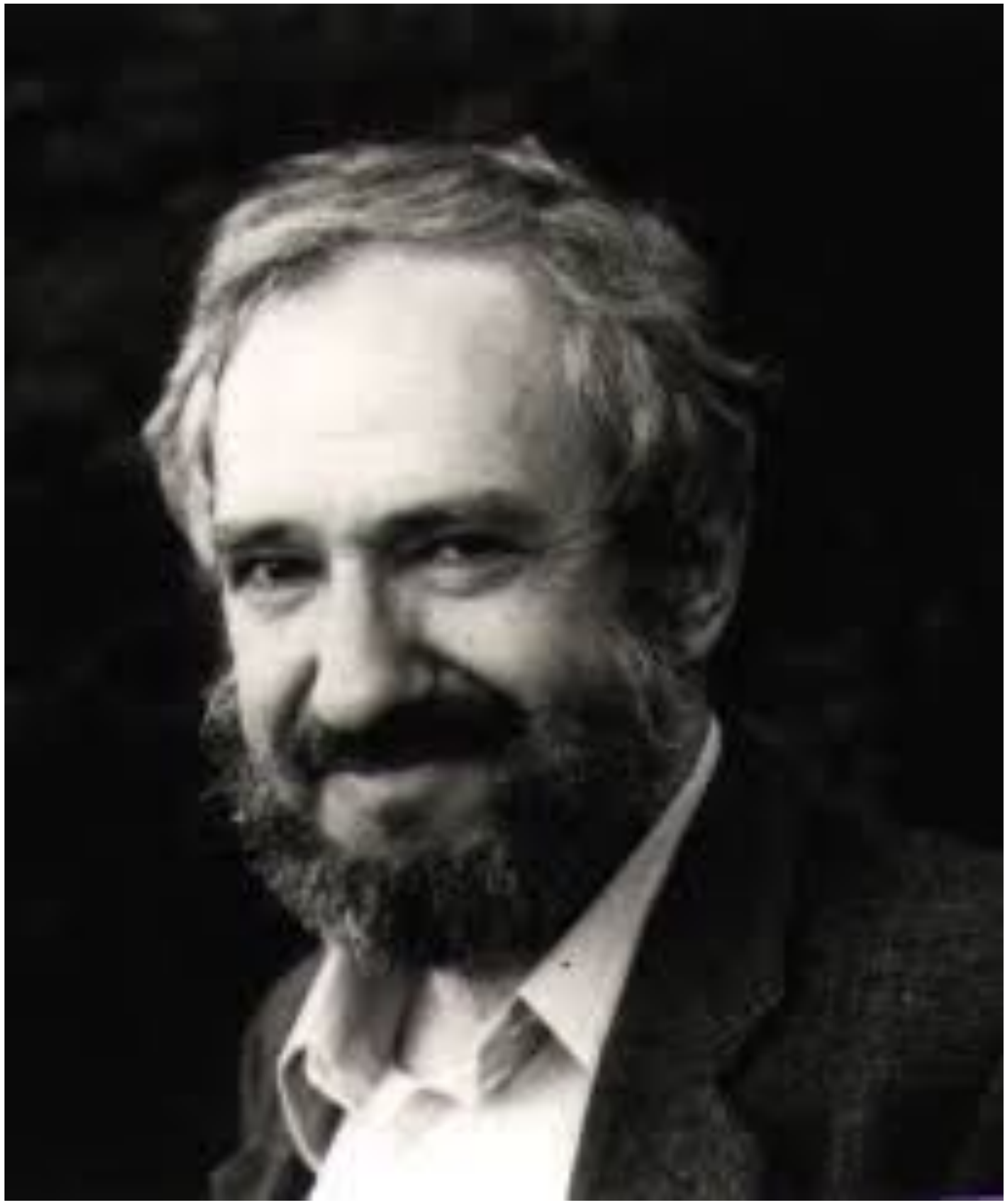
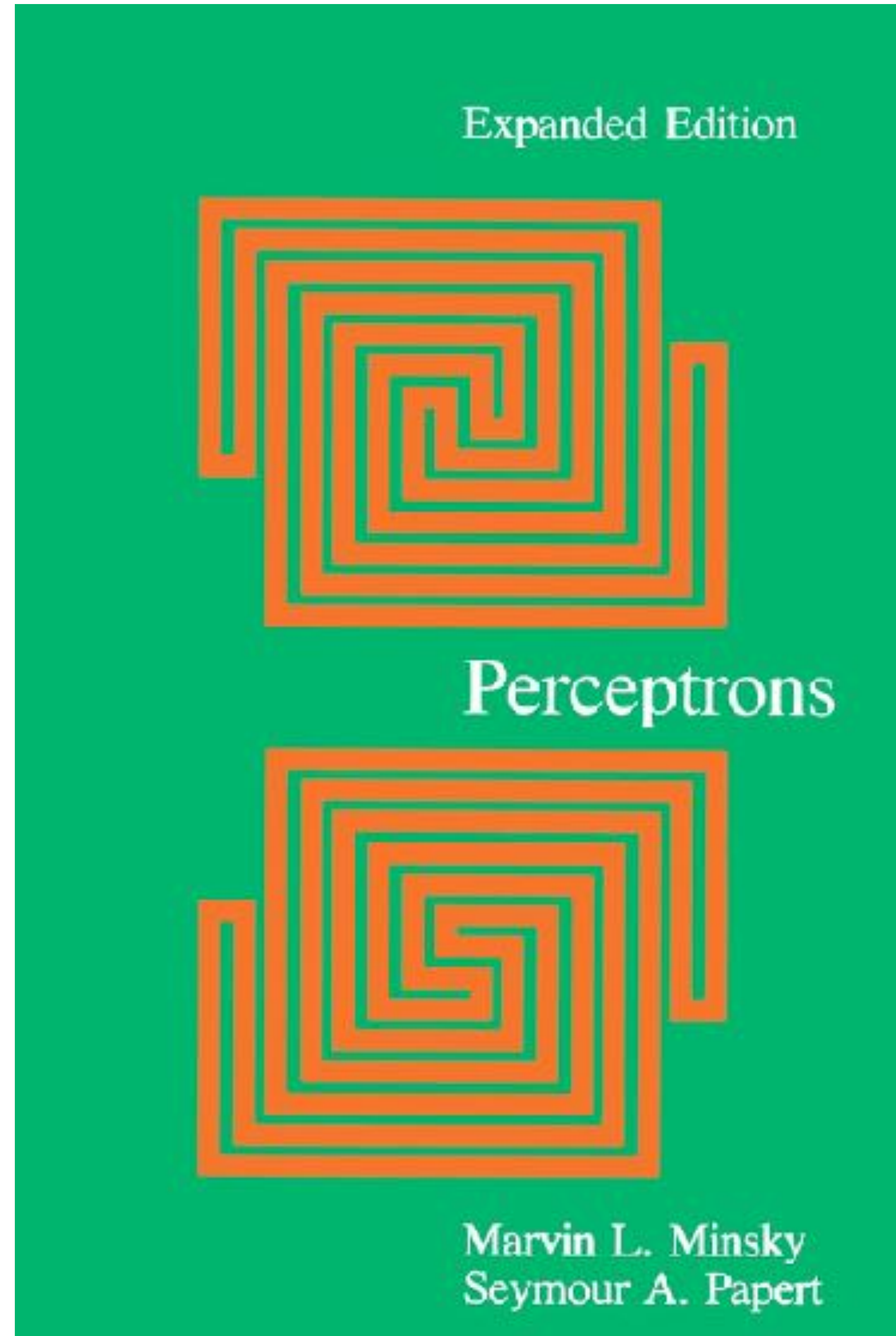
Advanced Reinforcement Learning

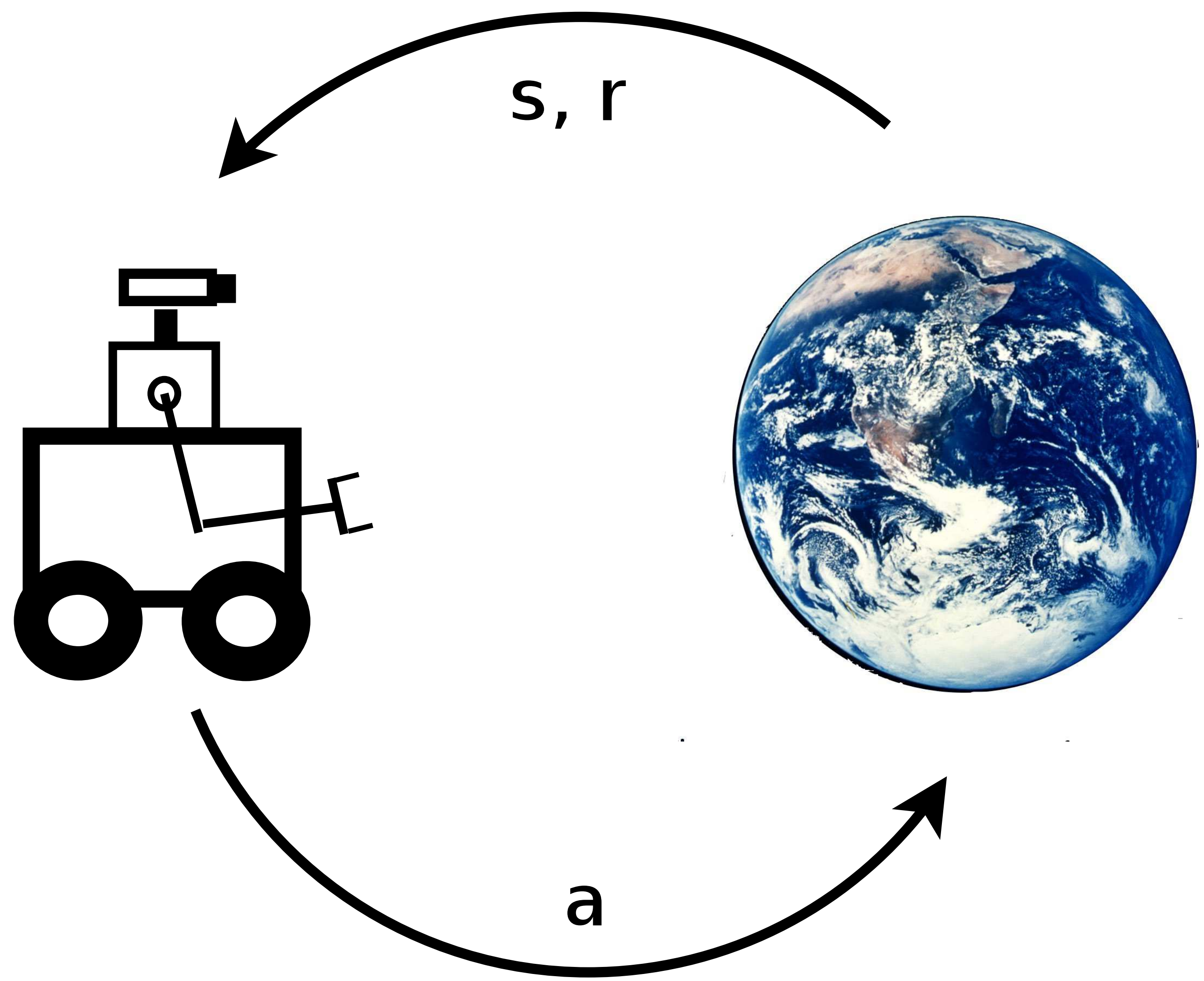
George Konidaris
gdk@cs.brown.edu

IN DEO SPERAMUS




Expanded Edition
Perceptrons
Marvin L. Minsky
Seymour A. Papert

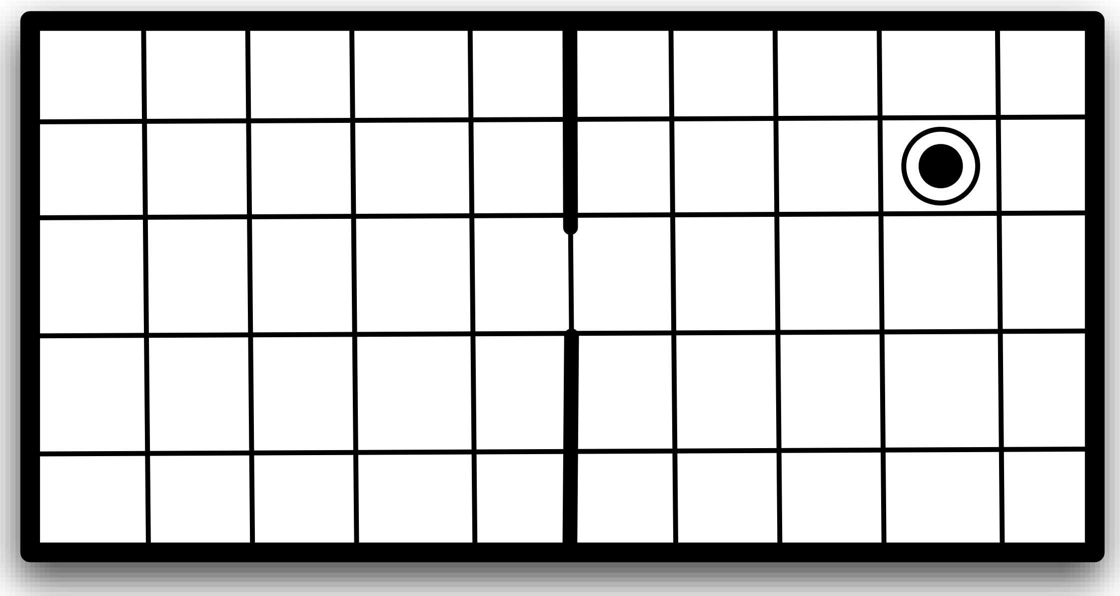
Reinforcement Learning

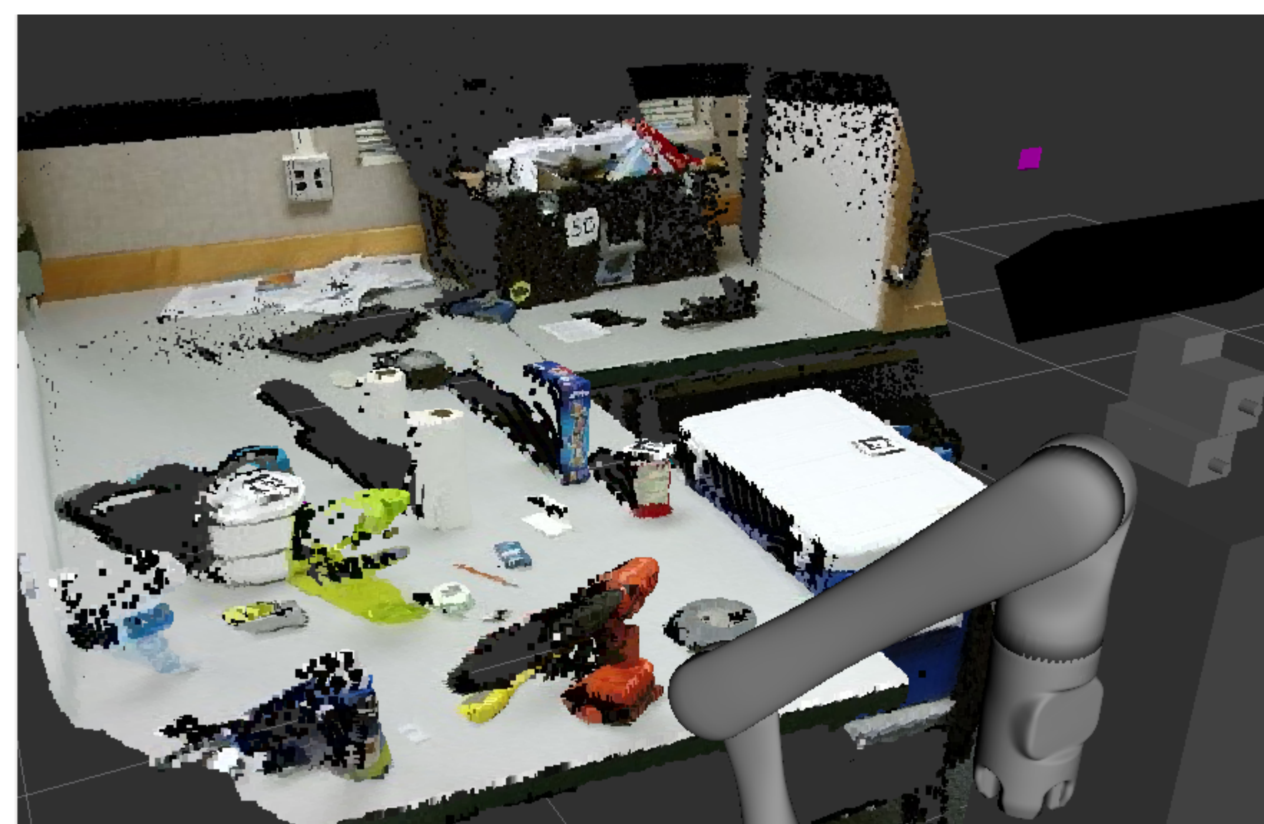


$$\pi : S \rightarrow A$$

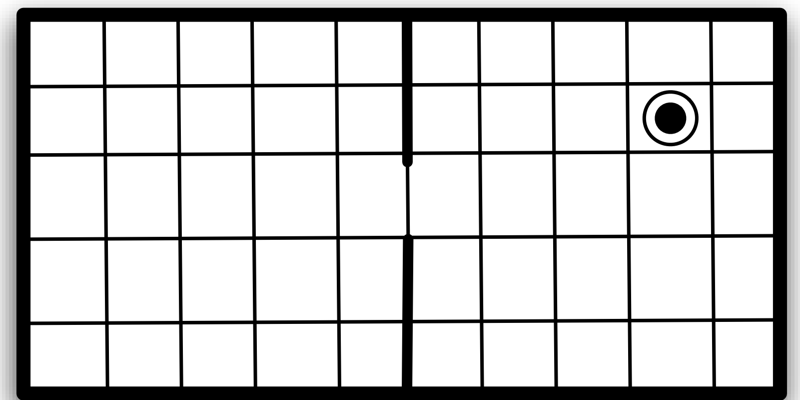
$$\max_{\pi} R = \sum_{t=0}^{\infty} \gamma^t r_t$$

