Optico:

A framework for model-based optimization with MuJoCo physics

Emo Todorov

Roboti LLC
University of Washington

Overview

Goal: define optimization problems with respect to MuJoCo physics, and solve them efficiently in a unified framework

PERFORMANCE CRITERIA	DECISION VARIABLES		
	control parameters	**state parameters**	**model parameters**
movement costs	optimal control	motion synthesis	mechanism design
model-data mismatch	imitation learning	state estimation	system identification

Audience: researchers working on any of the above problems

researchers in other areas who need the above to be solved
robotics and automation engineers
animators and game developers
students and educators

Run

Analogy:

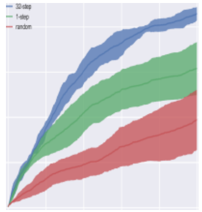
Optico platform ↔ MATLAB

Optico optimizers ↔ MathWorks toolboxes

Optico extensions ↔ contributed toolboxes

Productivity tool

setting up the official run
~ months



"official run"
~ hours

RL environment

(*OpenAI Gym, DM Control, Lyceum*)
Python and Julia wrappers for MuJoCo+Optico

Model-based optimization SDK

analytical derivatives of physics, costs, networks;
Gauss-Newton; inverse dynamics; contact-aware

Optico

Dynamic language for optimization

mid-level abstraction between programming and clicking "Run"; optimization-related primitives

Content creation

(*Solidworks, Blender, Final Cut*)
UI for optimization, visualization and editing

Optimization framework

desired trajectory
sensor data
goal state
zero

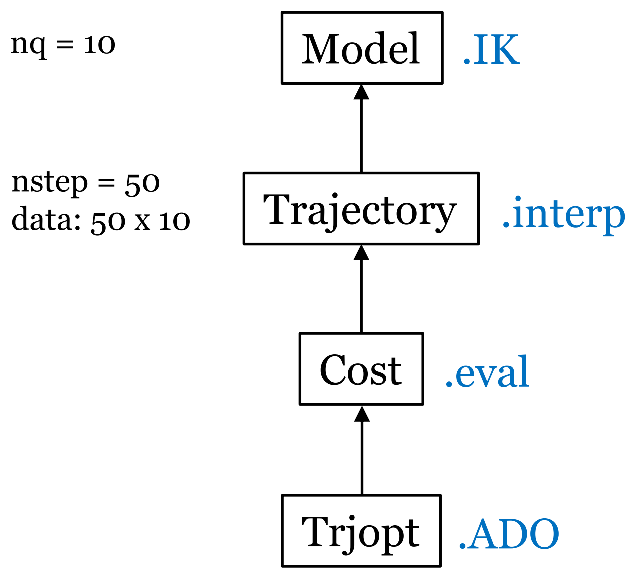
initial states	$\{x_0^n\}$	expectation and averaging
model variants	$\{\omega^n\}$	domain randomization, model optimization
trajectories	$x_{t+1}^n = f\left(x_t^n, u_t^n, \omega^n\right)$	also inverse dynamics
cost	$\ell\left(x, u, t\right) = \sum_i w_i\left(t\right) \text{loss}_i \underbrace{\left(\text{scale}_i\left(x\right)\left(\text{feature}_i\left(x, u\right) - \text{reference}_i\left(t\right)\right)\right)}_{\text{also used as network inputs}}$	
policy	$u = \pi\left(x, \theta\right)$	
performance	$L\left(\theta\right) = \sum_{n,t} \ell\left(x_t^n, u_t^n, t\right)$	also trajectory optimization
value	$v\left(x, \phi\right)$	
residual	$R\left(\phi\right) = \sum_{n,t} \left(\ell\left(x_t^n, u_t^n, t\right) + v\left(x_{t+1}^n, \phi\right) - v\left(x_t^n, \phi\right)\right)^2$	

$\nabla L\left(\theta\right)$ and $\nabla R\left(\phi\right)$ are computed analytically

