.inf.ed.ac.uk/slmc
FAROS: h2020faros.eu
CAI4CAI: cai4cai.ml
Video: youtu.be/gCMNOenFngU