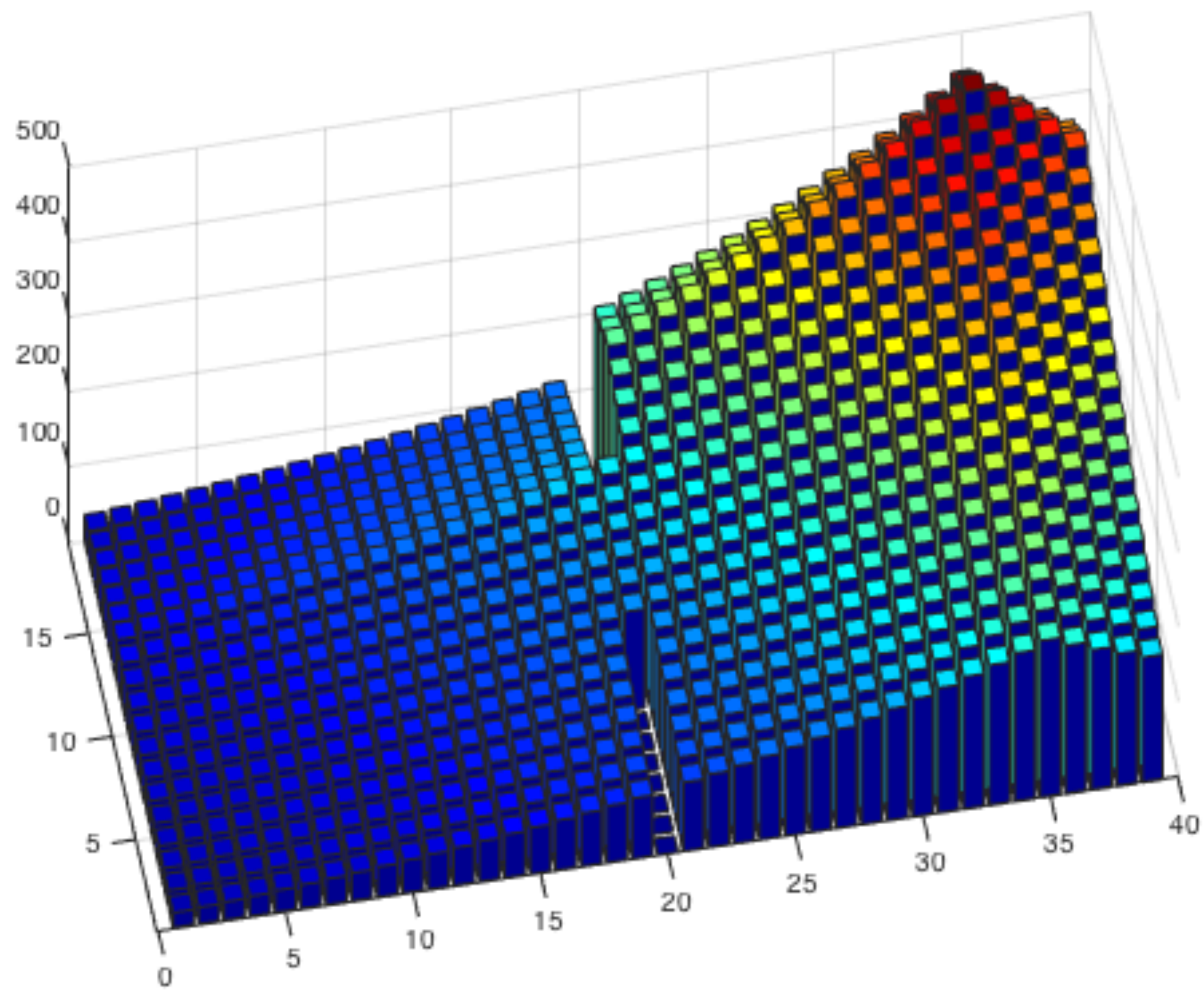
The World





Discrete RL





Real-Valued States

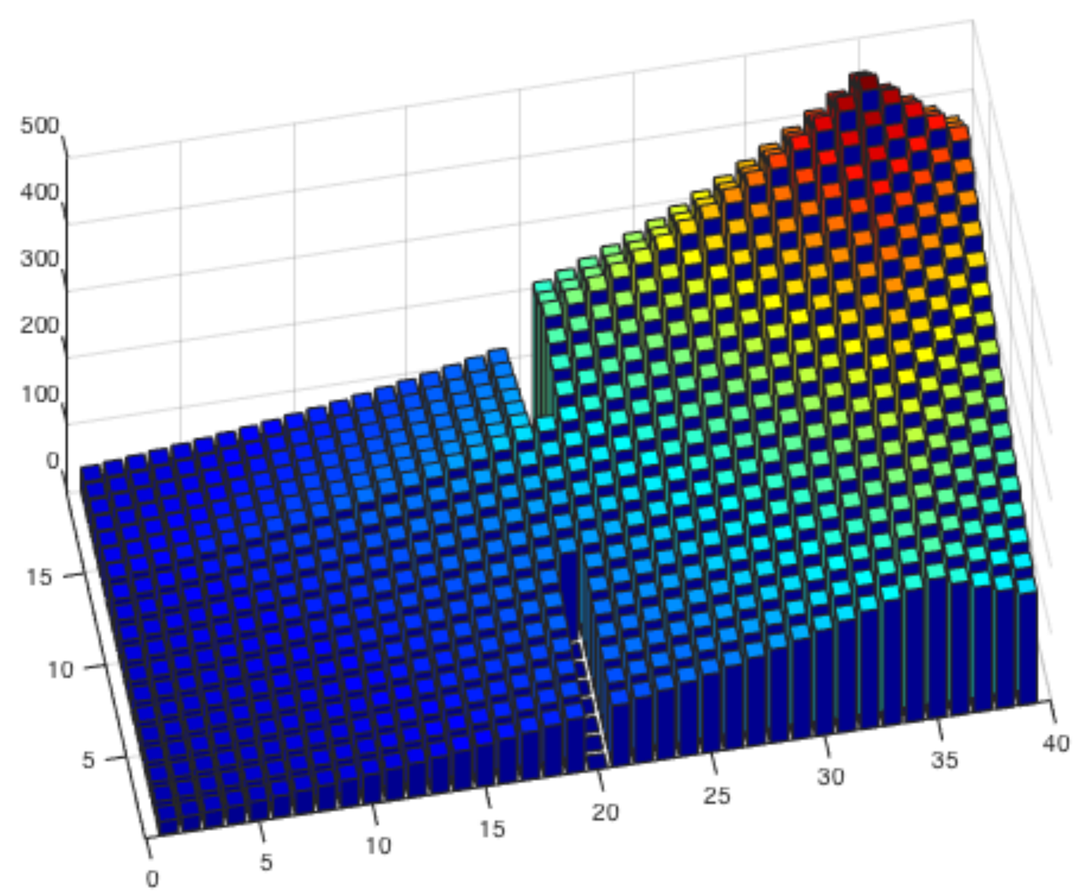
What if the states or actions are real-valued?

Need real-valued:

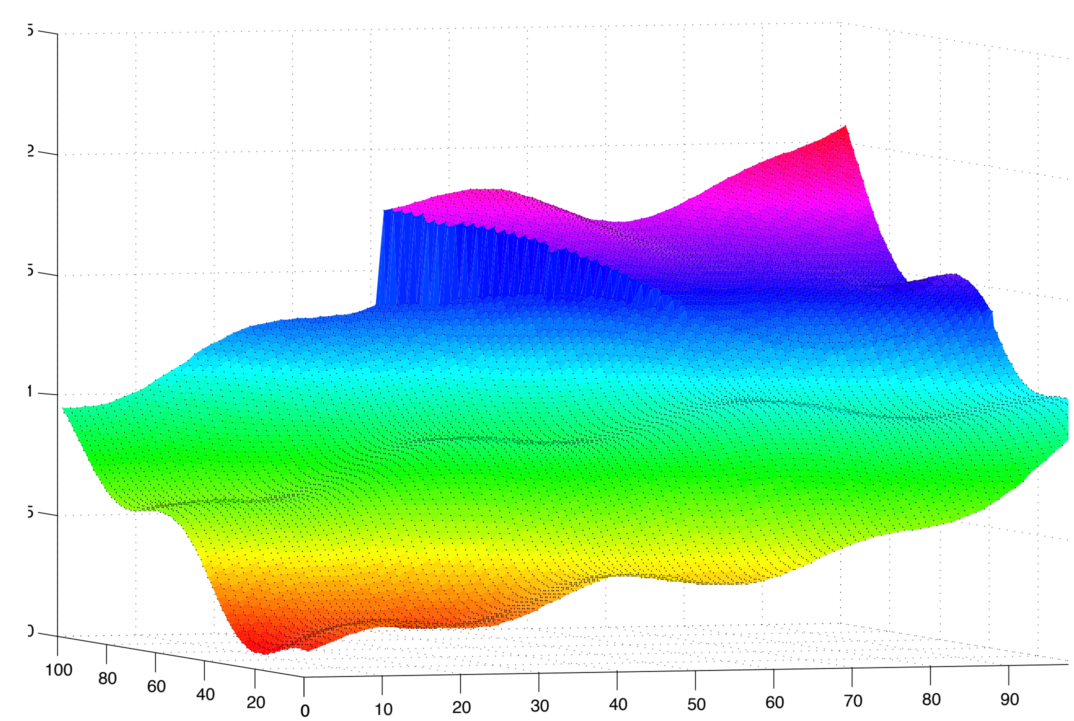
- Policies
- Value Functions
- Environmental Models

Key issues:

- Uncountable infinity
- May never revisit states
- Must generalize



vs



Function Approximation

Exactly as we have seen before.

- Represent function $f(x)$ in parametrized form:

$$f(x, w)$$

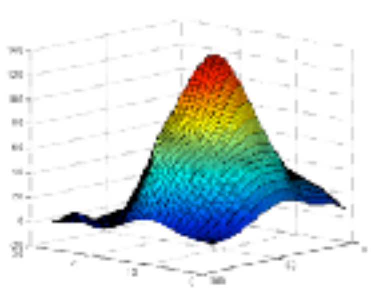
… for some parameter vector *w*.

- Write an objective function in terms of *w*.

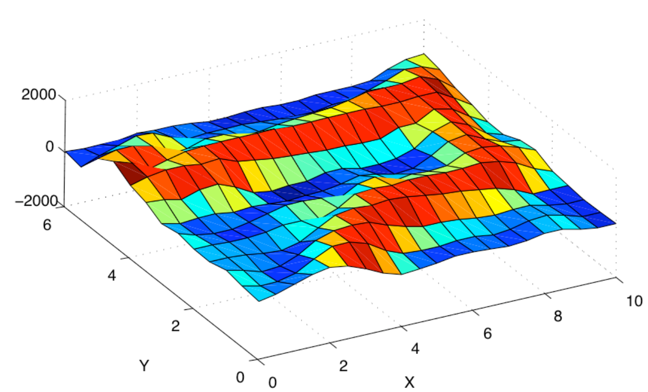
- Optimize (typically gradient descent).

Function Approximation

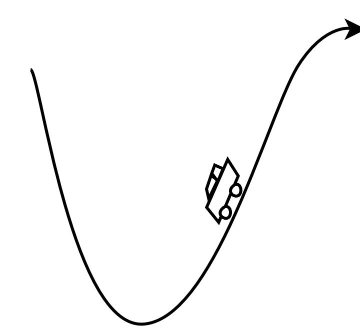
Value function



Policy

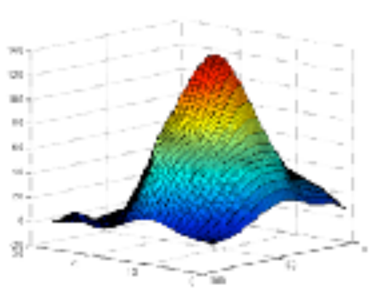


Model

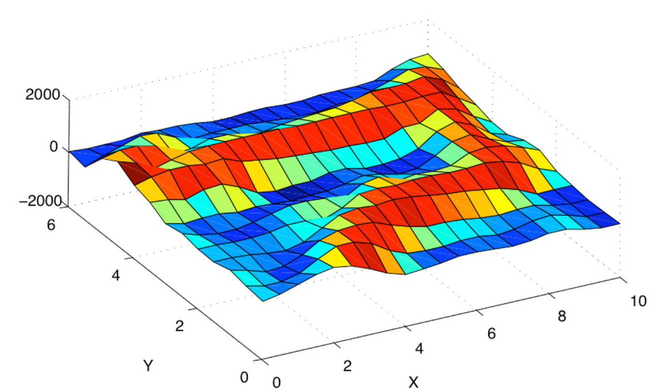


Function Approximation

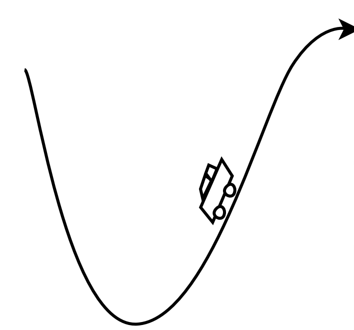
Value function



Policy



Model



Value Function Approximation

Represent Q function:

$$Q(s, a, w) : \mathbb{R}^n \rightarrow \mathbb{R}$$

Objective function?

Samples of form:

$$\left(s_i, a_i, r_i, s_{i+1}, a_{i+1}\right)$$

Minimize summed squared TD error:

$$\min_w \sum_{i=0}^{n} \left(Q(s_i, a_i, w) - r_i - \gamma Q(s_{i+1}, a_{i+1}, w)\right)^2$$

Value Function Approximation

Given a function approximator, compute the gradient and descend it.

Simplest thing you can do:

- *Linear value function approximation*.
- Use set of basis functions $\phi_1, ..., \phi_n$
- Q is a linear function of them:

$$\hat{Q}(s, a) = w \cdot \Phi(s, a) = \sum_{i=1}^{n} w_i \phi(s_i, a_i)$$

Function Approximation

One choice of basis functions:

- Just use state variables directly: $[1, x, y]$

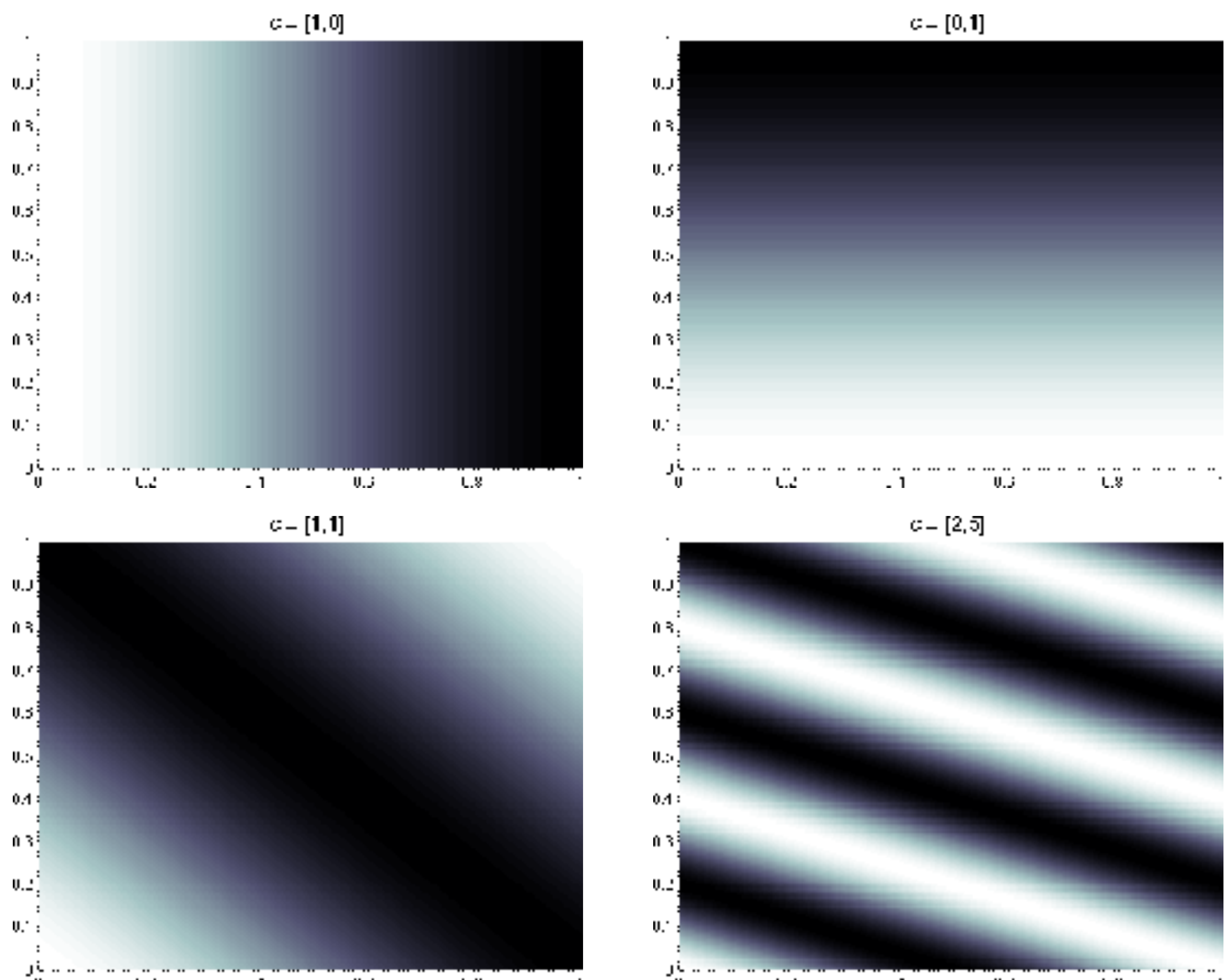
More powerful:

- Polynomials in state variables.
  - 1st order: $[1, x, y, xy]$
  - 2nd order: $[1, x, y, xy, x^2, y^2, x^2y, y^2x, x^2y^2]$
- This is like a Taylor expansion.

Function Approximation

Another:

- Fourier terms on state variables.
  - $[1, cos(\pi x), cos(\pi y), \cos(\pi[x+y])]$



Objective Function Minimization

First, let's do *stochastic gradient descent.*

As each data point (transition) comes in

- compute gradient of objective w.r.t. data point
- descend gradient a little bit

$$\hat{Q}(s,a) = w \cdot \Phi(s,a)$$

$$\min_w \sum_{i=0}^{n} \left(w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right)^2$$

Gradient

For each weight w_j:

$$\frac{\partial}{\partial w_j} \sum_{i=0}^{n} (w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1}))^2$$

$$= 2 \sum_{i=0}^{n} (w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1}))\, \phi_j(s_i, a_i)$$

TD error

so for each s_i the contribution is:

$$(w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1}))\, \phi_j(s_i, a_i)$$

make a step:

$$w_{j,i+1} = w_{j,i} + \alpha\, (w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1}))\, \phi_j(s_i, a_i)$$

$$w_{i+1} = w_i + \alpha \delta \phi(s_i, a_i)$$

λ-Gradient

The same logic applies when using eligibility traces.

$$w_{i+1} = w_i + \alpha\delta\phi(s_i, a_i)$$

becomes

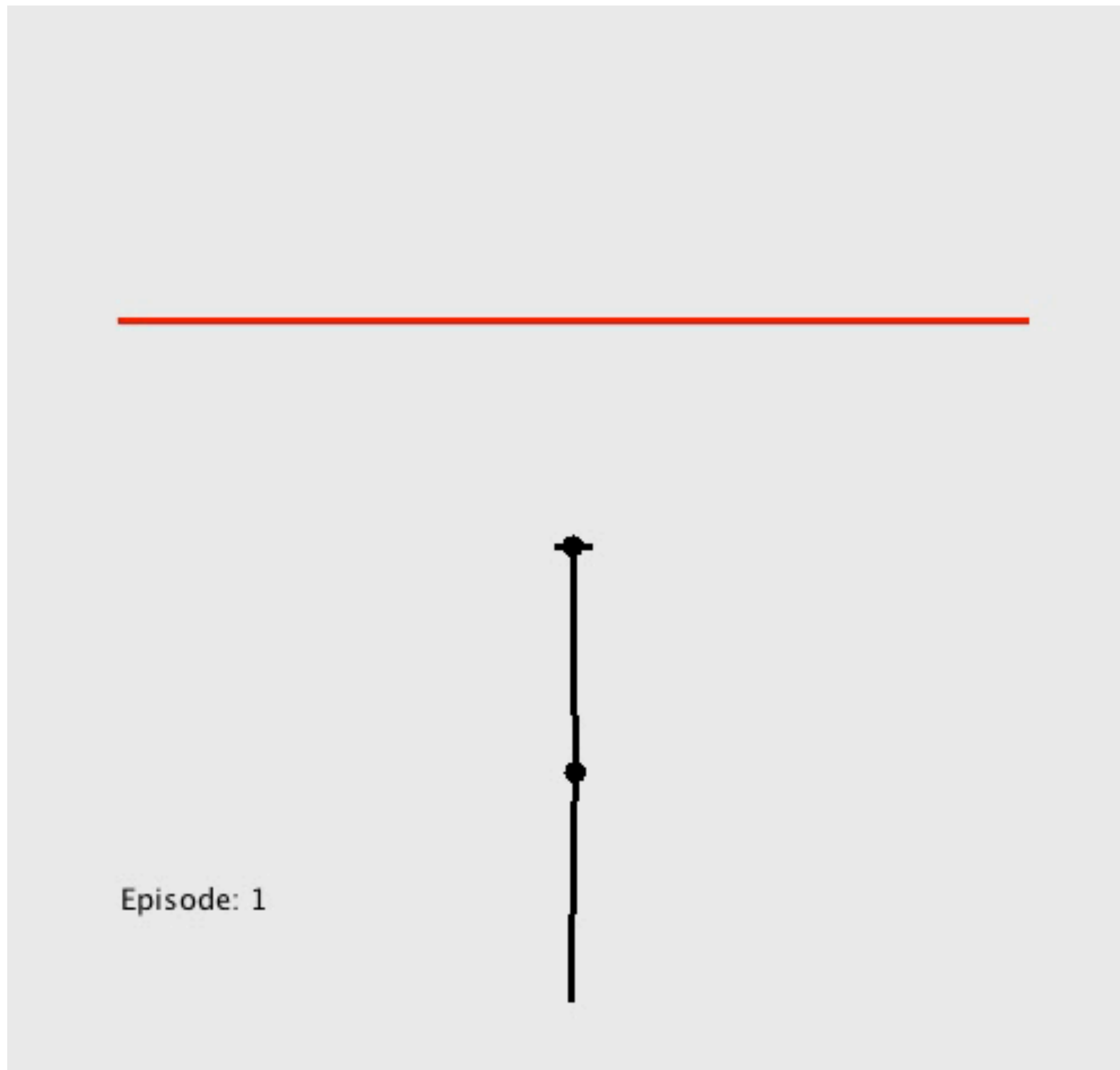
$$w_{i+1} = w_i + \alpha\delta e$$

where

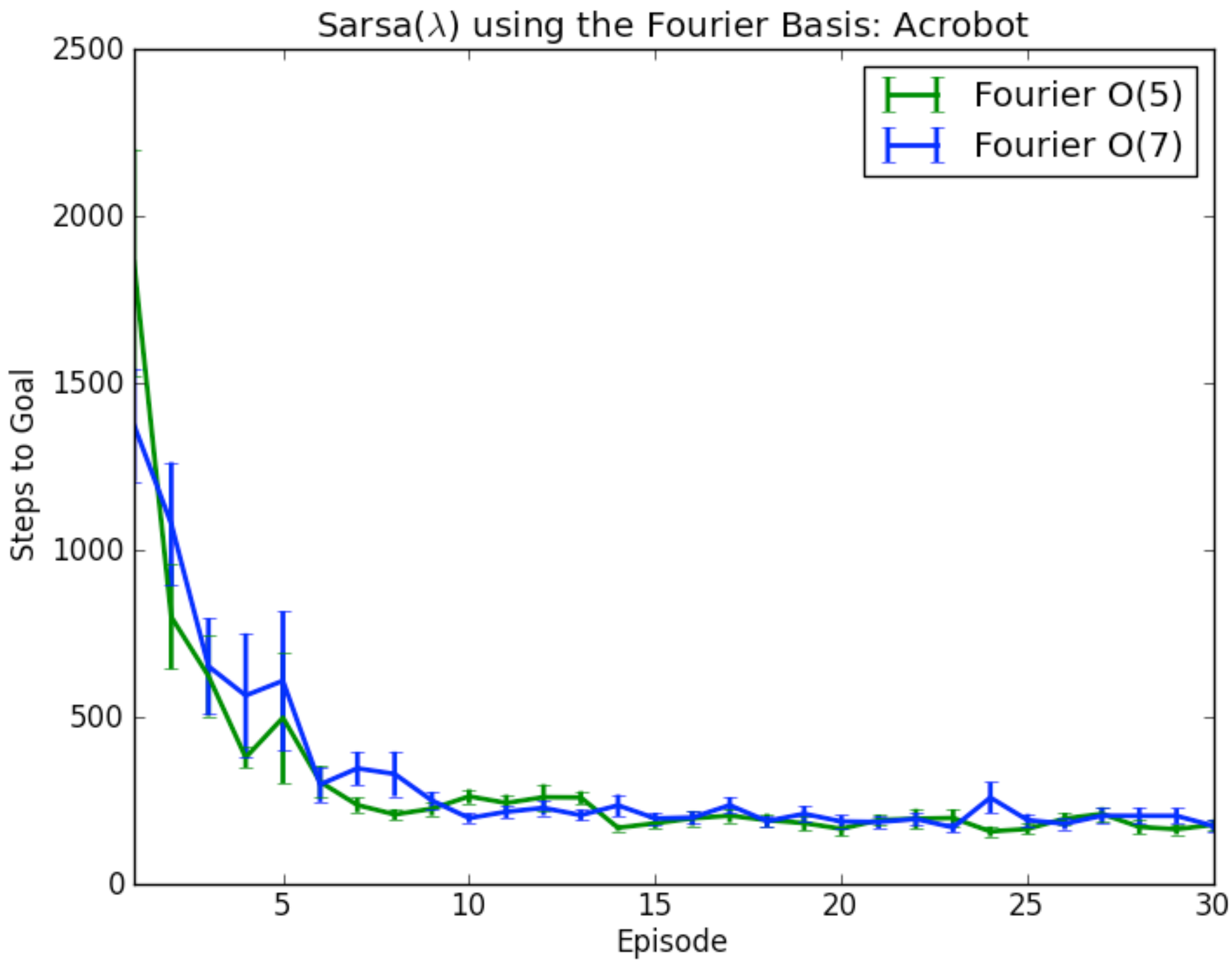
$$e_t = \gamma\lambda e_{t-1} + \phi(s_t, a_t)$$
$$e_0 = \bar{0}$$

[Sutton and Barto, 1998]

Acrobot



Acrobot



Sarsa(λ) using the Fourier Basis: Acrobot
Fourier O(5)
Fourier O(7)
Steps to Goal
Episode
2500
2000
1500
1000
500
0
5
10
15
20
25
30

Least-Squares TD

Minimize:

$$\min_w \sum_{i=0}^{n} \left(w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right)^2$$

Error function has a bowl shape, so *unique minimum*. Just go right there!

