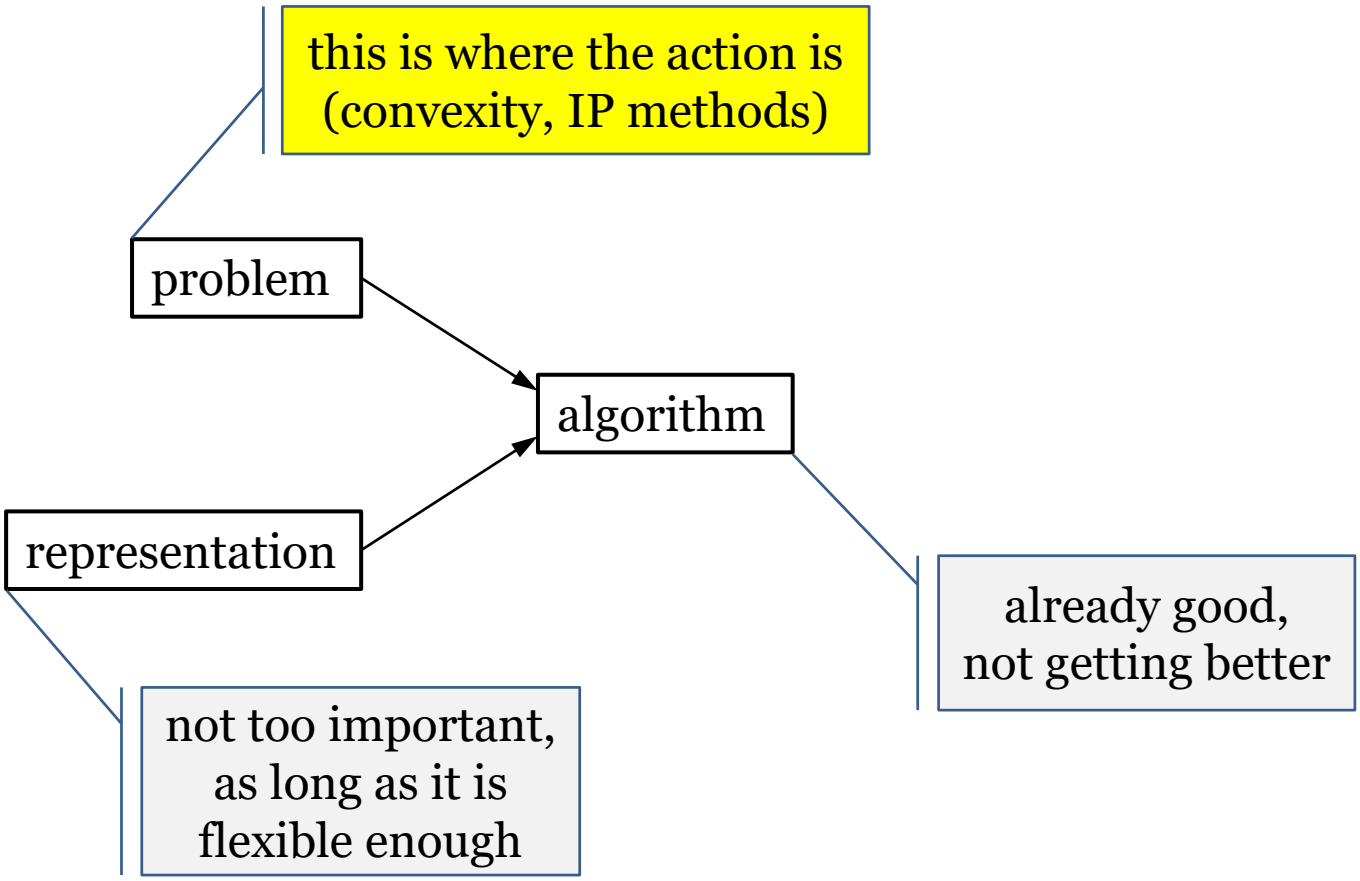
Using costs as meta-controls

(RL terminology: Using rewards as meta-actions)

Emo Todorov

University of Washington
Roboti LLC

Optimization landscape

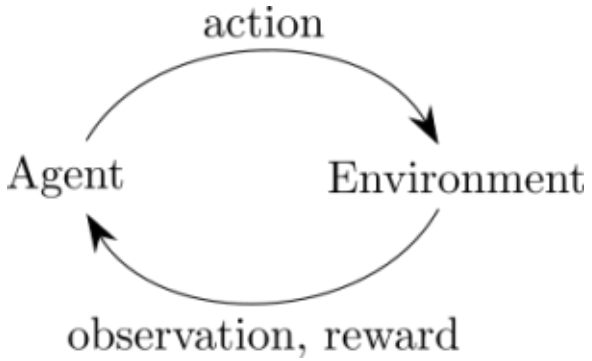


we should (re) define the problem so as to help the algorithm

Where does reward come from?