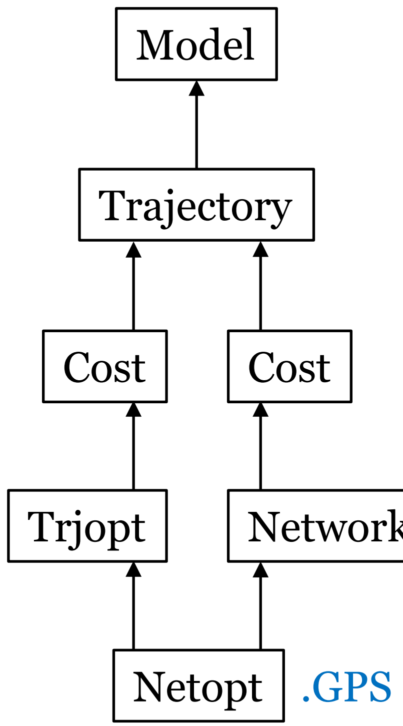
Dynamic language for optimization

*Language is a formal system of symbols governed by grammatical rules of **combination** to communicate meaning.*

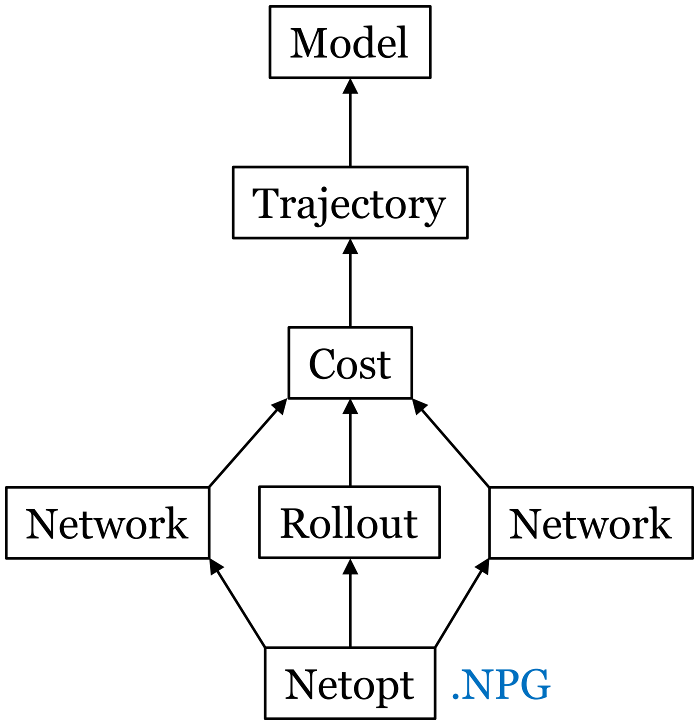
Object-oriented spreadsheet



Guided Policy Search



Natural Policy Gradient

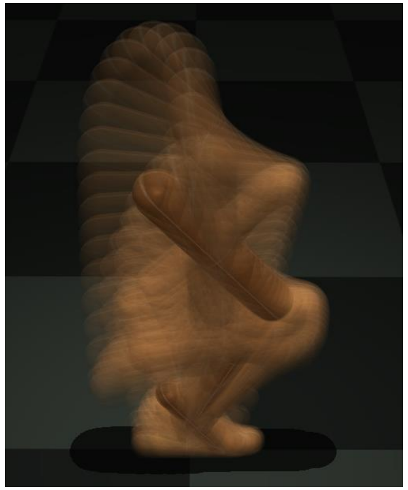


Optico workspace file format (OPF)

```xml
<optico>
  <model name="hopper" nstate="2" file="hopper.xml"/>

  <trajectory name="hop" nstep="200" cycle="true">
    <dependence model="hopper"/>
    <pin step="0" stateid="0"/>
    <pin step="100" stateid="1"/>
  </trajectory>

  <cost name="hop">
    <dependence trajectory="hop"/>
    <term name="actuation" weight="1000" loss="equality" scale="nullspace" residual="qfrc"/>
    <term name="energy" weight="0.1" loss="l2" scale="minv" residual="qfrc"/>
    <term name="smooth" weight="0.05" loss="l2" residual="qacc"/>
  </cost>

  <trjopt name="hop">
    <dependence cost="hop"/>
    <ADO maxiter="300" diagadd="0.2"/>
  </trjopt>
</optico>
```