Least-Squares TD

Derivative set to zero:

$$\sum_{i=1}^{n} \left(w \cdot \phi(s_i, a_i) - r_i - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right) \phi(s_i, a_i)^T = 0$$

$$w^T \sum_{i=1}^{n} \left(w \cdot \phi(s_i, a_i) - \gamma w \cdot \phi(s_{i+1}, a_{i+1})\right) \phi^T(s_i, a_i) = \sum_{i=1}^{n} r_i \phi^T(s_i, a_i)$$

$$w = A^{-1} b$$

$$A = \sum_{i=1}^{n} \left(\phi(s_i, a_i) - \gamma \phi(s_{i+1}, a_{i+1})\right) \phi^T(s_i, a_i)$$

$$b = \sum_{i=1}^{n} r_i \phi^T(s_i, a_i)$$

[Bradtke and Barto, 1996]

LSTD(λ)

Can derive the least-squares version of LSTD(λ) in this way.
Try it at home!

- Write down the objective function …
- Sample r_i replaced by complex reward estimate.

- You will get a trace vector if you do some clever algebra.
- Trace vector is the same size as w.

[Boyan, 1999]

LSTD(λ)

One inversion solves for *w*!

But:

- Computationally expensive.
- *A* may not be invert-able.
- Least-squares behavior sometimes unstable outside of data.

- LSPI: Least Squares Policy Iteration
- Requires recomputing *A* over historical data.
  - a_{i+1} changes with the policy

[Lagoudakis and Parr, 2003]

Linear Methods Don't Scale

Why not?

- They're complete.
- They have nice properties (bowl-shaped error).
- They are easy to use!

How many basis functions in a complete *n*th order Taylor series of *d* variables?

$$(n+1)^d$$

Function Approximation

TD-Gammon: Tesauro (circa 1992-1995)

- At or near best human level
- Learn to play Backgammon through self-play
- 1.5 million games
- Neural network function approximator
- TD(λ)

Changed the way the best human players played.

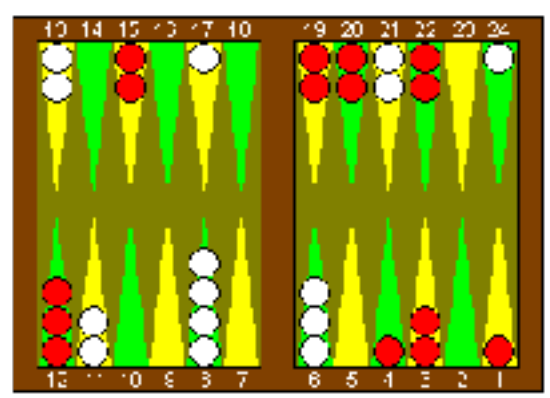
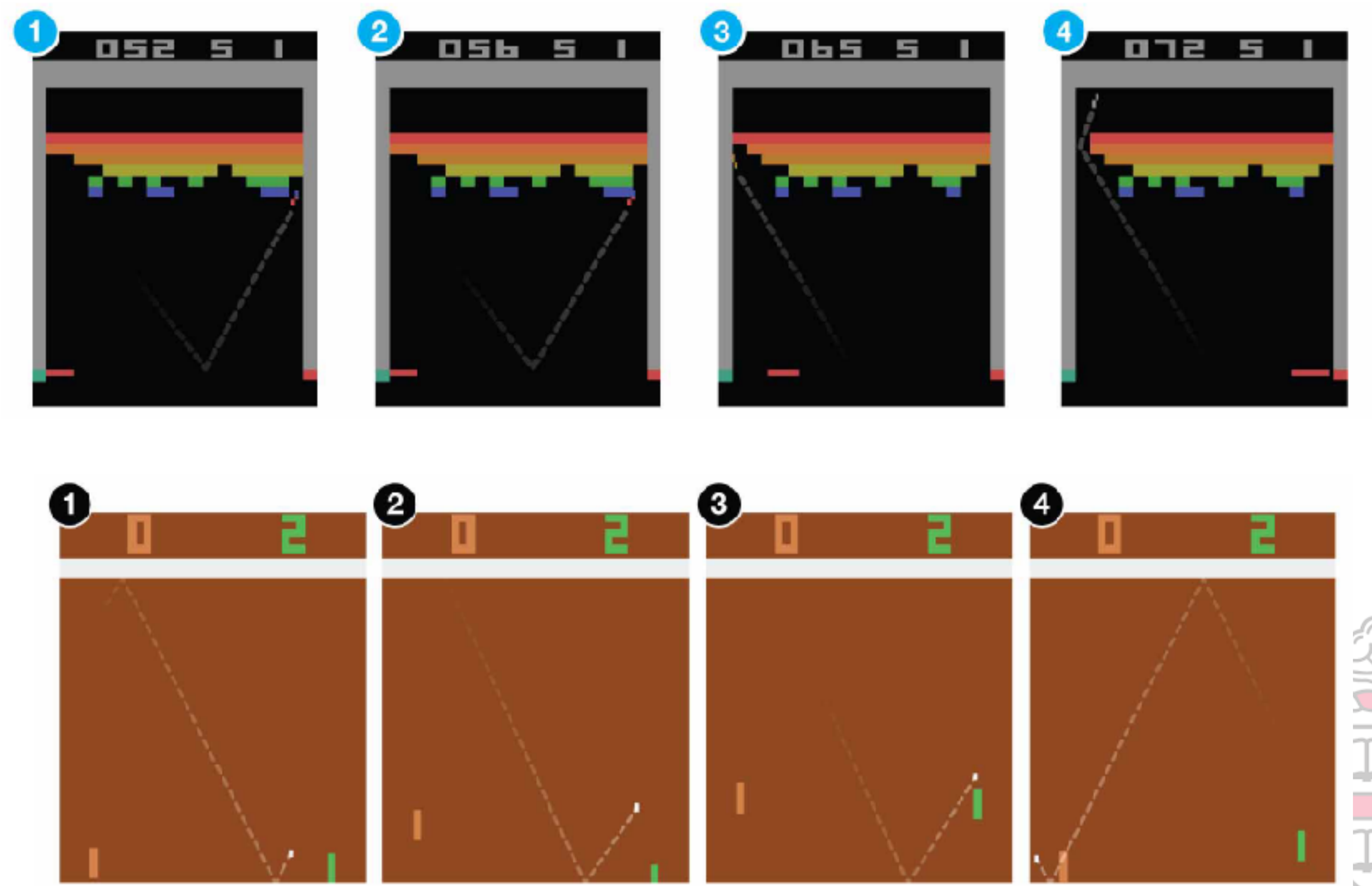


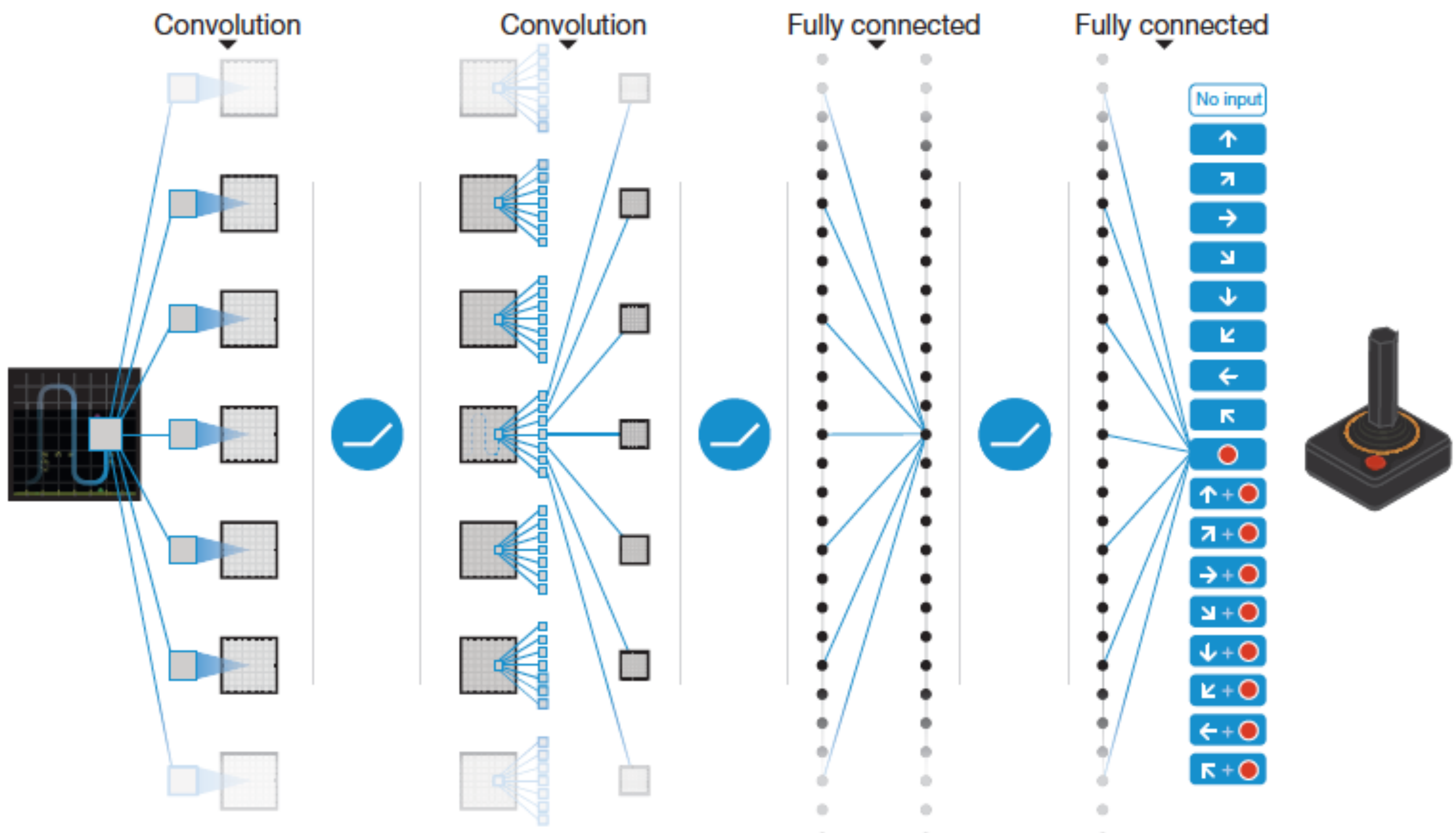
Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 8-4*, 8-4, 21-17, 21-17! TD-Gammon's analysis of the two plays is given in Table 3.

Arcade Learning Environment



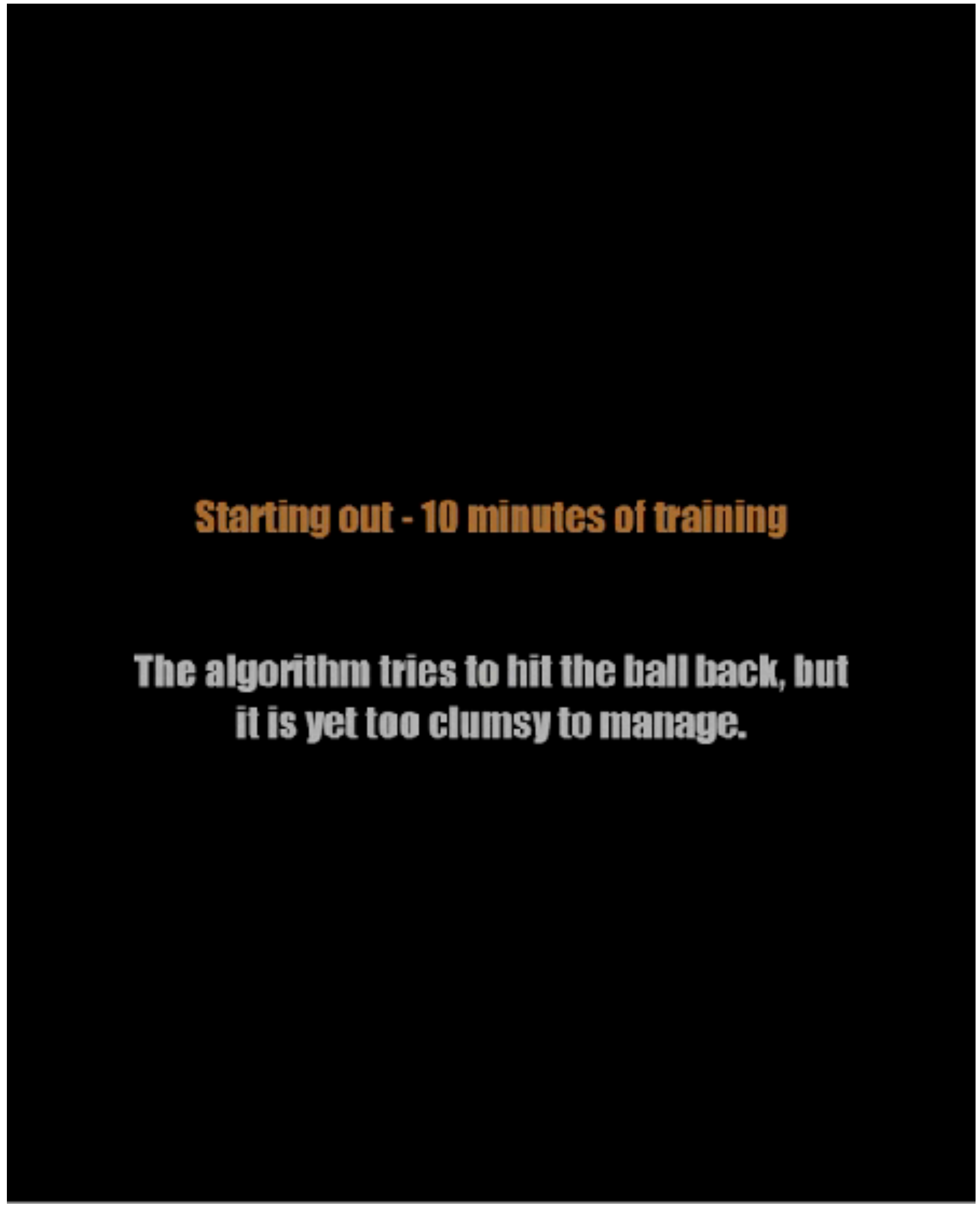
[Bellemare 2013]

Deep Q-Networks



[Mnih et al., 2015]

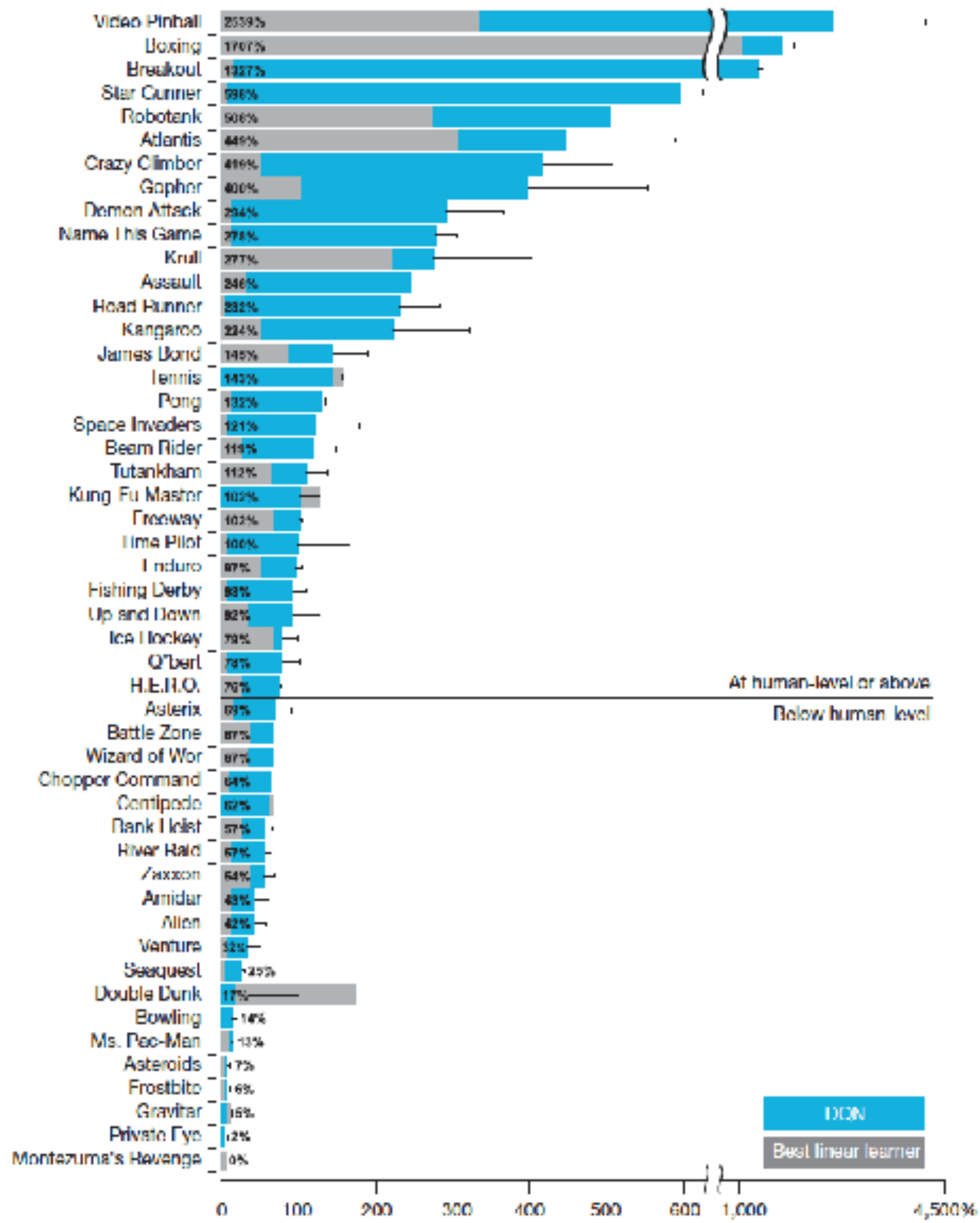
Atari



[Mnih et al., 2015]

video: Two Minute Papers

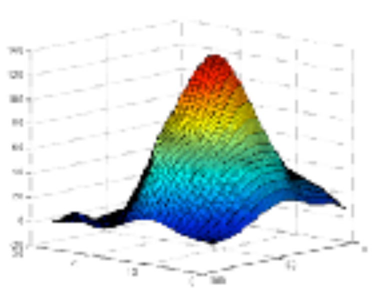
Atari



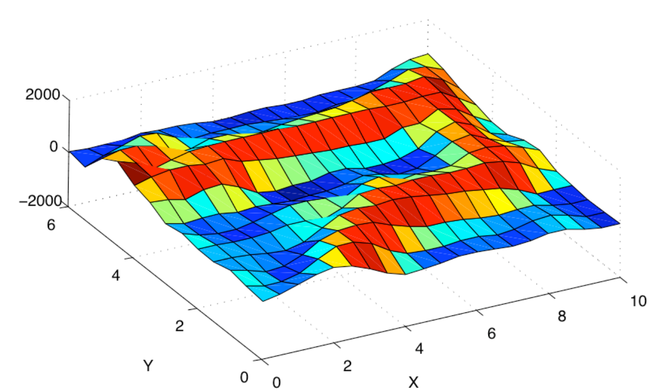
[Mnih et al., 2015]

Function Approximation

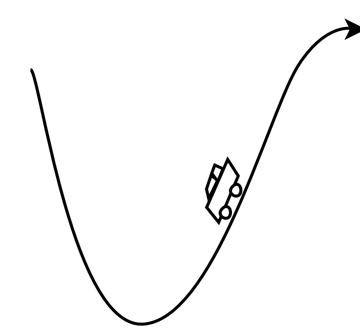
Value function



Policy



Model



Policy Search

Represent policy directly:

$$\pi(s, a, \theta) : \mathbb{R}^n, \mathbb{R}^m \rightarrow [0, 1]$$

Why?