In an effort to be model-free, RL treats reward as being generated externally.

This requires an oracle.

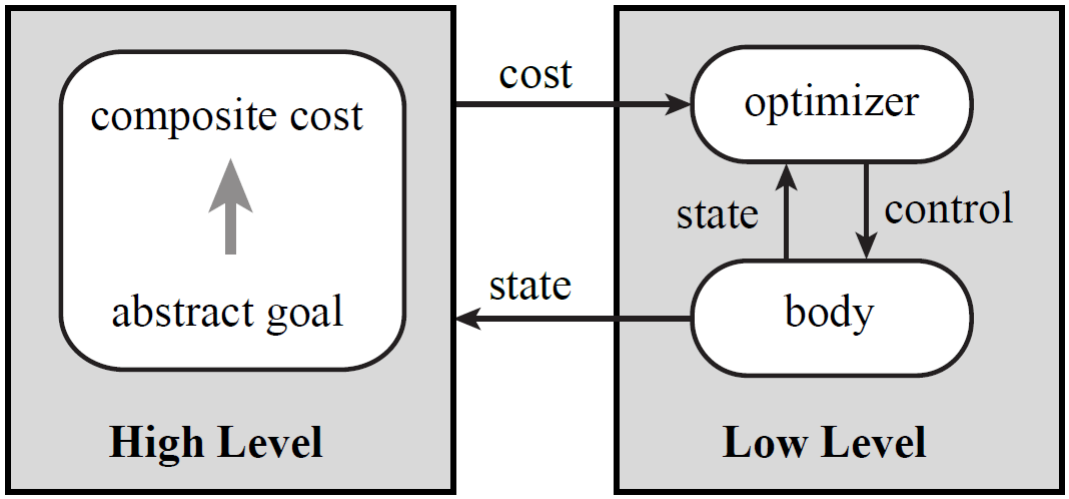


Brains have elaborate neural mechanisms to recognize situations that are good for the organism, and deliver reward internally. This is likely because external rewards are too sparse (in time) to be useful for learning – consider a predator chasing pray and only getting reward upon eating it.

Similarly, robots must have the ability to compute reward internally, and this requires a perception system.
Demos suggesting otherwise involve tasks where success can be measured by simple sensors, however such tasks can be performed by equally simple actuators, in which case we don't even need robots.

Reward used for learning does not come from the environment.
A real-world agent needs to compute its own reward internally.
This computation should approximate some external reward, but on a longer timescale (evolution for brains, design for robots).

Costs as meta-controls

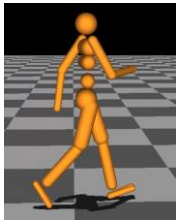


Abstract goal: leave the room

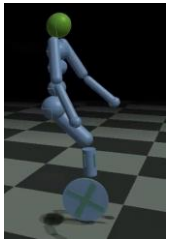
Composite cost:
- minimize distance to target (outside room)
- minimize energy
- avoid collisions
- if sitting in chair, get up
- if up, walk with symmetric gait
- if near door and door closed, open door

Cost terms that make optimization work

height adjustment,
collisions avoidance



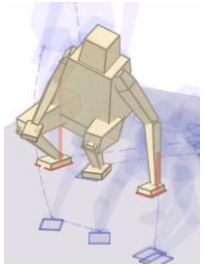
limit cycle,
gait symmetry



multiple
stability costs

sequence of
sub-goals

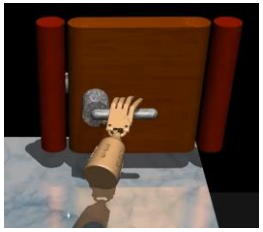
KL divergence



CIO
(and others)

ensemble

proximity,
synergy



imitation

Abstract goal → Composite cost → Optimizer

Instead of pretending that the goal and the cost are the same thing, we should exploit composite cost functions to simplify the problem.