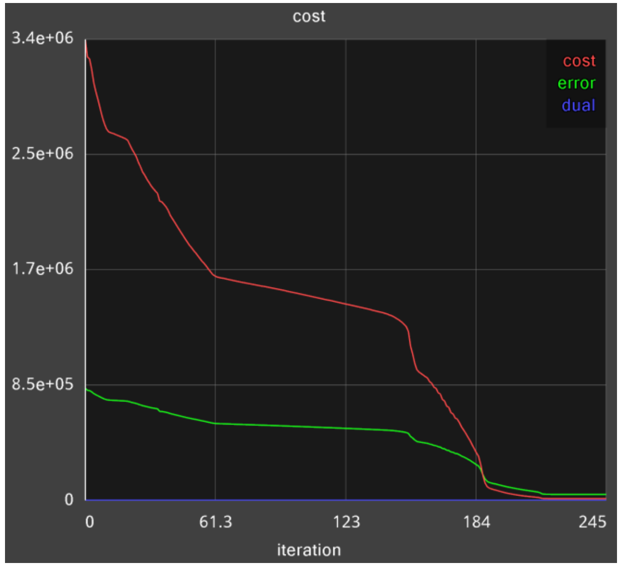
Preliminary timing tests

200-step hopper trajectory
10-core desktop processor

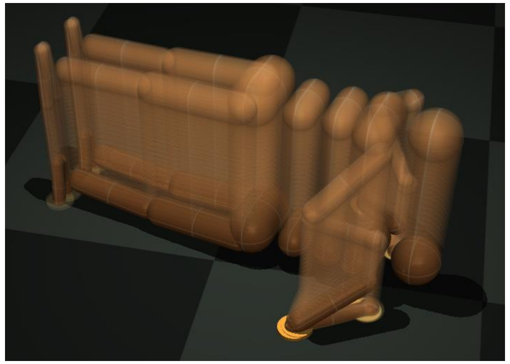


CPU time (20 steps per core)