Objective function?

Hill Climbing

What if you can't differentiate π?

Sample-based optimization:

- Sample some θ values near your current best θ.
- Adjust your current best to the highest value θ.

Aibo Gait Optimization

from Kohl and Stone, ICRA 2004.

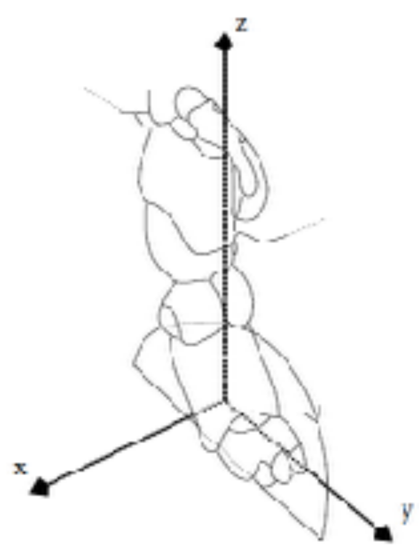


Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.

All told, the following set of 12 parameters define the Aibo's gait [10]:

- The front locus (3 parameters: height, x-pos., y-pos.)
- The rear locus (3 parameters)
- Locus length
- Locus skew multiplier in the x-y plane (for turning)
- The height of the front of the body
- The height of the rear of the body
- The time each foot takes to move through its locus
- The fraction of time each foot spends on the ground

PoWER and PI2

More recently, two closely related algorithms:

- Generate some sample θ values.
- Next θ is sum of prior samples weighted by reward.

(Theodorou and Schaal 2010, Kober and Peters 2011)

REINFORCE

If we can differentiate π …

- Compute and ascend $\partial R / \partial \theta$
- This is the gradient of return w.r.t policy parameters

REINFORCE: one particularly popular sample-based estimate of the gradient.

$$\Delta \theta_t = \alpha r_t \frac{\nabla \pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta)}$$

Policy Search

Slightly more general theorem - policy gradient theorem.

$$\frac{\partial R}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} (Q^\pi(s,a) - b(s))$$

Therefore, one way is to learn Q and then ascend gradient.
Q need only be defined using basis functions computed from θ.

[Sutton et al. 1999]

Deep Policy Search

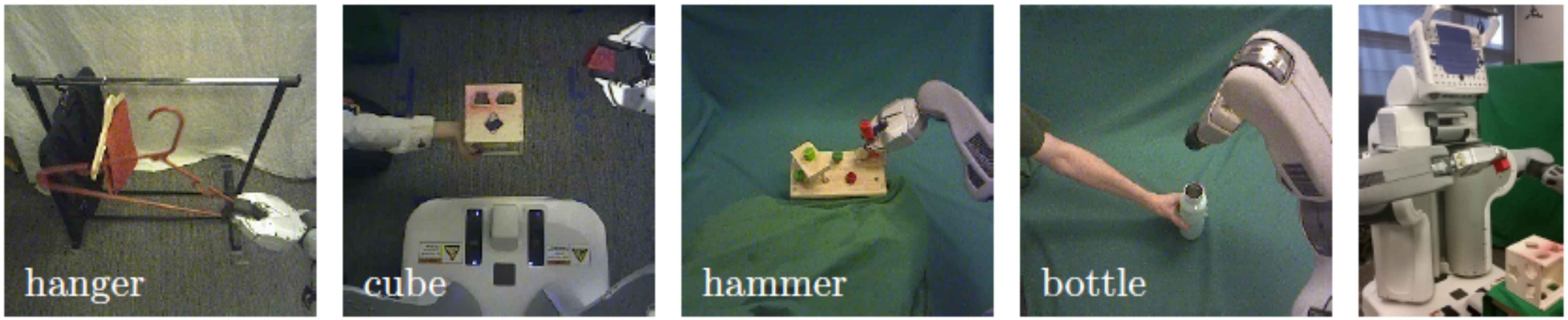
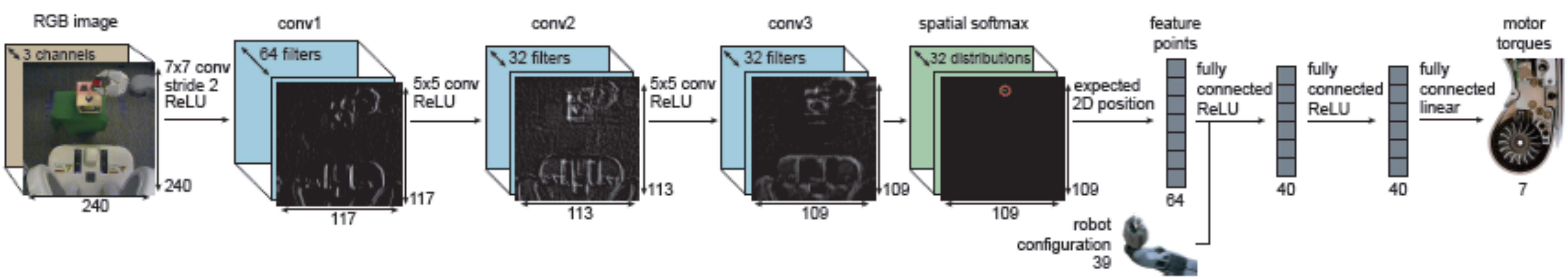
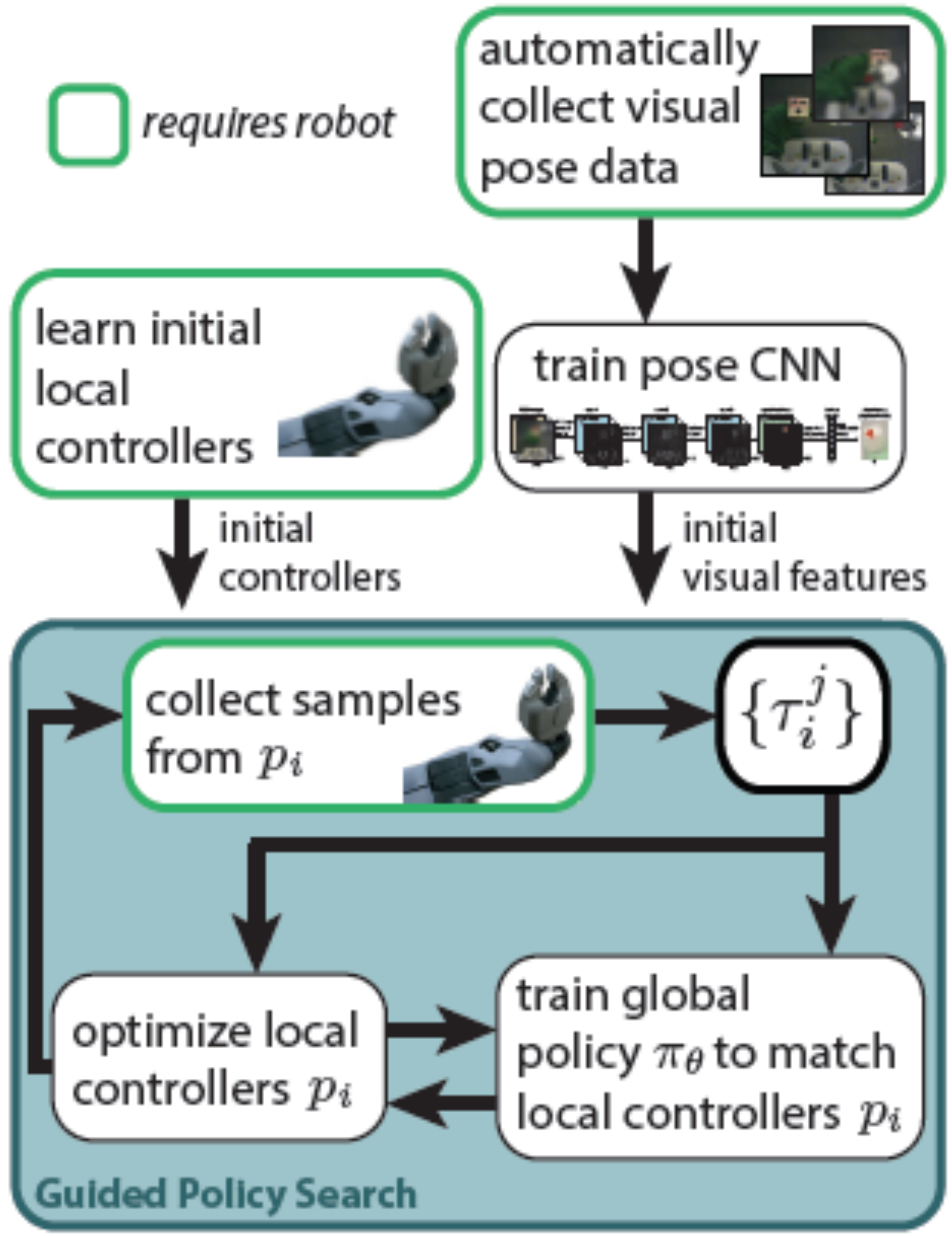


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).



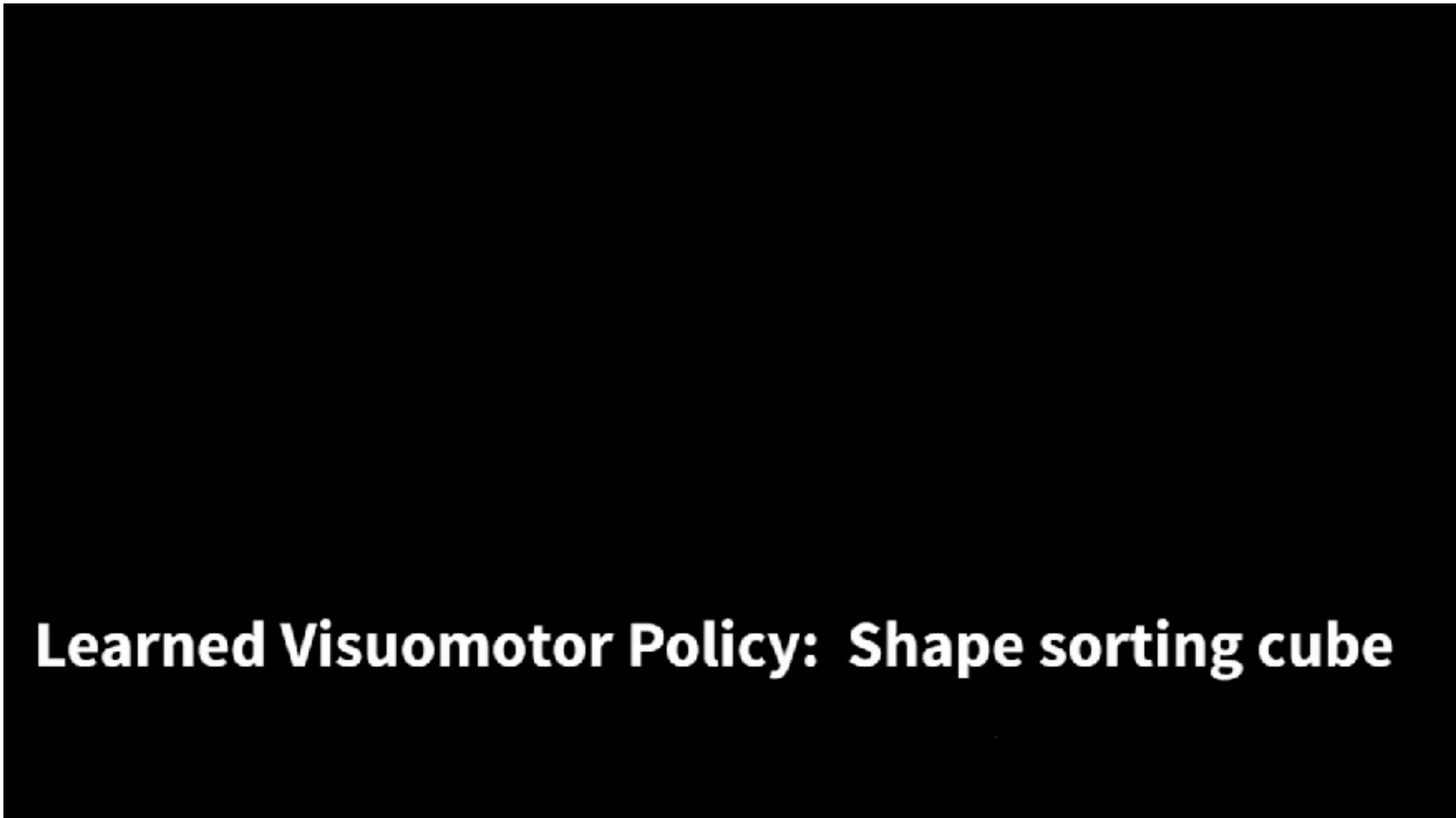
[Levine et al., 2016]

Deep Policy Search



[Levine et al., 2016]

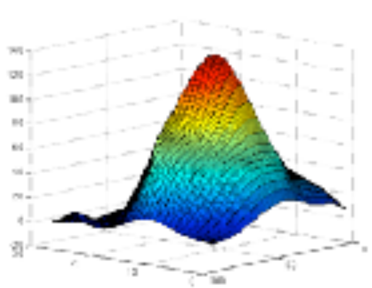
Robotics



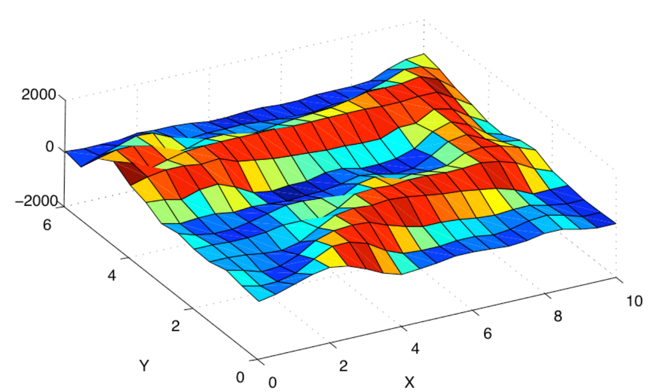
[Levine et al., 2016]

Function Approximation

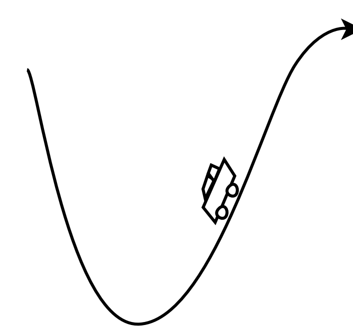
Value function



Policy



Model



Learning a Model

Learn a model:

$$T(s_{i+1}|s_i, a_i, w)$$

Why?

Objective function?

Samples of form:

$$(s_i, a_i, r_i, s_{i+1}, a_{i+1})$$

Maximize likelihood of observed transitions:

$$\max_w \Pi_{i=1}^n T(s_{i+1}|s_i, a_i, w)$$

Procedure

Model-based RL algorithms roughly look like:

- Get some transition data
- Learn a model
- Run RL on samples from that model to convergence
- Repeat

Advantages?

This never works. Why?

PILCO

The main issue is that *your model is never exactly right.*

- Policy specialized to model.
- Typically assume predictions are "correct".
- But the model is **uncertain**!

Recent breakthrough: *Bayesian policy search:*

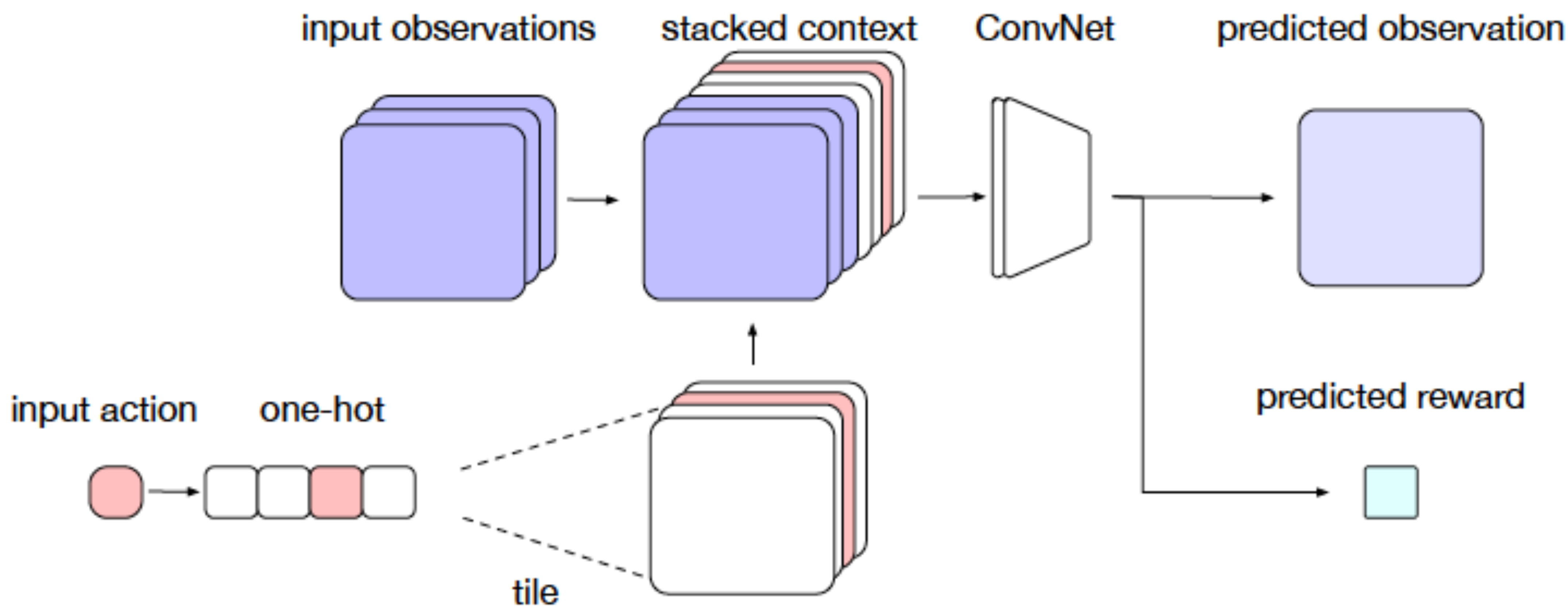
$$\int_M \mathbb{E}\left[\sum_t R(s_t)\right]$$

[Deisenroth et al, 2011]

PILCO

Combine Gaussian process dynamics learning with analytic policy gradient methods.