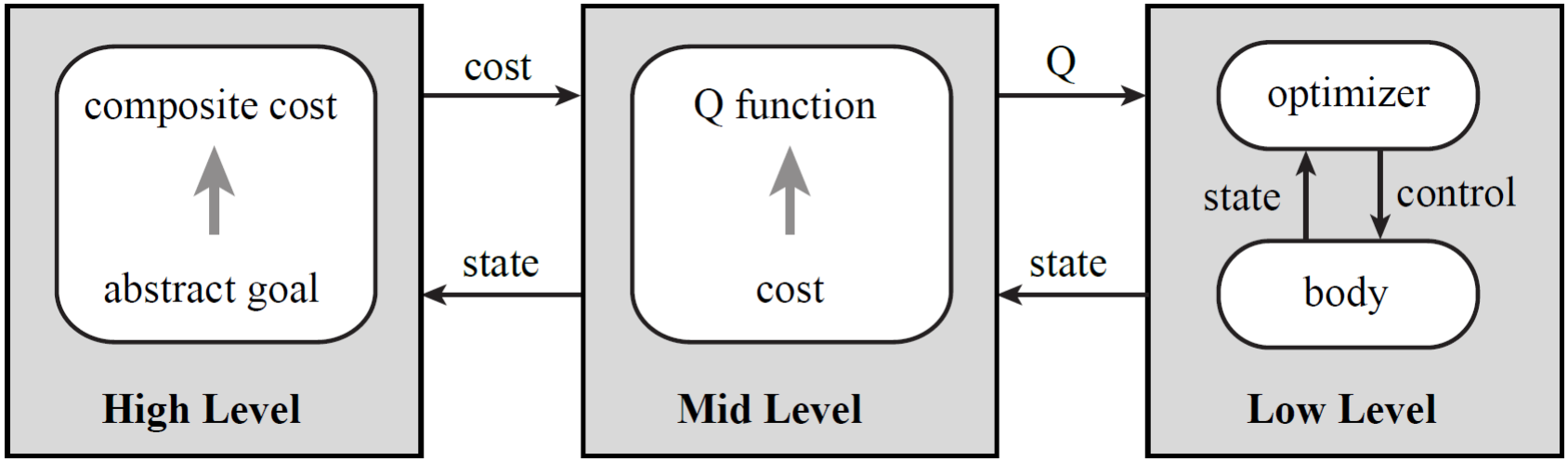
Language for cost functions

$$
\begin{aligned}
\text{term}_i\,(\mathbf{x},\mathbf{u}) &= \text{LOSS}_i\,(\text{SCALE}_i\,(\mathbf{x})\;\;(\text{FEATURE}_i\,(\mathbf{x},\mathbf{u}) - \text{reference}_i)) \\
\text{function}_j\,(\mathbf{x},\mathbf{u}) &= \sum_i w_{ji}\;\;\text{term}_i\,(\mathbf{x},\mathbf{u}) \\
\text{mix}_j\,(\mathbf{x},\mathbf{u}) &= \sum_i m_{ji}\;\;\text{term}_i\,(\mathbf{x},\mathbf{u}) \\
\text{weight}_j\,(\mathbf{x},\mathbf{u}) &= \frac{\exp\left(-\text{mix}_j\,(\mathbf{x},\mathbf{u})\right)}{\sum_k \exp\left(-\text{mix}_k\,(\mathbf{x},\mathbf{u})\right)}
\end{aligned}
$$

$$
\text{cost}\,(\mathbf{x},\mathbf{u}) = \sum_j \text{weight}_j\,(\mathbf{x},\mathbf{u})\;\;\text{function}_j\,(\mathbf{x},\mathbf{u})
$$

This resembles a mixture-of-experts, but here we use it to represent the problem instead of the solution.

Q functions as meta-controls



MuJoCo physics

q	configuration
v	velocity
τ	applied force
$c\,(q, v)$	internal force
$M\,(q)$	inertia matrix
$J\,(q)$	constraint Jacobian
$\lambda\,(q, v, \tau/\dot{v})$	constraint force

Forward dynamics: convex optimization

$$\dot{v} = \arg\min_{a} \left\| a + M^{-1}\,(c - \tau) \right\|_M^2 + s\,(Ja - r)$$

Inverse dynamics: analytical solution

$$\tau = M\dot{v} + c + J^T \nabla s\,(J\dot{v} - r)\,, \quad \lambda = -\nabla s$$

Adding task costs to the physics cost

Physics:

$$\dot{v}(\tau) = \arg\min_{a} \{\text{physics}(a, \tau)\} \text{ s.t. } \text{constraints}(a, \tau)$$

Control, non-physical (QP, CIO):

$$\dot{v}, \tau = \arg\min_{a,t} \{\text{physics}(a, t) + \text{task}(a, t)\} \text{ s.t. } \text{constraints}(a, t)$$

Control, physical:

$$\dot{v}, \tau = \arg\min_{a,t} \{\text{task}(a, t)\} \text{ s.t.}$$
$$a = \arg\min_{b} \{\text{physics}(b, t)\} \text{ s.t. } \text{constraints}(b, t)$$

Can we avoid nested optimization without violating physics?