per iteration	**2.9** ms
total time	700 ms
total iterations	245

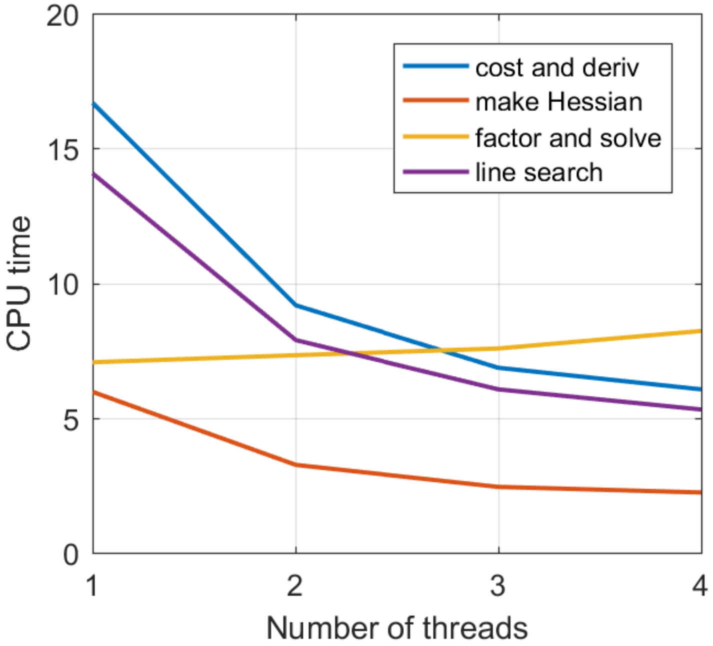


200-step humanoid trajectory
4-core laptop processor

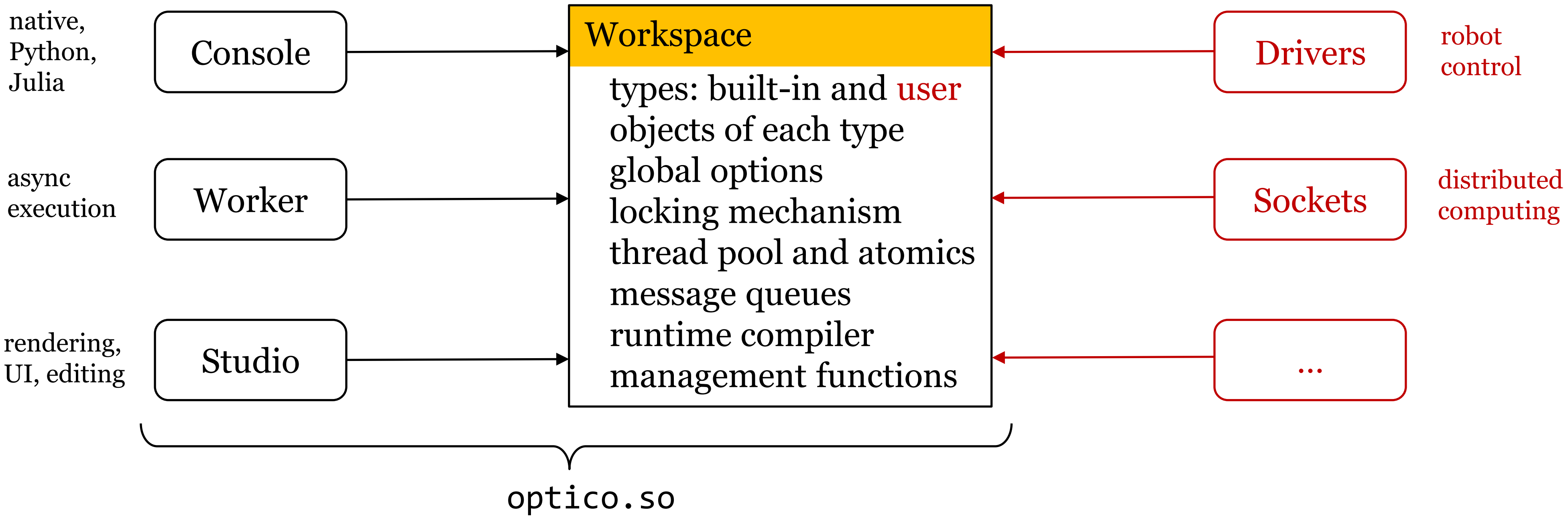


CPU time (50 steps per core)

cost only:	**1.5** ms
cost + analytical:	**7.5** ms
cost + one-sided FD:	**48.0** ms



Client-workspace architecture




```
#include "optico.h"

void main(void)
{
	mj_activate("mjkey.txt");

	// opw_addType() to add user types
	// opw_startClient() to start user clients

	opw_startClient(op_clientWorker, "worker");
	opw_startClient(op_clientStudio, "studio");
	op_clientConsole();
}
```

```
void op_clientWorker(void)
{
	opw_addQueue("worker");

	opWorkspace* W = opw_getWorkspace();
	while( !W->option.terminate )
	{
		opMessage msg;
		if( opw_wait("worker", 10, &msg) )
			opw_exec(msg.word[1], msg.word[2], msg.word[3]);
	}
}
```

User clients and types

Thread-safe clients:

```
opw_lockAll()
```
structural changes
exclusive access

```
opw_lockWrite()
```
content changes
single client can write

```
opw_lockRead()
```
no changes
many clients can read

Top-level API is automatic:

```
opw_addQueue()
opw_exec()
opw_load()
```

Workspace types:

```
typedef struct opBase
{
   int type;
   char name[opMAXNAME];
   opBase* previous;
   opBase* next;
   // ...
};

typedef struct opwRollout
{
   opBase base;
   opRolloutSpec source, spec;
   opRolloutOpt opt;

   opArray cost;
   opArray qpos;
   // ...
};

typedef struct opArray
{
   int ndim;
   int size[opMAXDIM];
   int ntotal;
   mjtNum* buf;
};
```

Adding types:

```
opRes opw_addType
(
   const char* type, int nfield, int nfieldsave,
   int size, int sizespec, int sizeopt,
   const opUserDef* userspec, const opUserDef* useropt,
   void (*fnInit)(opBase* obj),
   opRes (*fnCompile)(opBase* obj),
   void (*fnDecompile)(opBase* obj),
   void (*fnInfo)(const opBase* obj, char* text, int textsz),
   const char* (*fnFieldInfo)(int fid, int* ftype),
   void* (*fnFieldPtr)(opBase* obj, int fid),
   opRes (*fnExec)(opBase* obj, int method)
);
```

Potential impact on future research directions

more derivatives, less sampling

more sophisticated optimization methods

more trajectory optimization and demonstration
- model-predictive control
- training data for policies

more applications to physical robots
- state estimation and feedback control loop
- system identification, domain randomization

more comprehensive benchmarks
- standard cost function machinery
- same problem format for different optimization styles

more creative learning algorithms by combining primitives
- ML is often about combining existing ideas and scaling them