Deep Models



[Weber et al., 2017]

Deep Models

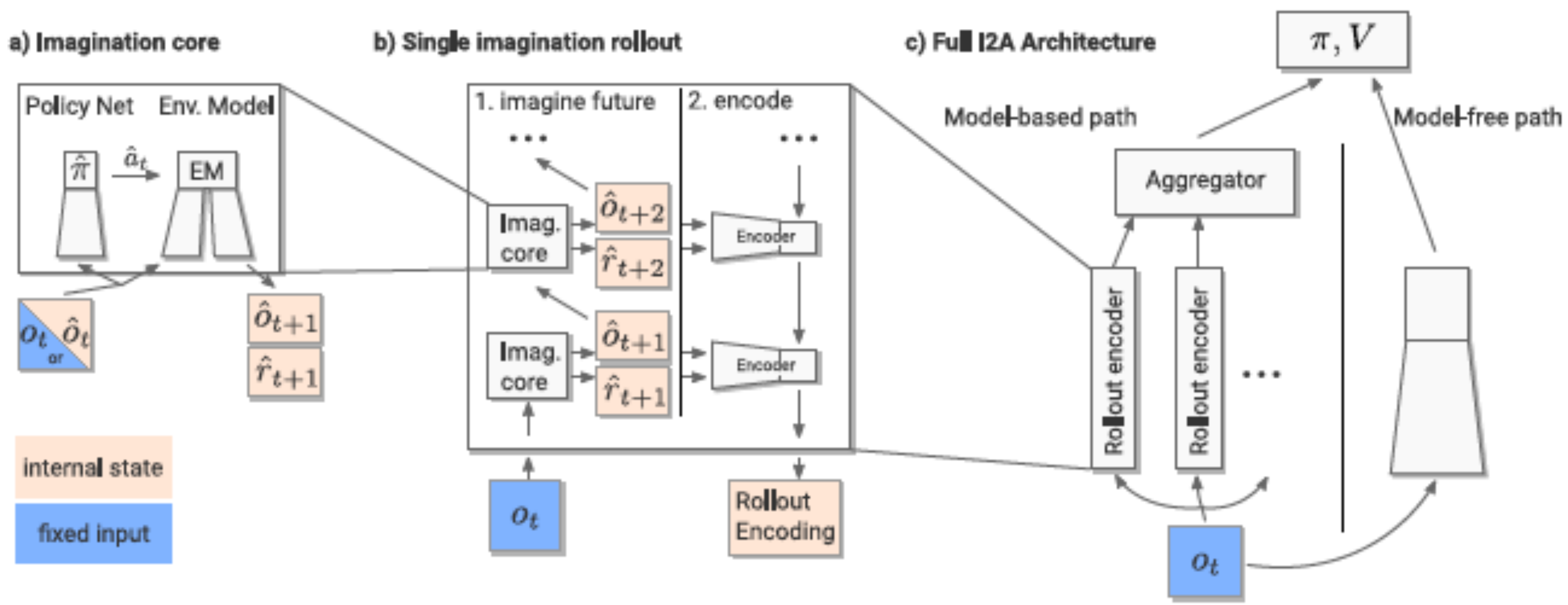
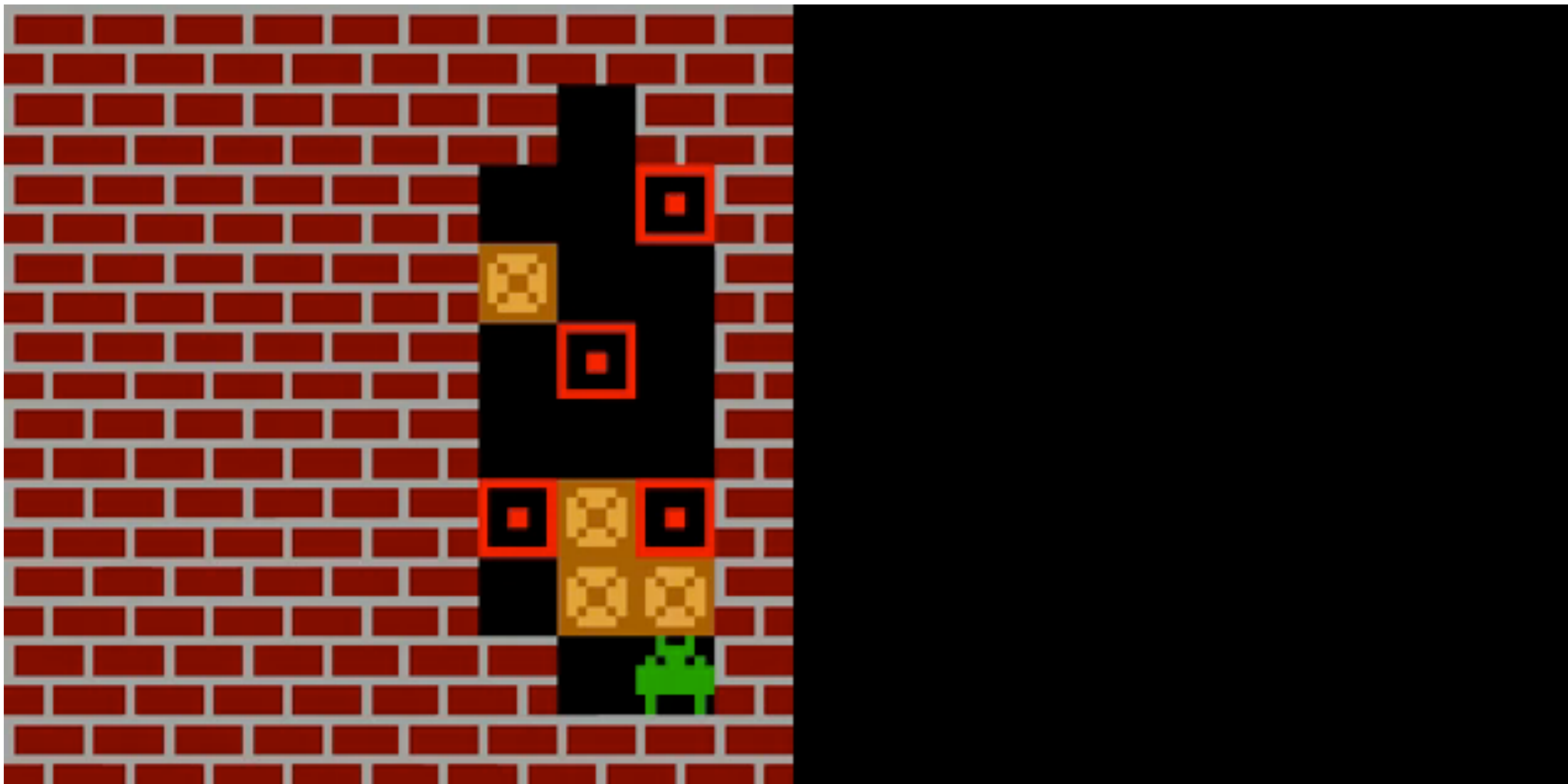


Figure 1: *I2A architecture.* $\hat{\cdot}$ notation indicates imagined quantities. *a)*: the imagination core (IC) predicts the next time step conditioned on an action sampled from the rollout policy $\hat{\pi}$. *b)*: the IC imagines trajectories of features $\hat{f} = (\hat{o}, \hat{r})$, encoded by the rollout encoder. *c)*: in the full I2A, aggregated rollout encodings and input from a model-free path determine the output policy π.

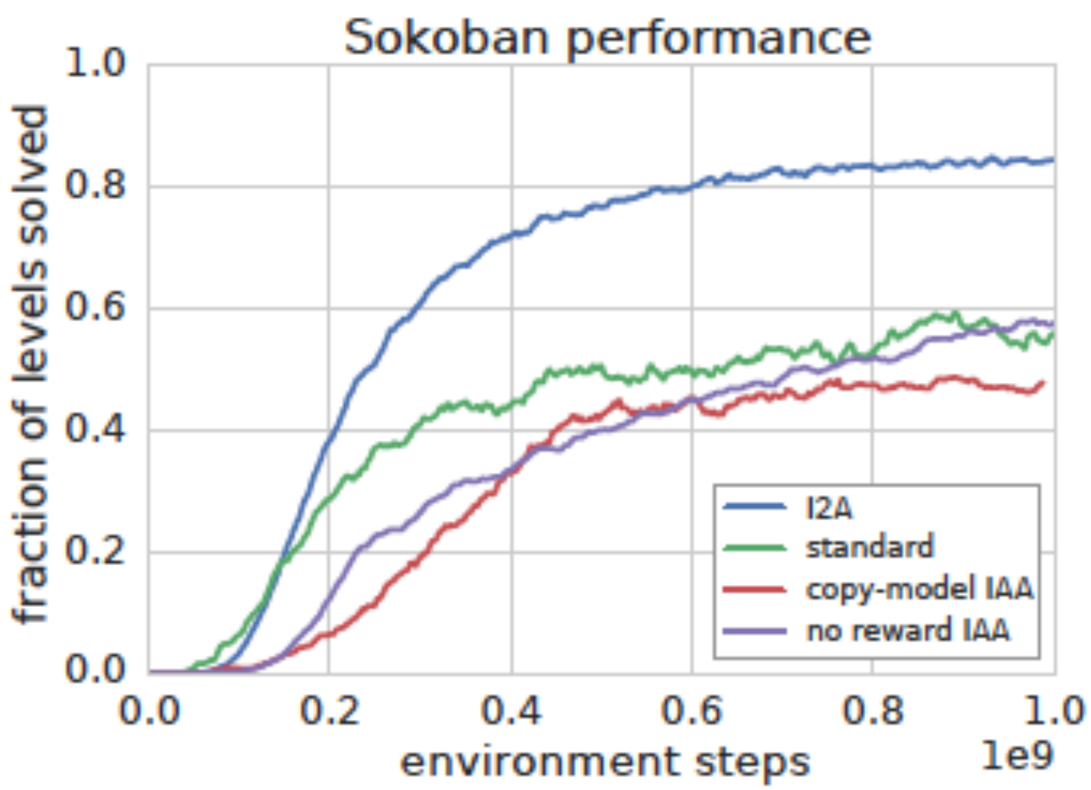
[Weber et al., 2017]

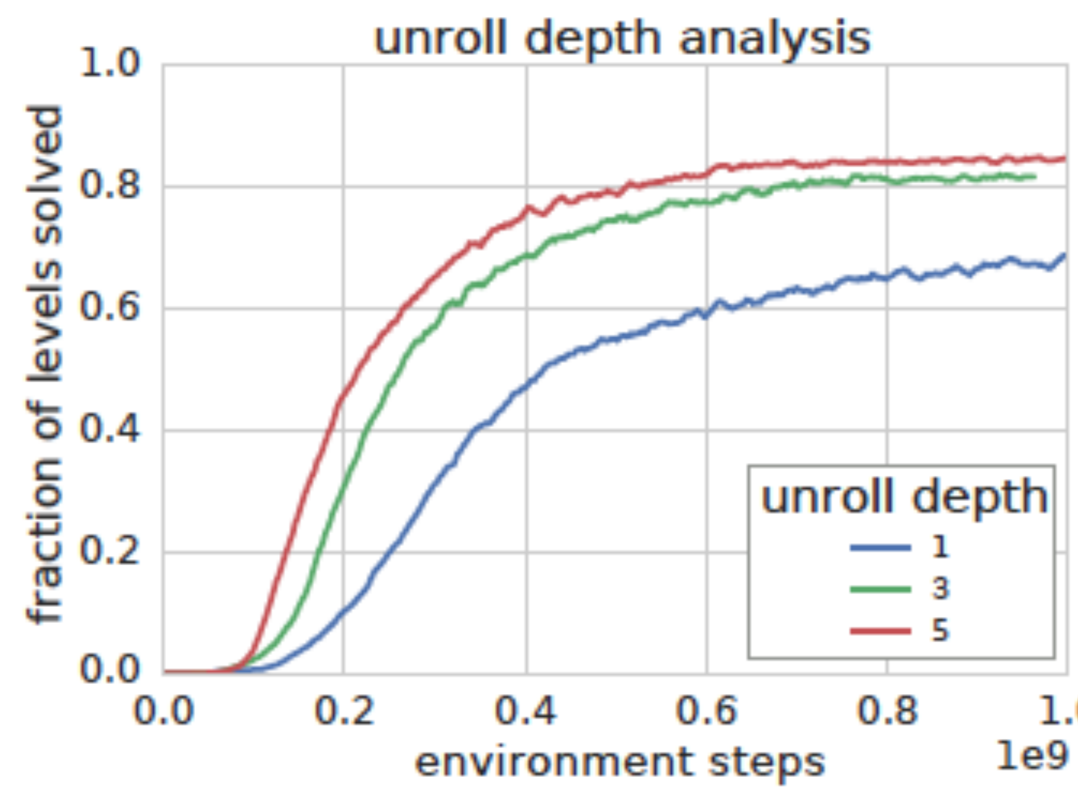
Deep Models



[Weber et al., 2017]

Deep Models

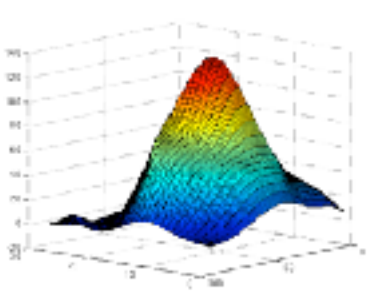




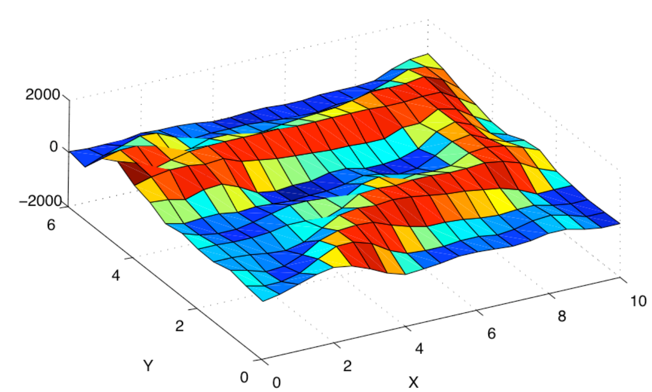
[Weber et al., 2017]

Function Approximation

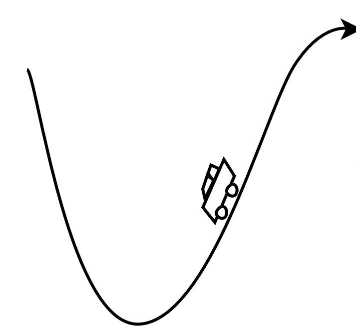
Value function



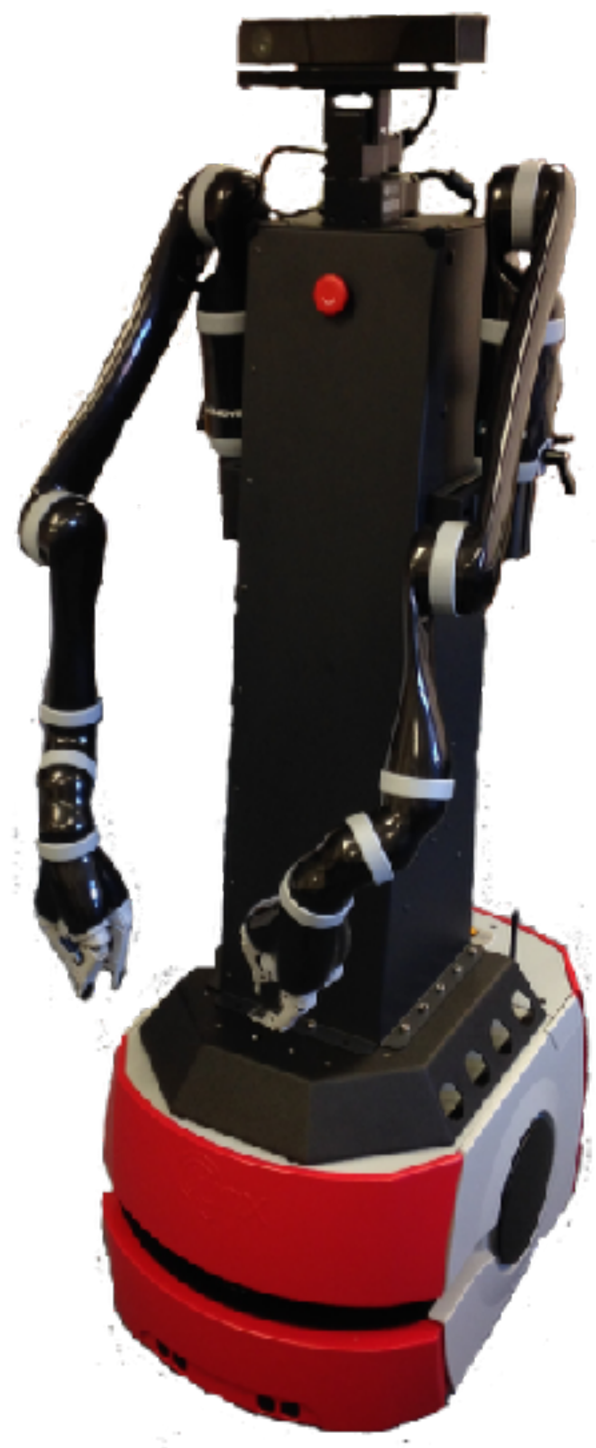
Policy



Model



Hierarchical RL



Skill Hierarchies

Hierarchical RL: base hierarchical control on *skills*.

- Component of behavior.
- Performs continuous, low-level control.
- Can treat as discrete action.

Behavior is modular and compositional.

Skills are like *subroutines*.

```
def abs(x):
    if(x > 0):
        return x
    else:
        return -x
```

[Wilkes, Wheeler and Gill, 1951]

Hierarchical RL

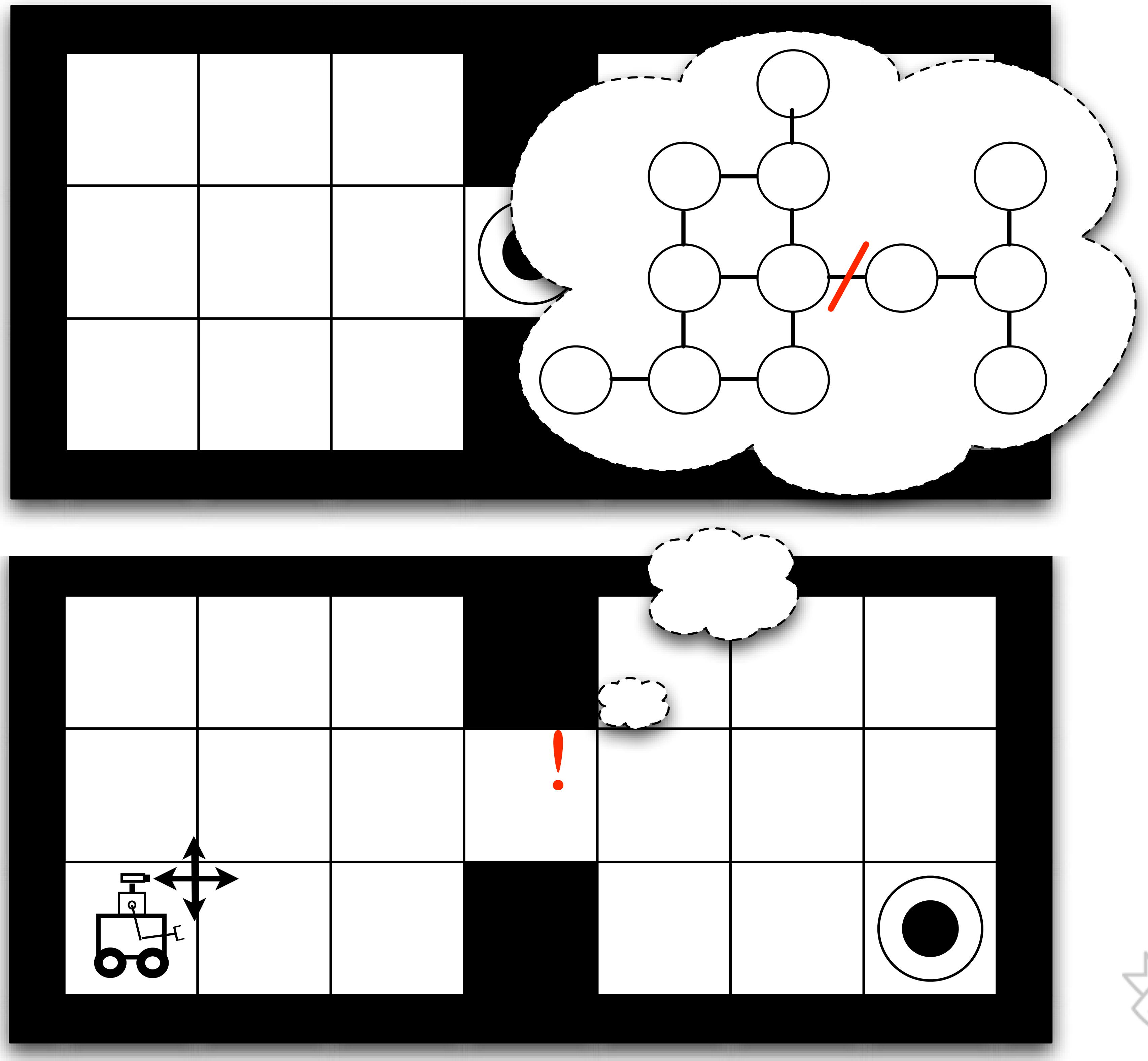
RL typically solves a *single* problem *monolithically*.

Hierarchical RL:

- Create and use higher-level macro-actions.
- Problem now contains subproblems.
- Each subproblem is also an RL problem.

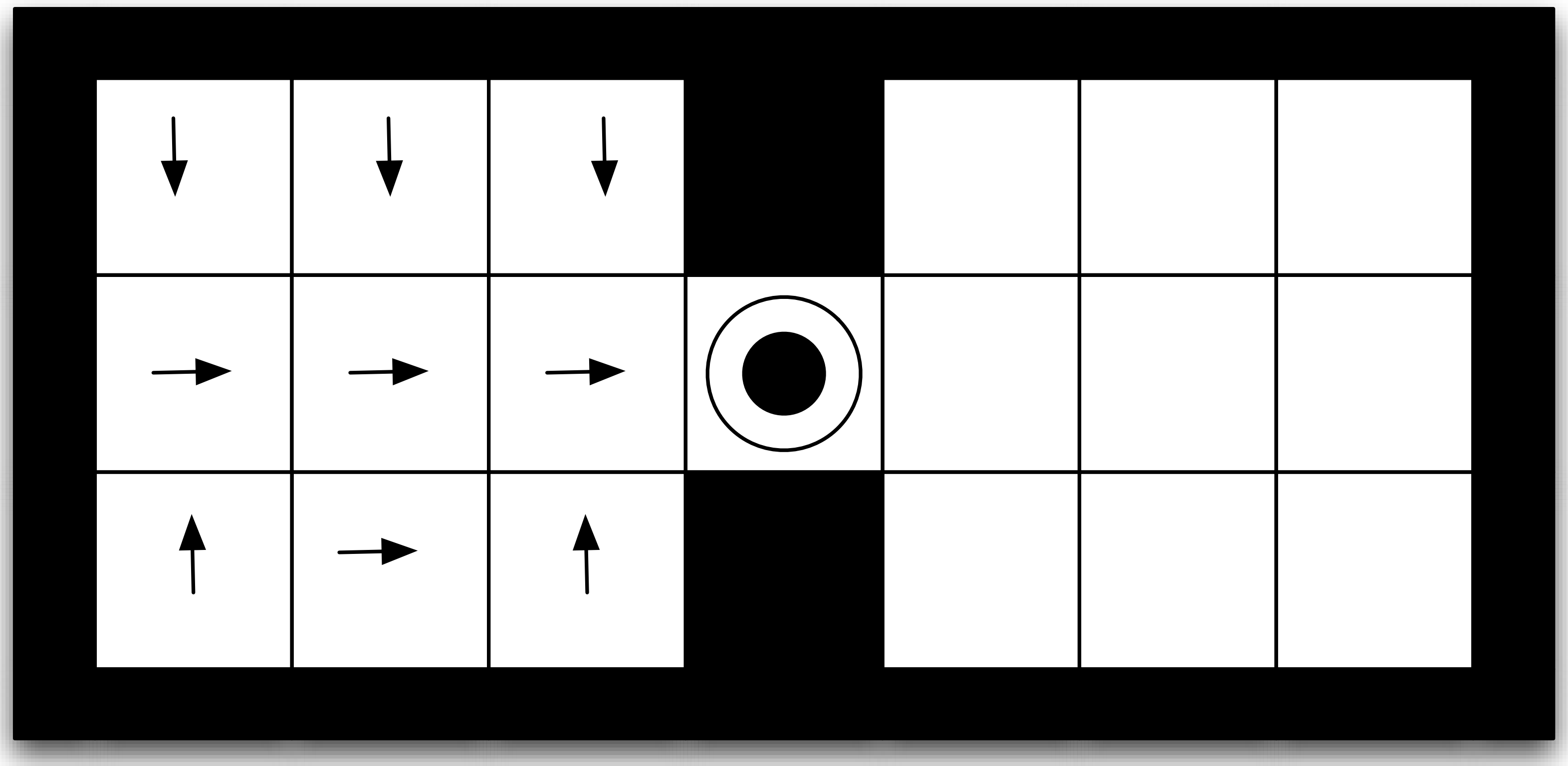
Options Framework: theoretical basis for skill acquisition, learning and planning using higher-level actions (options).