Goal-directed dynamics (GDD)

Force partitioning: $\tau = (u, z)$

Forward dynamics: $\dot{v} = f(q, v, \tau)$

Inverse dynamics: $\tau = g(q, v, \dot{v}) = (g_u, g_z)$

Forward definition: nested

$$\dot{v} = f\left(q, v, \arg\min_{u \in \mathcal{U}} \{\|(u, 0)\|_R + \ell(f(q, v, u))\}\right)$$

Inverse definition: not nested!

$$\dot{v} = \arg\min_{a \in \mathcal{A}(q,v)} \|g(q, v, a)\|_R + \ell(a)$$

Acceleration constraint: $\mathcal{A}(q, v) = \{a \in \mathcal{R}^n : g_u(q, v, a) \in \mathcal{U},\ g_z(q, v, a) = 0\}$

Regularization/control cost: $\|\tau\|_R \equiv \frac{1}{2}\tau^T R \tau$ where $R = M^{-1}$

GDD applied to MuJoCo physics

Optimization problem (from definition)

$$\dot{v} = \arg \min_{a \in \mathcal{A}(q,v)} \|g(q,v,a)\|_R + \ell(a)$$

$$\mathcal{A}(q,v) = \{a \in \mathcal{R}^n : g_u(q,v,a) \in \mathcal{U},\ g_z(q,v,a) = 0\}$$

Objective: $L(a) = \|g(a)\|_R + \ell(a)$ $\qquad g(a) = Ma + c + J^T \nabla s(Ja - r)$

Gradient: $\nabla L = PRg + \nabla \ell$ $\qquad P = \dfrac{\partial g}{\partial a} = M + J^T H[s] J$

Hessian: $H[L] = PRP + H[\ell]$

Non-convex non-smooth constrained optimization problem.

Primal-dual method with exact line search specific to MuJoCo.

Relation to forward and inverse dynamics

forward:	applied force $\rightarrow$ acceleration, constraint force
inverse:	acceleration $\rightarrow$ applied force, constraint force
GDD:	cost $\rightarrow$ acceleration, applied force, constraint force

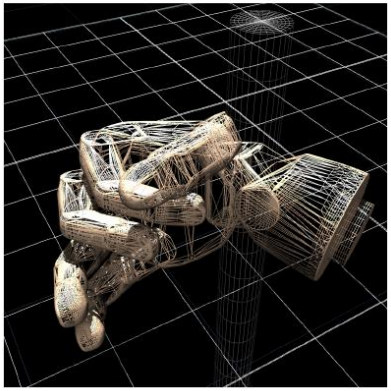
Contacts make the relation between force and acceleration piece-wise linear. With N active contacts we have 3^N pieces.

GDD optimizes over all pieces, instead of assuming a fixed piece.

Examples





GDD cost with three terms
(heuristic Q function):

- virtual damping on all joints

- virtual spring-damper between selected body and user-controlled spatial target

- optional desired pose for grasping

Numerical results

CPU time per step (μs)	**humanoid**	**hand**
forward	49	128
goal-directed	90	182
goal-directed + forward	99	194

satistic (50% (95%))	**humanoid**	**hand**
constraints	23 (38)	19 (30)
iterations	4 (9)	5 (9)
dual updates	1 (1)	1 (3)
physics violation	3e-11 (6e-7)	2e-8 (5e-7)
residual gradient	1e-7 (1e-6)	1e-9 (8e-9)

Acceleration-based Dynamic Programming

Discrete-time integration

$$q_{t+h} = q_t + hv_t$$
$$v_{t+h} = v_t + ha_t$$

Running cost

$$\|g(q, v, a)\|_R + p(q, v)$$

Bellman equation

$$V^*(q, v) = \quad p(q, v) + \min_{a \in \mathcal{A}(q,v)} \|g(q, v, a)\|_R + V^*(q + hv, v + ha)$$

The minimization step in the Bellman equation is equivalent to GDD with

$$\ell(a) = V^*(q + hv, v + ha)$$

Since $Q^* = p + ||g|| + V^*$, optimizing Q^* over actions is equivalent to GDD

Applications of GDD

Generalization of QP-based control and feature-based control

Trajectory optimization with acceleration-based formulation (AILQR)

Q-learning with optimization over actions built into MuJoCo

Training neural network controllers that output accelerations