Hierarchical RL

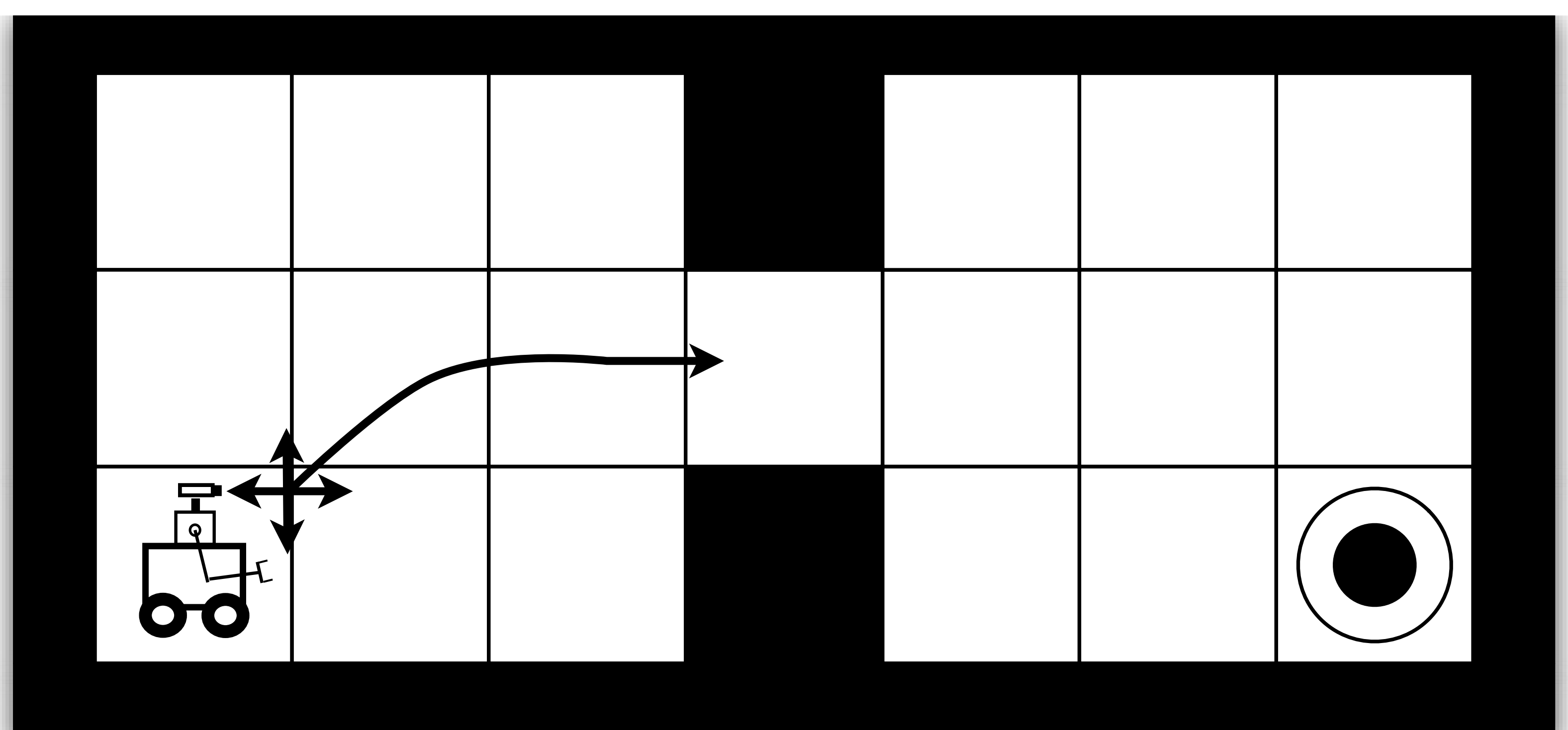


Hierarchical RL

Skill



Problem



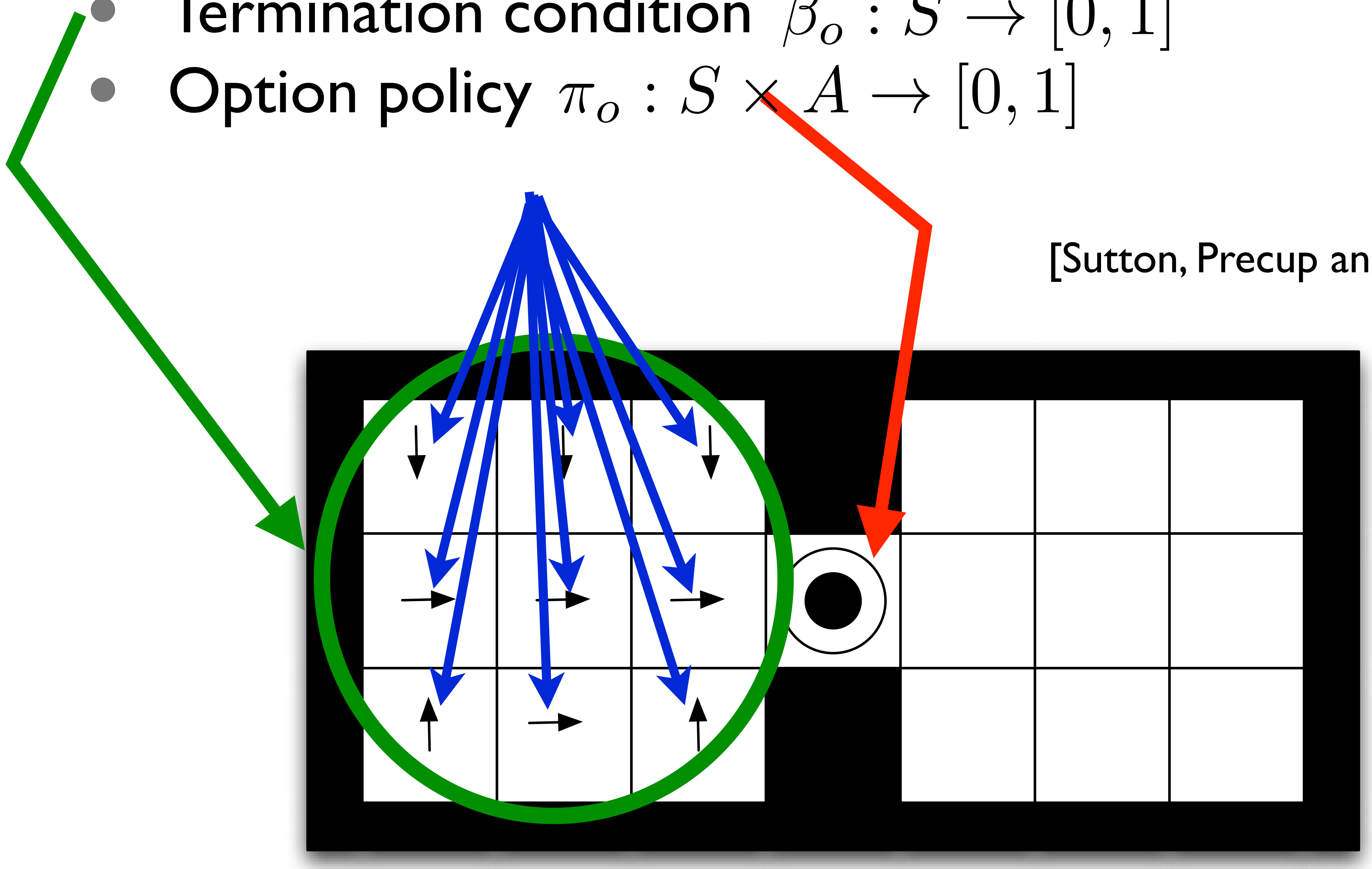
The Options Framework

An option is one formal model of a skill.

An option *o* is a policy unit:

- Initiation set $I_o : S \rightarrow \{0, 1\}$
- Termination condition $\beta_o : S \rightarrow [0, 1]$
- Option policy $\pi_o : S \times A \rightarrow [0, 1]$

[Sutton, Precup and Singh 1999]



Actions as Options

A primitive action *a* can be represented by an option:

- $I_a(s) = 1, \forall s \in S$
- $\beta_a(s) = 1, \forall s \in S$
- $\pi_a(s, b) = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$

A primitive action can be executed anywhere, lasts exactly one time step, and always chooses action *a*.

Questions

Given an MDP:

$$(S, A, R, T, \gamma)$$

... let's replace A with a set of options O (some of which may be primitive actions).

- How do we characterize the resulting problem?
- How do we plan using options?
- How do we learn using options?
- How do we characterize the resulting policies?

SMDPs

The resulting problem is a *Semi-(Markov Decision Process)*.
This consists of:

- S Set of states
- O Set of options
- $P(s', t|o, s)$ Transition model
- $R(s', s, t)$ Reward function
- γ Discount factor (per step)

In this case:

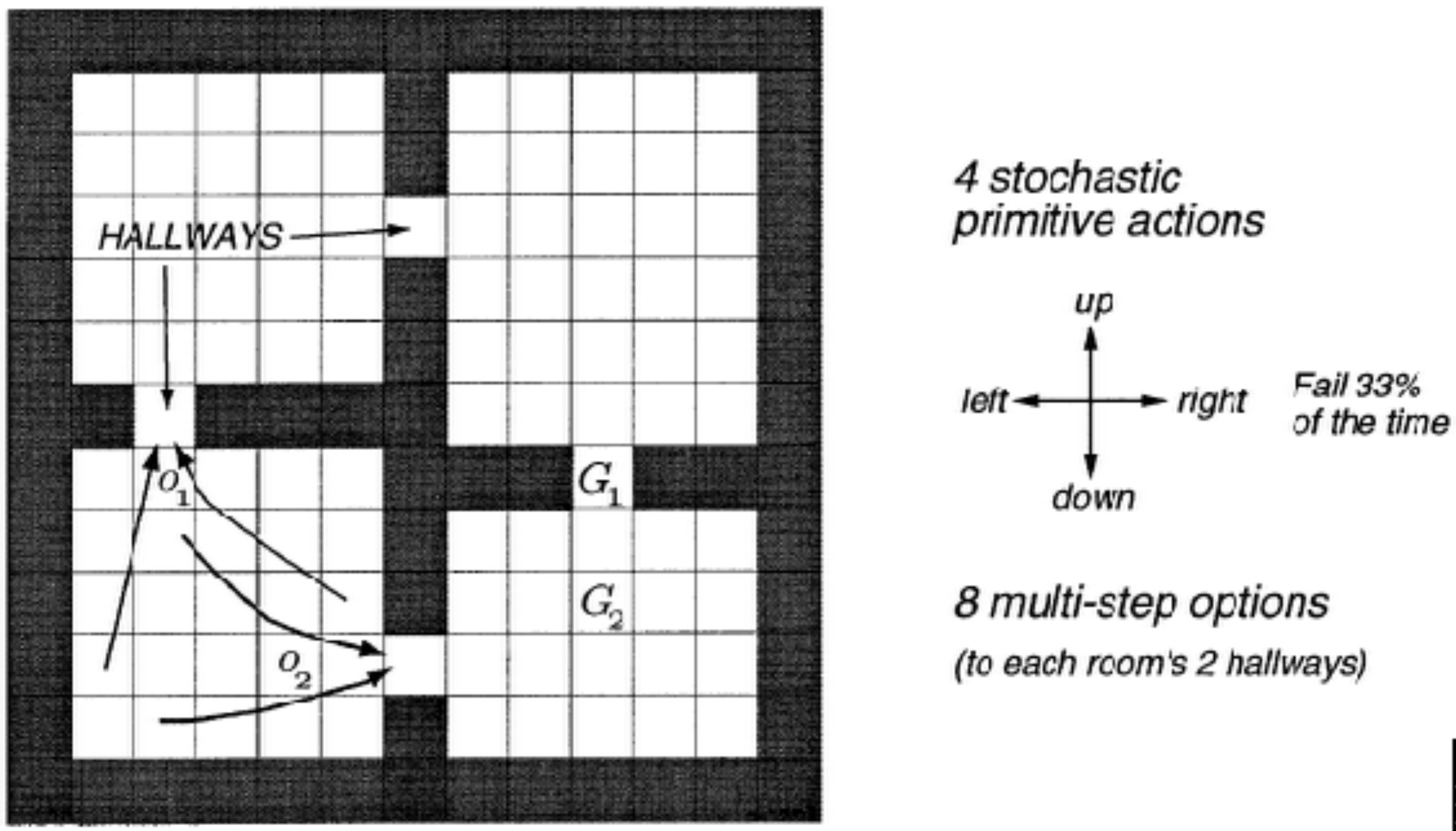
- All times are natural numbers.
- "Semi" here means transitions can last *t* timesteps.
- Transition and reward function involve time taken for option to execute.

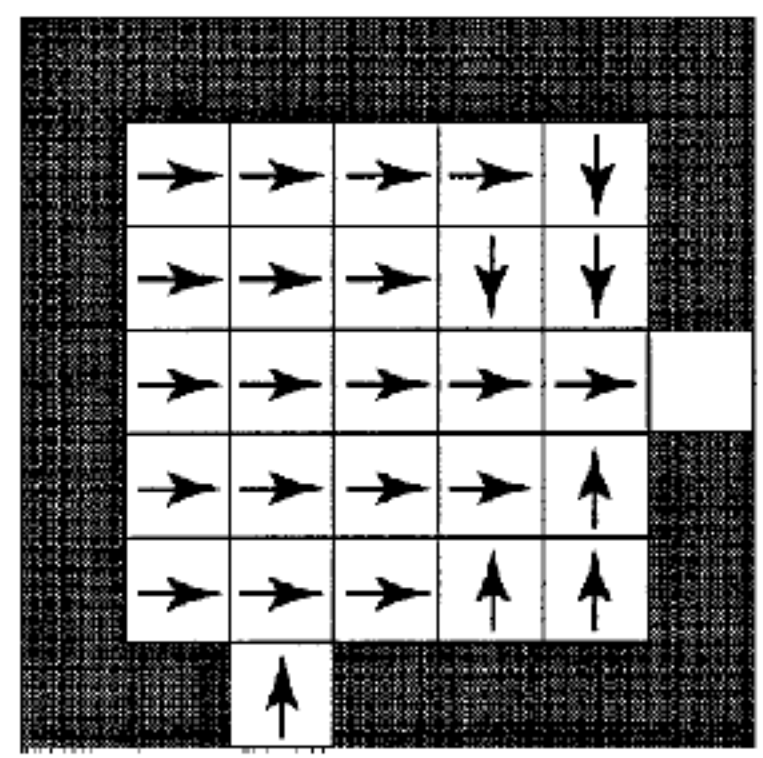
Easy

$$Q^{\pi}(s,o) = \mathbb{E}_{t,s'}[R(s',s,t)] + \mathbb{E}_{t,s'}[\gamma^{t}\pi(s',o')Q^{\pi}(s',o')]$$

All things flow from Bellman.

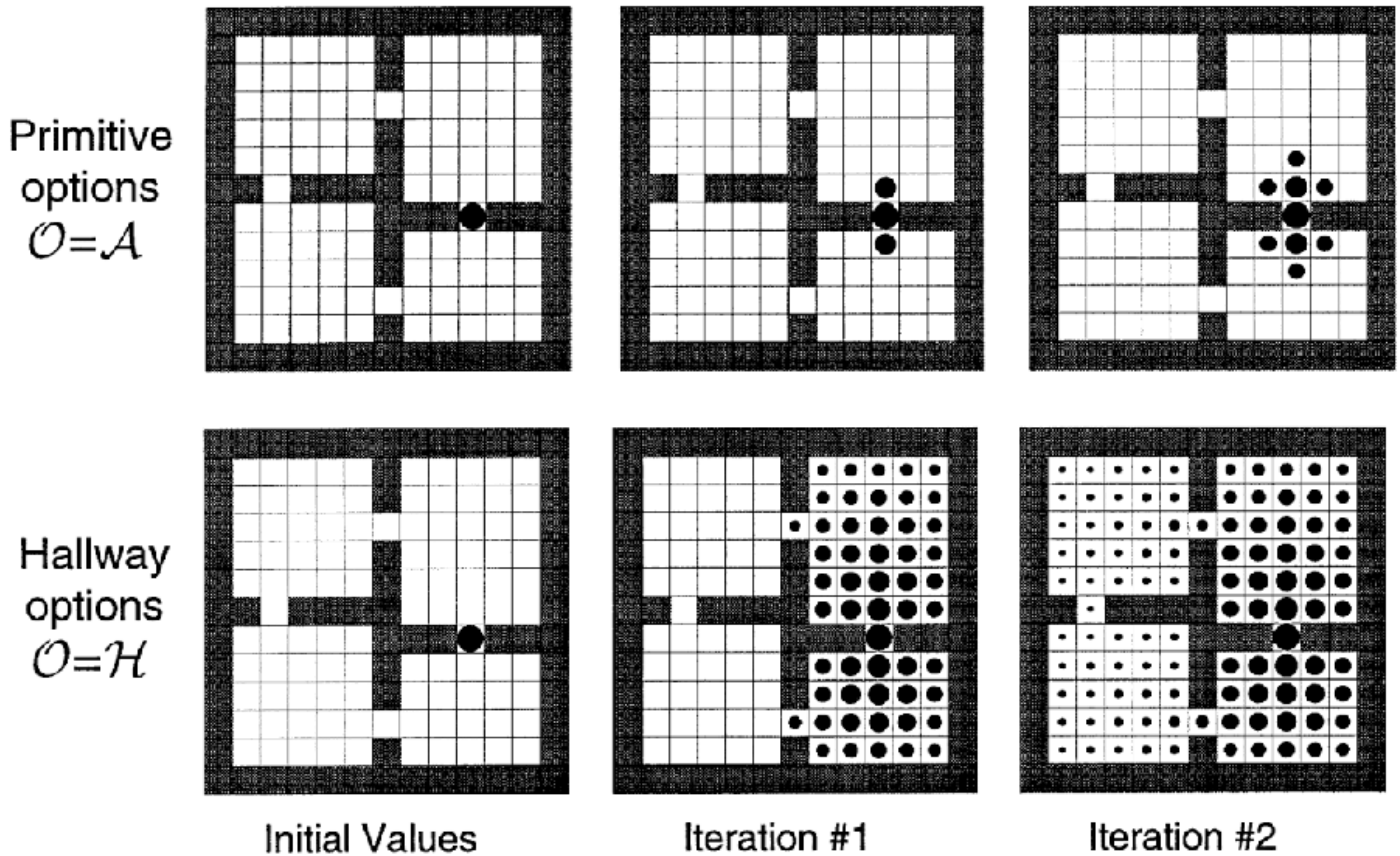
Example





(Sutton, Precup and Singh, AIJ 1999)

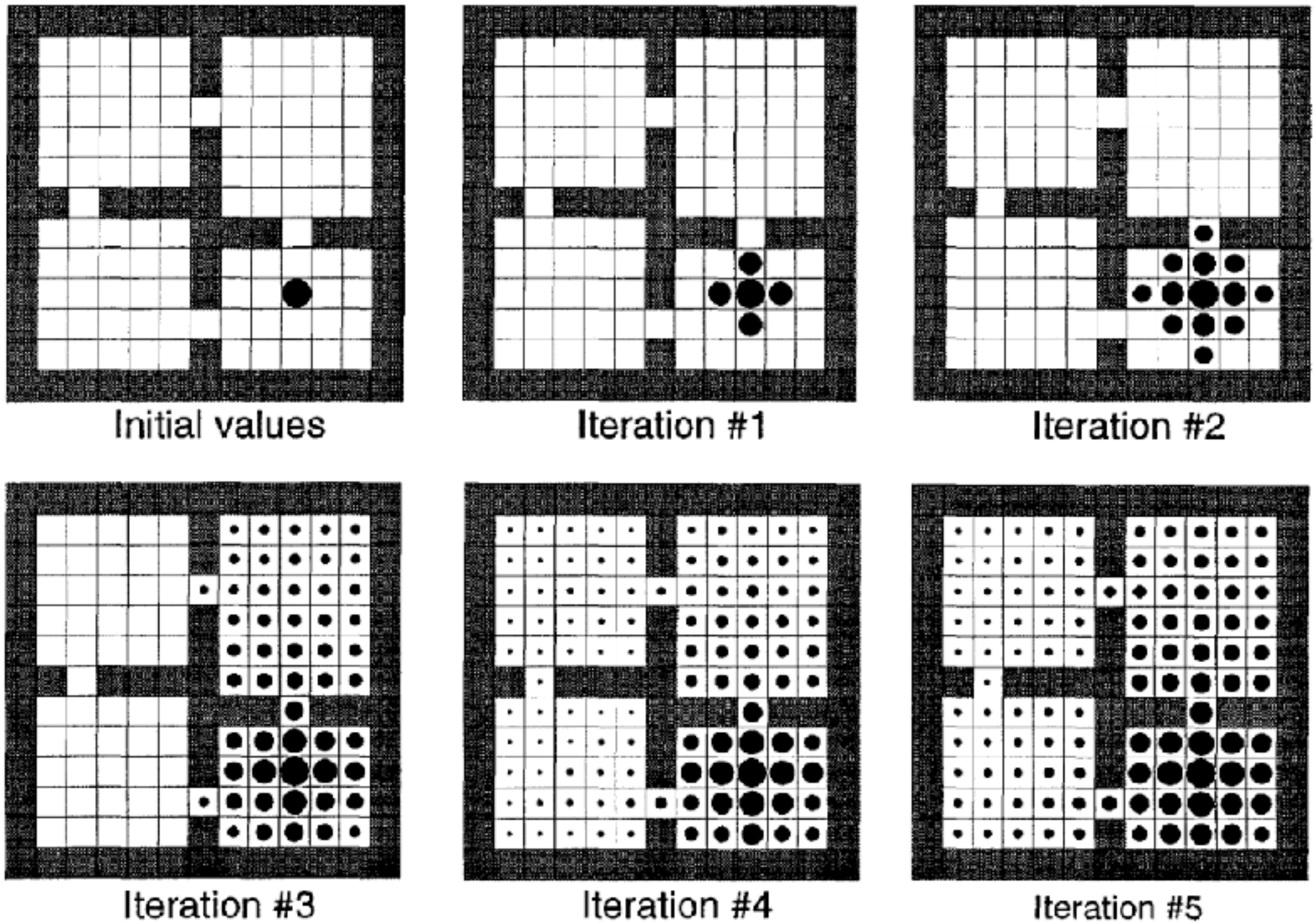
Example



(Sutton, Precup and Singh, AIJ 1999)

Example

Primitive and hallway options $\mathcal{O}=\mathcal{A}\cup\mathcal{H}$



(Sutton, Precup and Singh, AIJ 1999)

What are Skills For?

Lots of things!

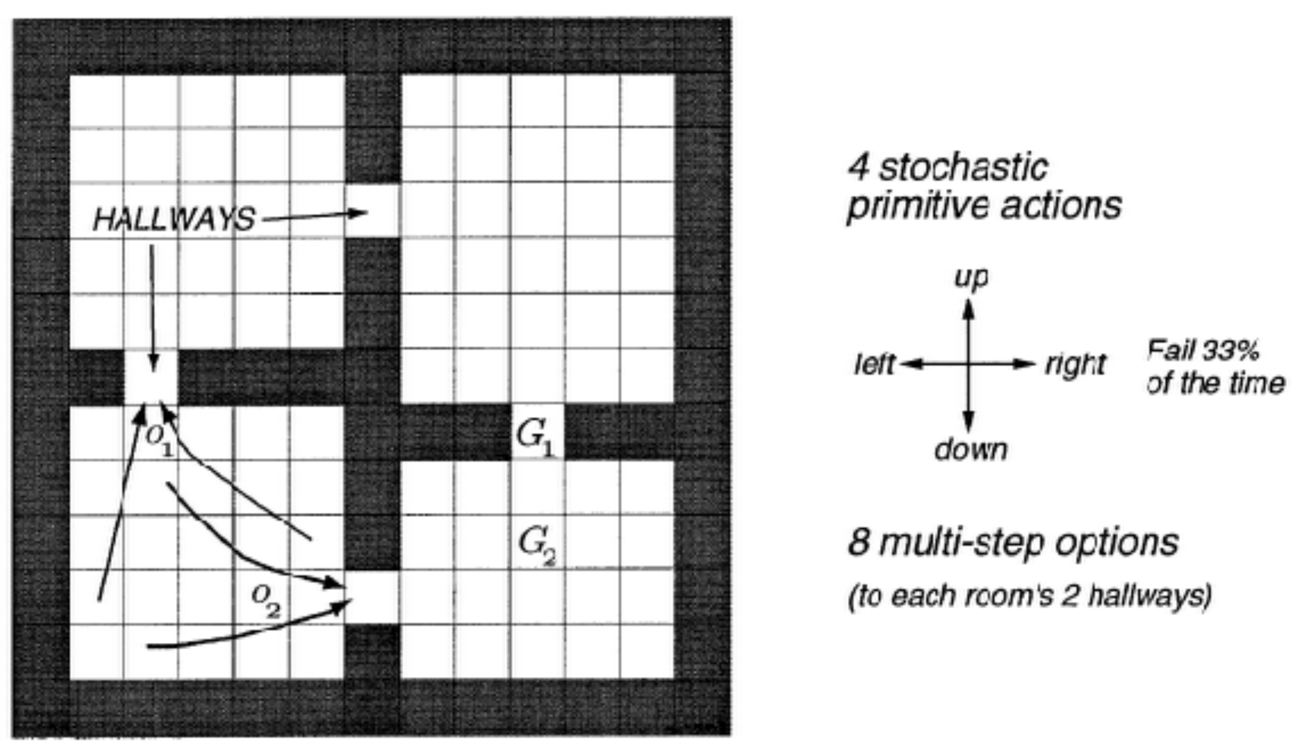
A few salient points:

- Rewiring.
- Transfer.
- Skill-Specific Abstractions.

Rewiring

Adding an option changes the connectivity of the MDP.
This affects:

- Learning and Planning.
- Exploration.
- State-visit distribution.
- *Diameter of problem.*



(Sutton, Precup and Singh, AIJ 1999)

Transfer

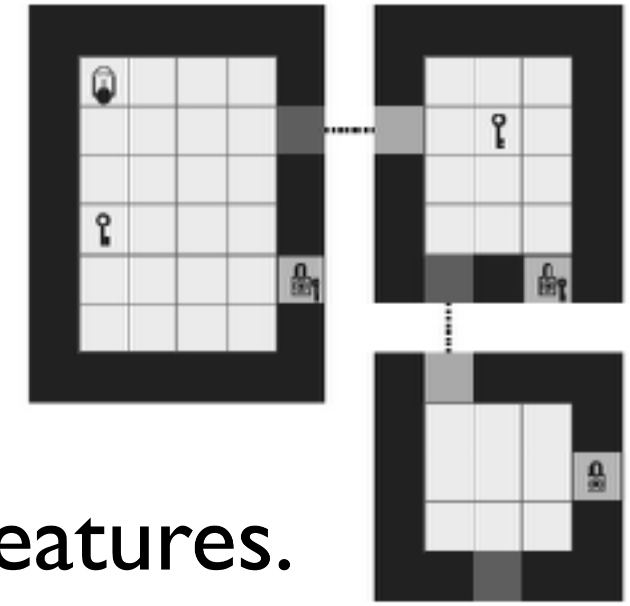
Use experience gained while solving one problem to improve performance in another.

Skill transfer:

- Use options as mechanism for transfer.
- Transfer *components* of solution.
- Can drastically improve performance
- ... even if it takes a lot of effort to learn them.

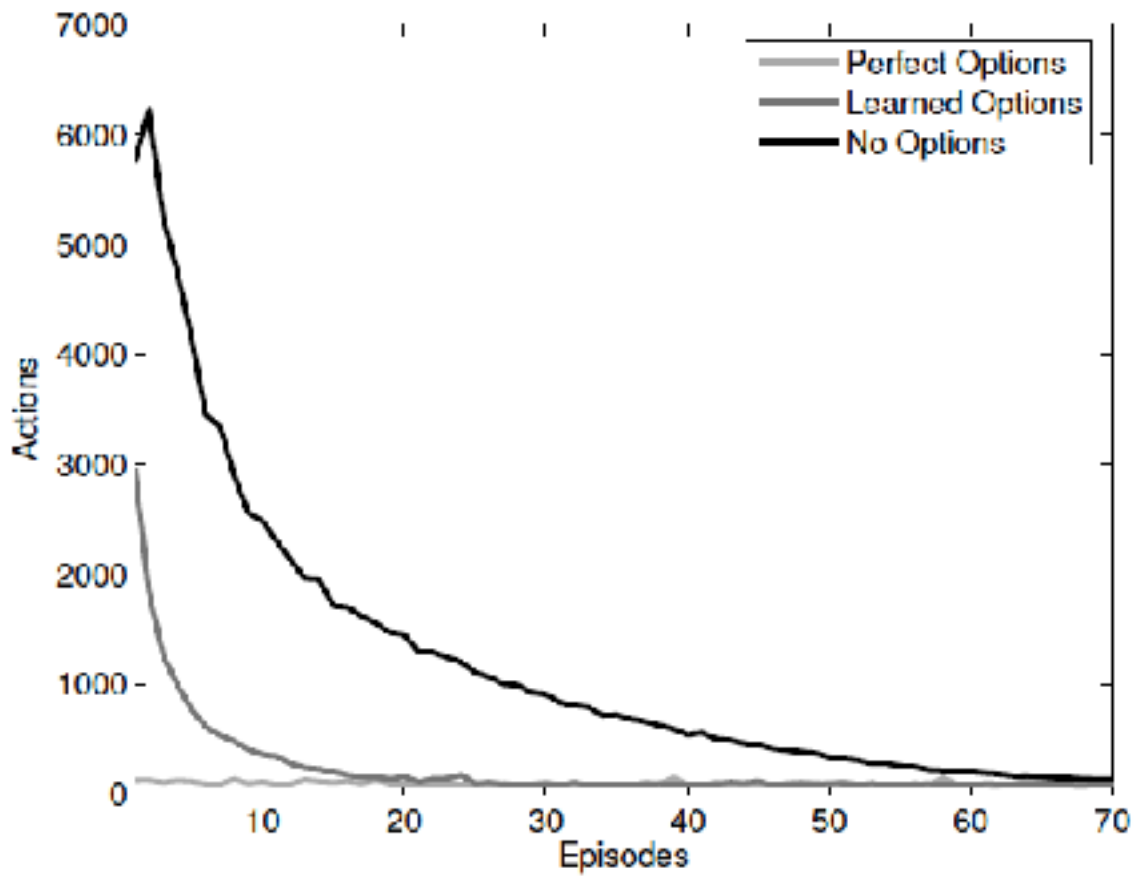
General principle: **subtasks recur.**

Transfer

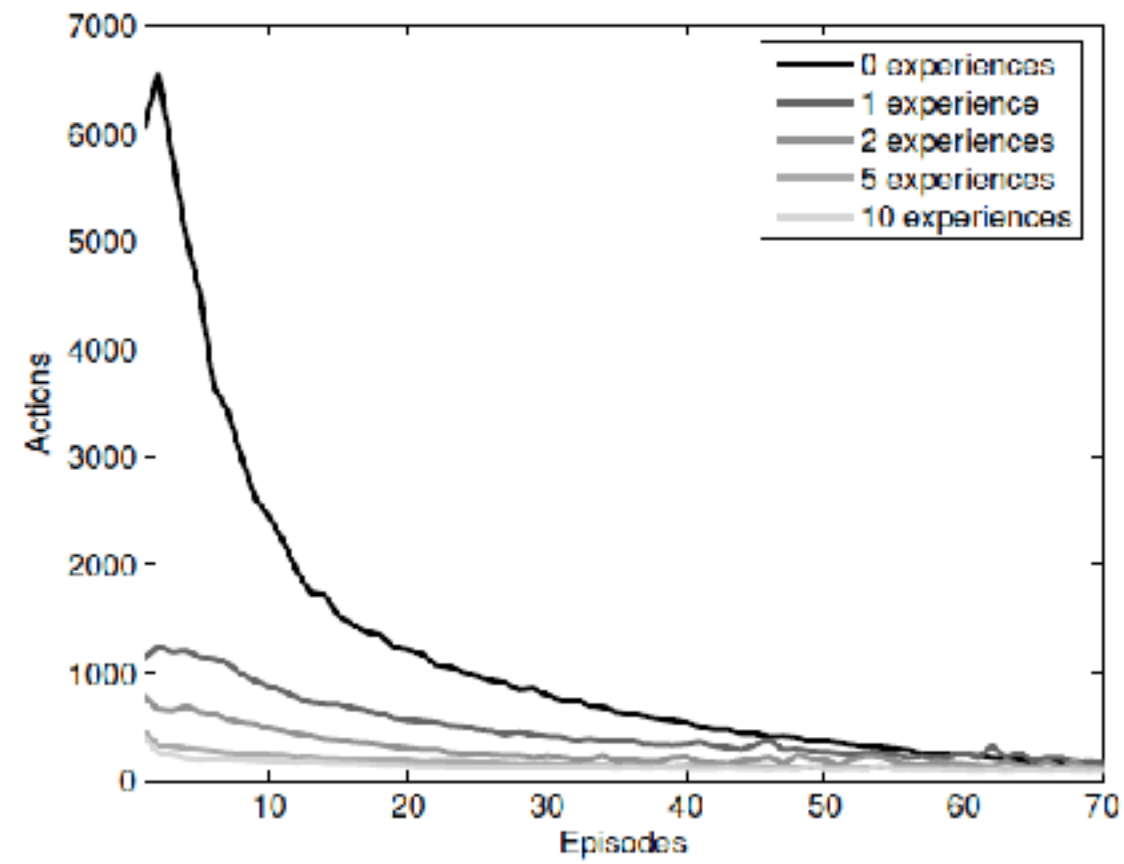


Tasks drawn from parametrized family.

- Common features present.
- Options defined using only common features.



(a) Learning curves for agents with problem-space options.



(b) Learning curves for agents with agent-space options, with varying numbers of training experiences.

(Konidaris and Barto, IJCAI 2007)

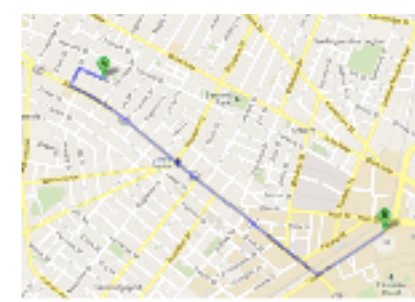
Skill-Specific Abstractions

Options provide opportunities for abstraction

- Split high-dimensional problem into subproblems ...
- ... such that each one supports a solution using an abstraction.








Working hypothesis: *behavior is piecewise low-dimensional.*

Skill Discovery

Where do skills come from?

Discover options autonomously, through interaction with an environment.

- Typically *subgoal options*.
- This means that we must determine β_o.
- Sometimes also R_o.

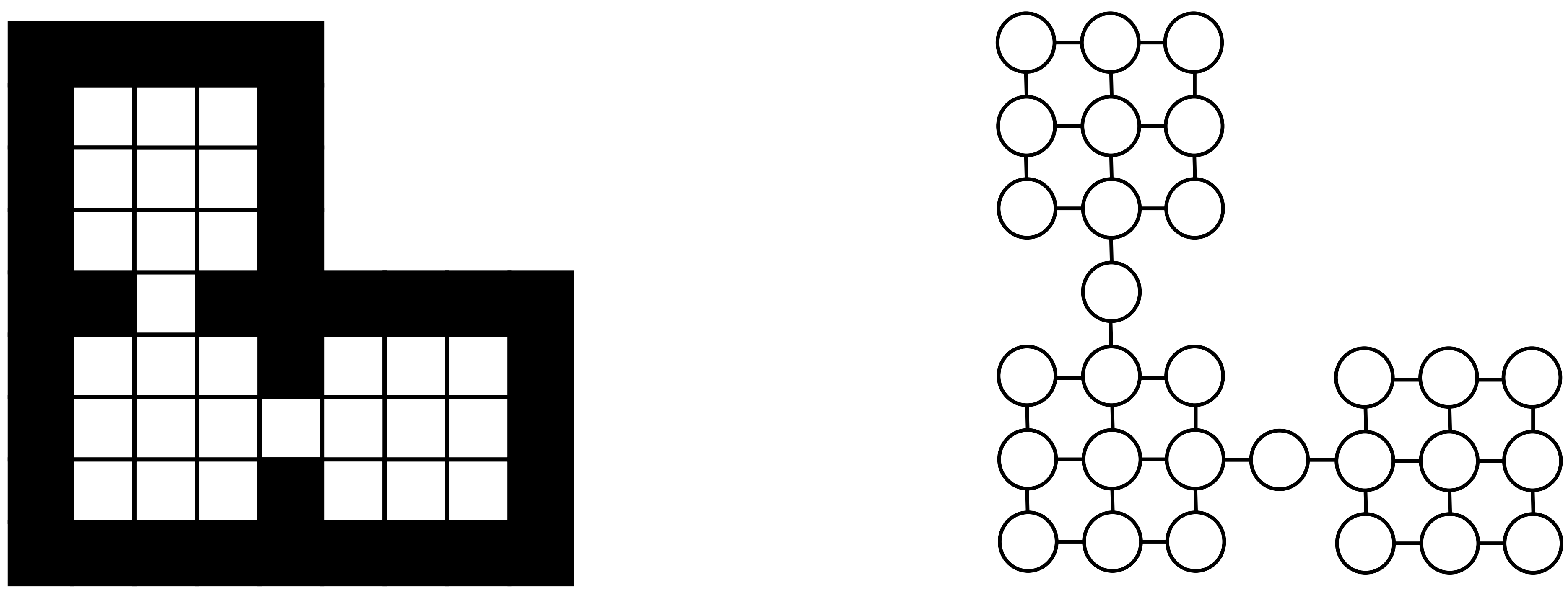
The question then becomes:

- Which states are good subgoals?

Betweenness Centrality

Consider an MDP as a graph.

- States are vertices.
- Edges indicate possible transition between two states.



Further, let us assume a task distribution over start states and goal pairs:

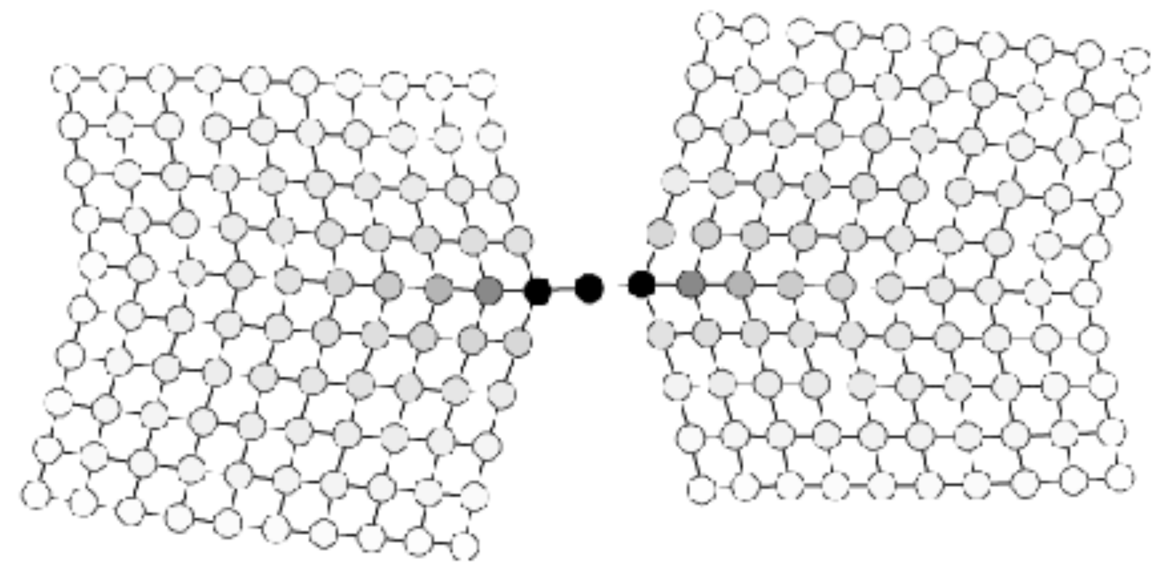
- $P_T(s, e)$

(Simsek and Barto, 2008)

Betweenness Centrality

We can define the *betweenness centrality* of a vertex (state) as:

$$\sum_{s,e} \frac{\sigma_{se}(v)}{\sigma_{se}} w_{se}$$



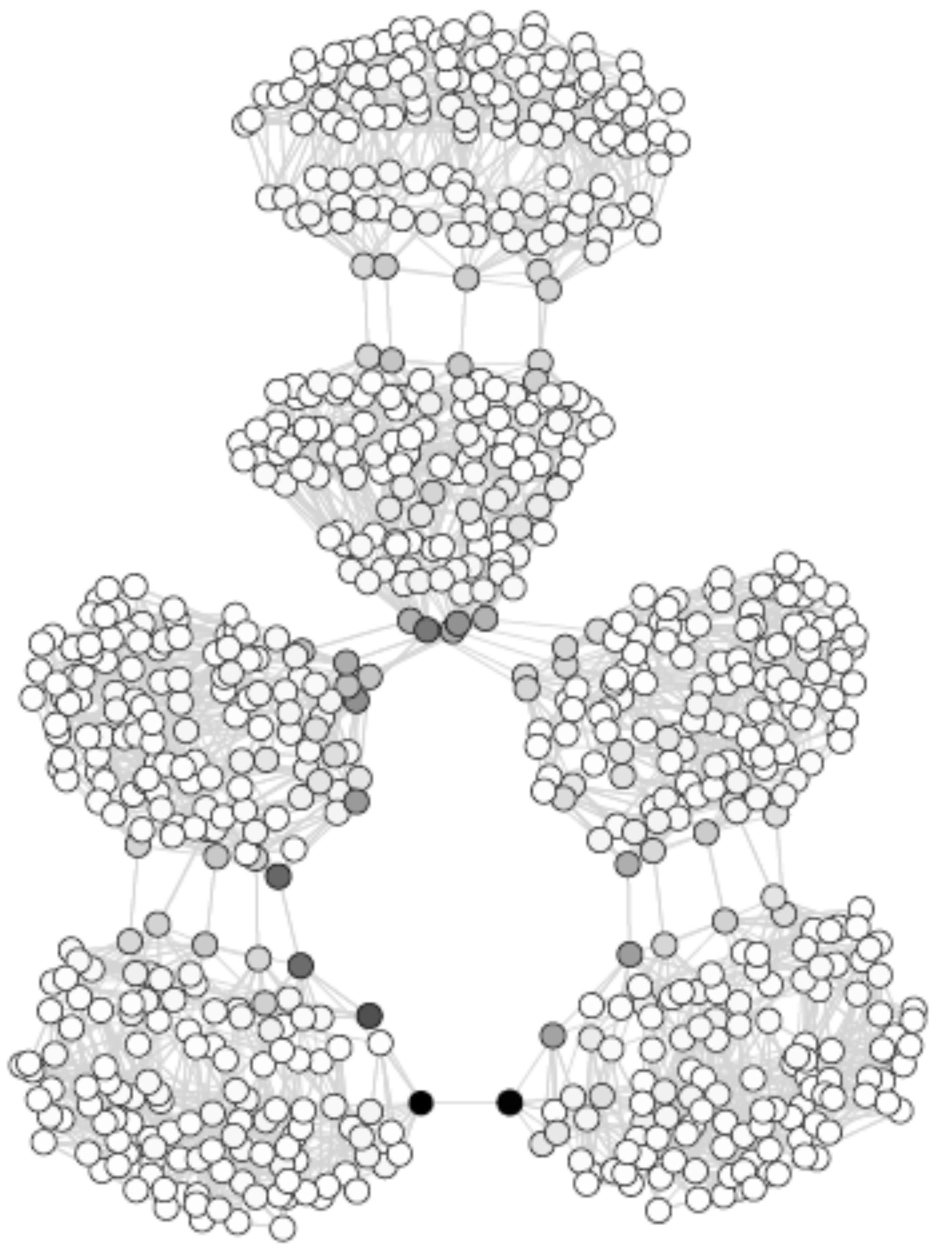
This indicates it probability of being on a shortest path from s to e; if we define:

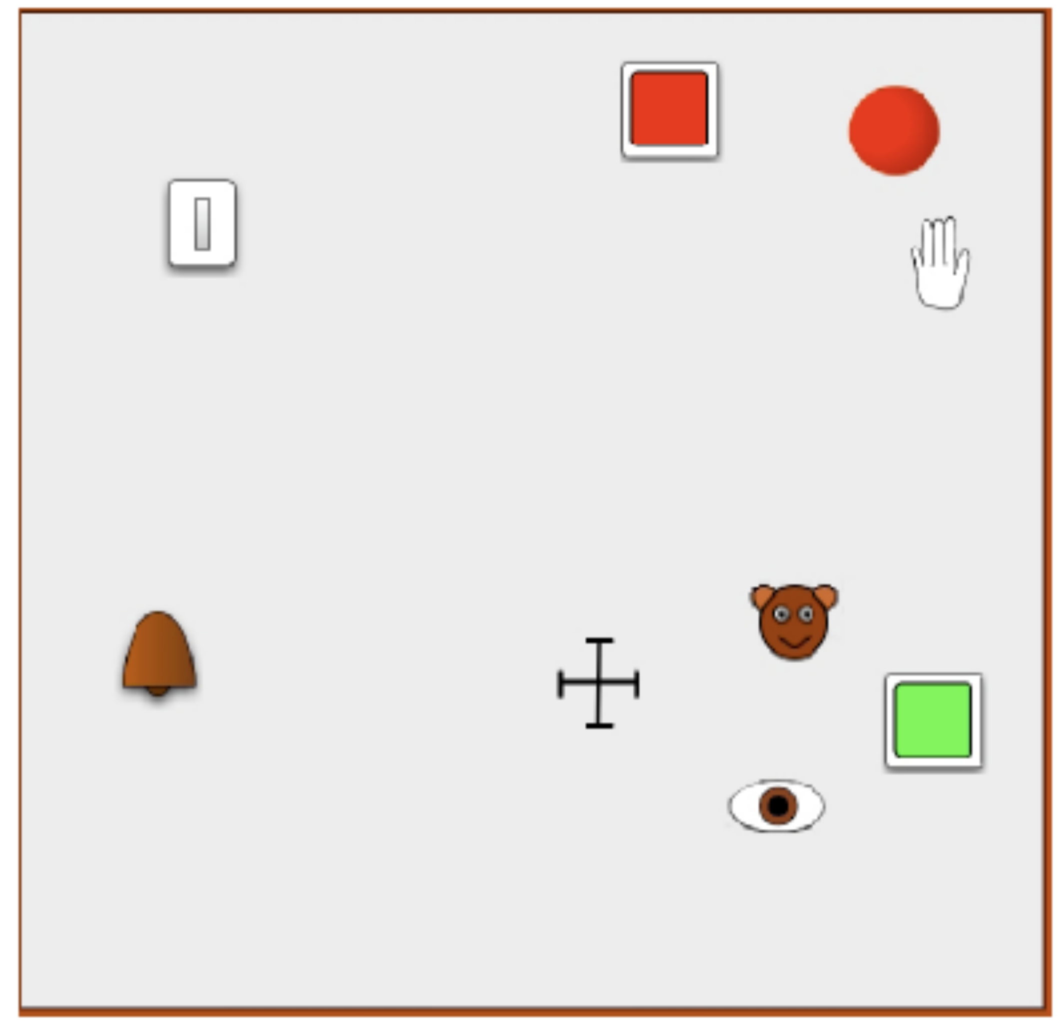
- *Shortest path* as *optimal solution*.
- $w_{se} = P_T(s, e)$

... then we get something sensible for RL.

(Simsek and Barto, 2008)

Betwenness Centrality

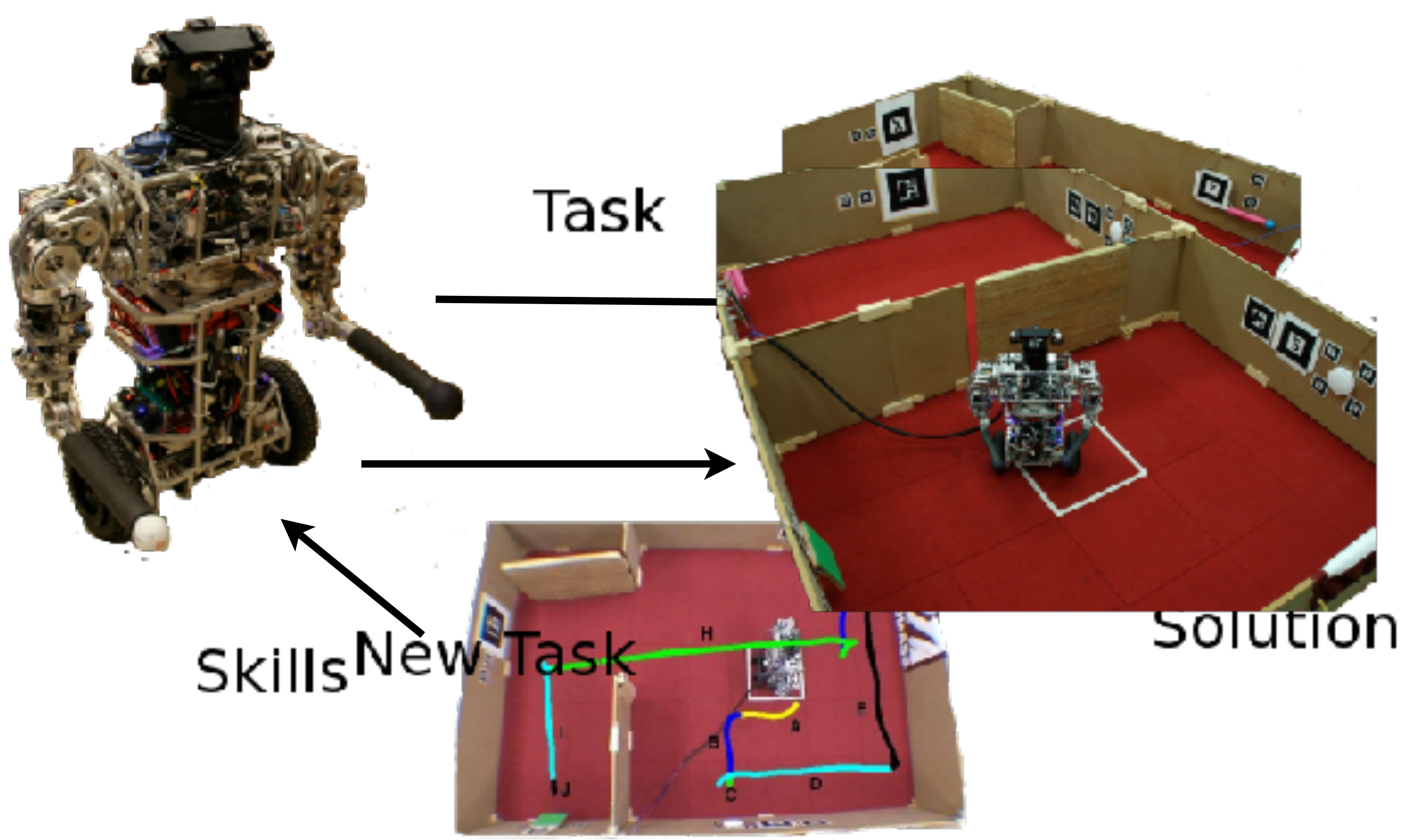




(Simsek and Barto, 2008)

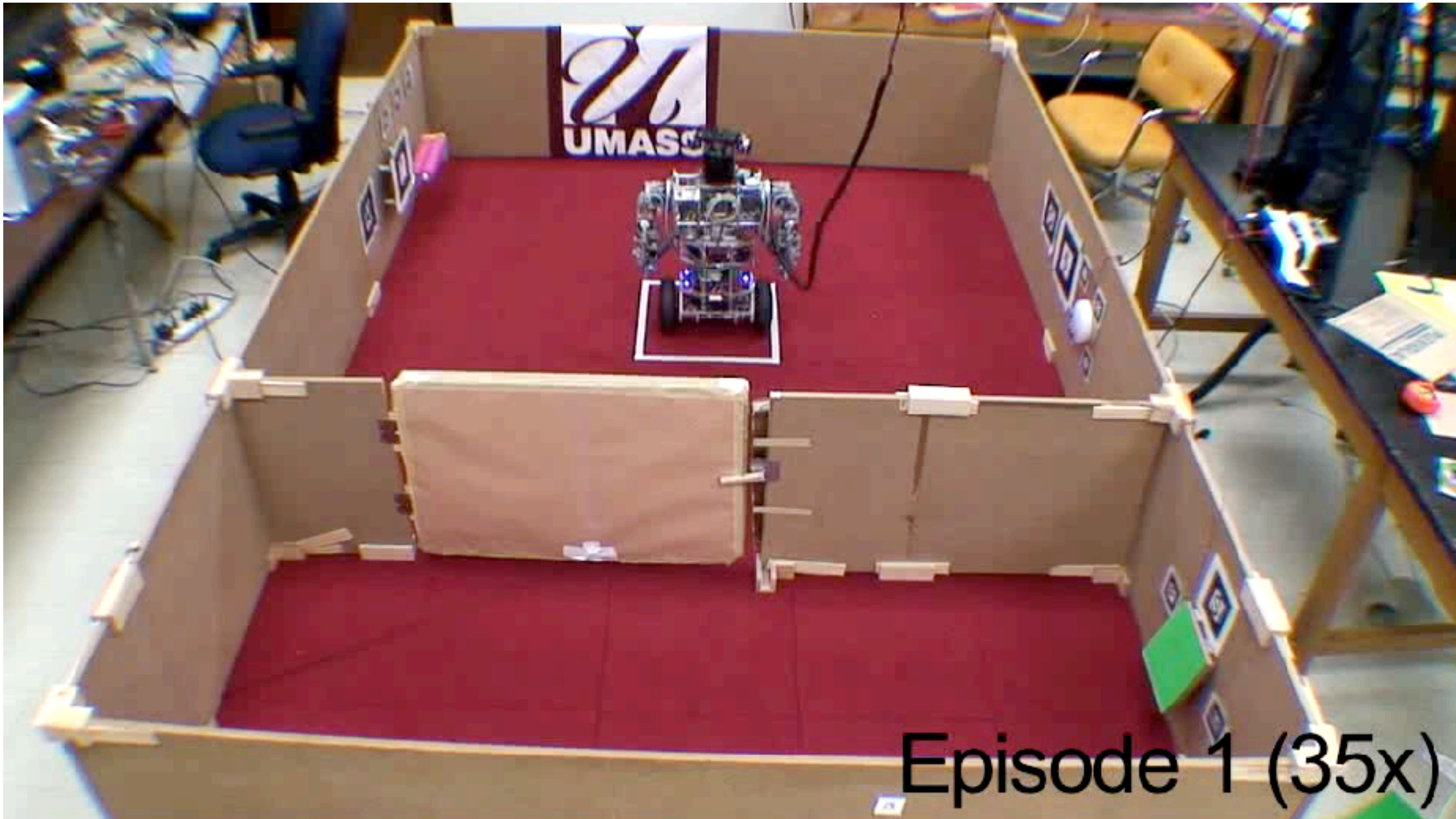
Skill Acquisition

- A robot learning to solve a task
- Extracting skills from solution
- Deploying them in a new task

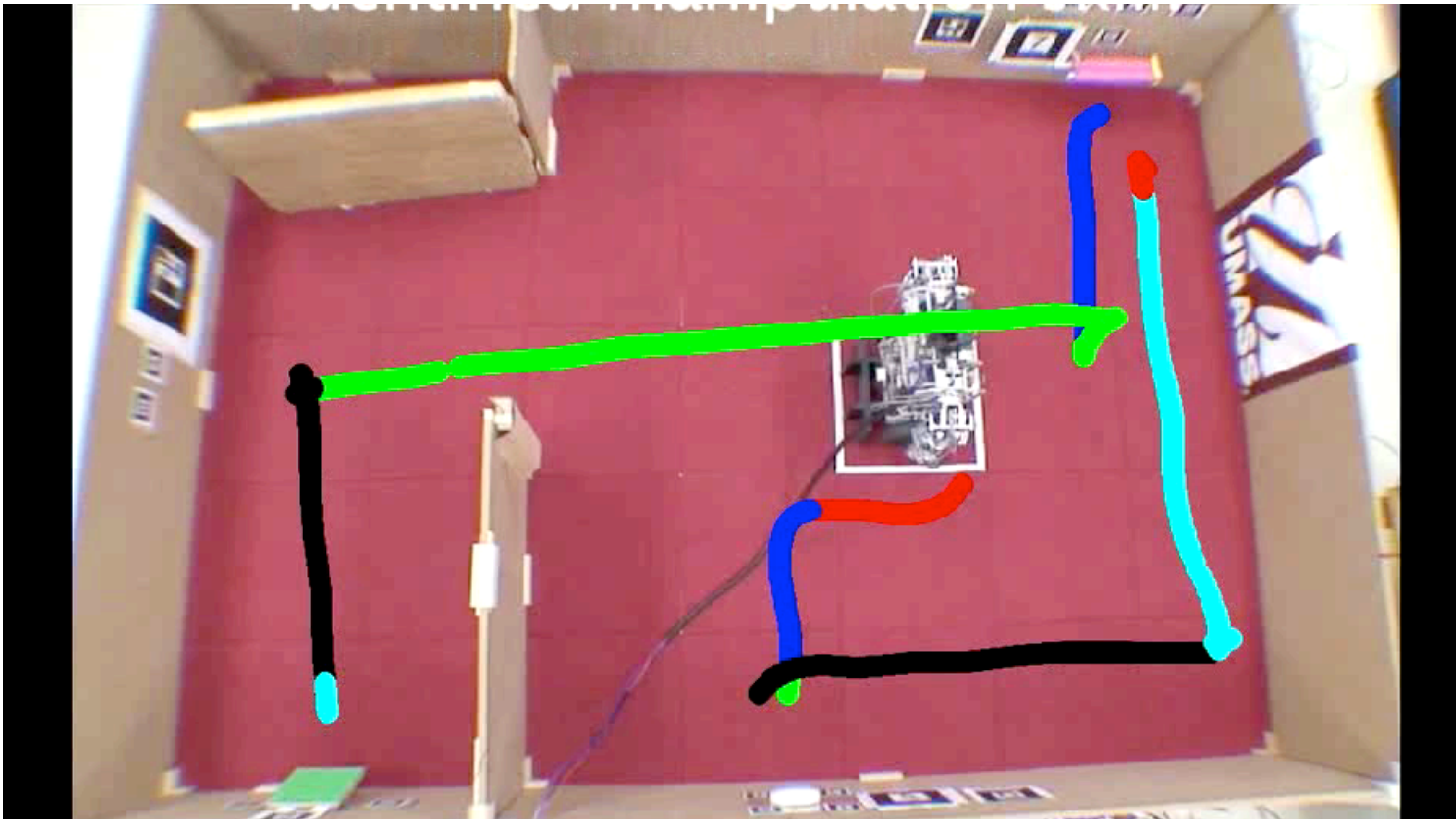


[Konidaris et al., 2011]

Training Room

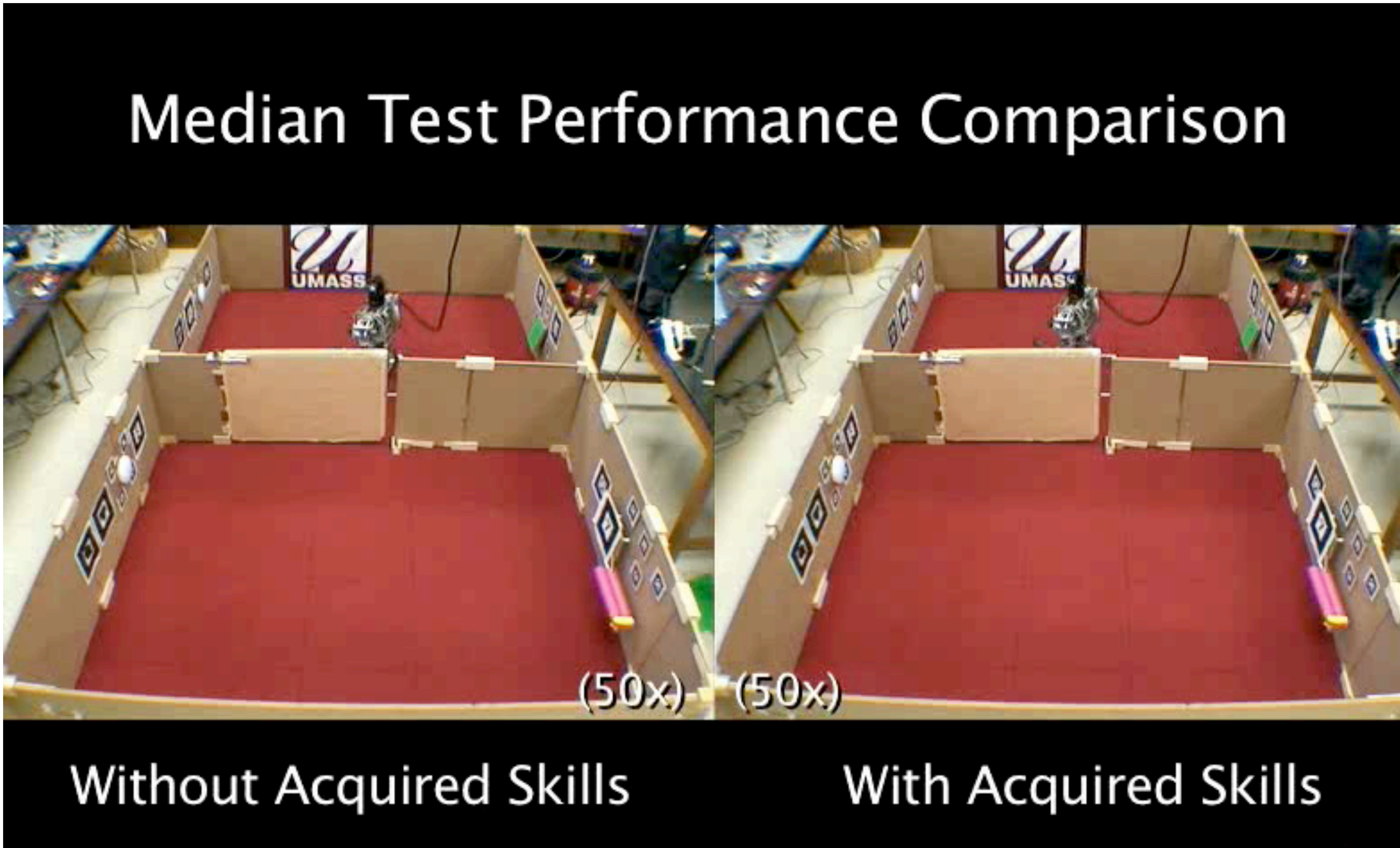


Acquired Skills

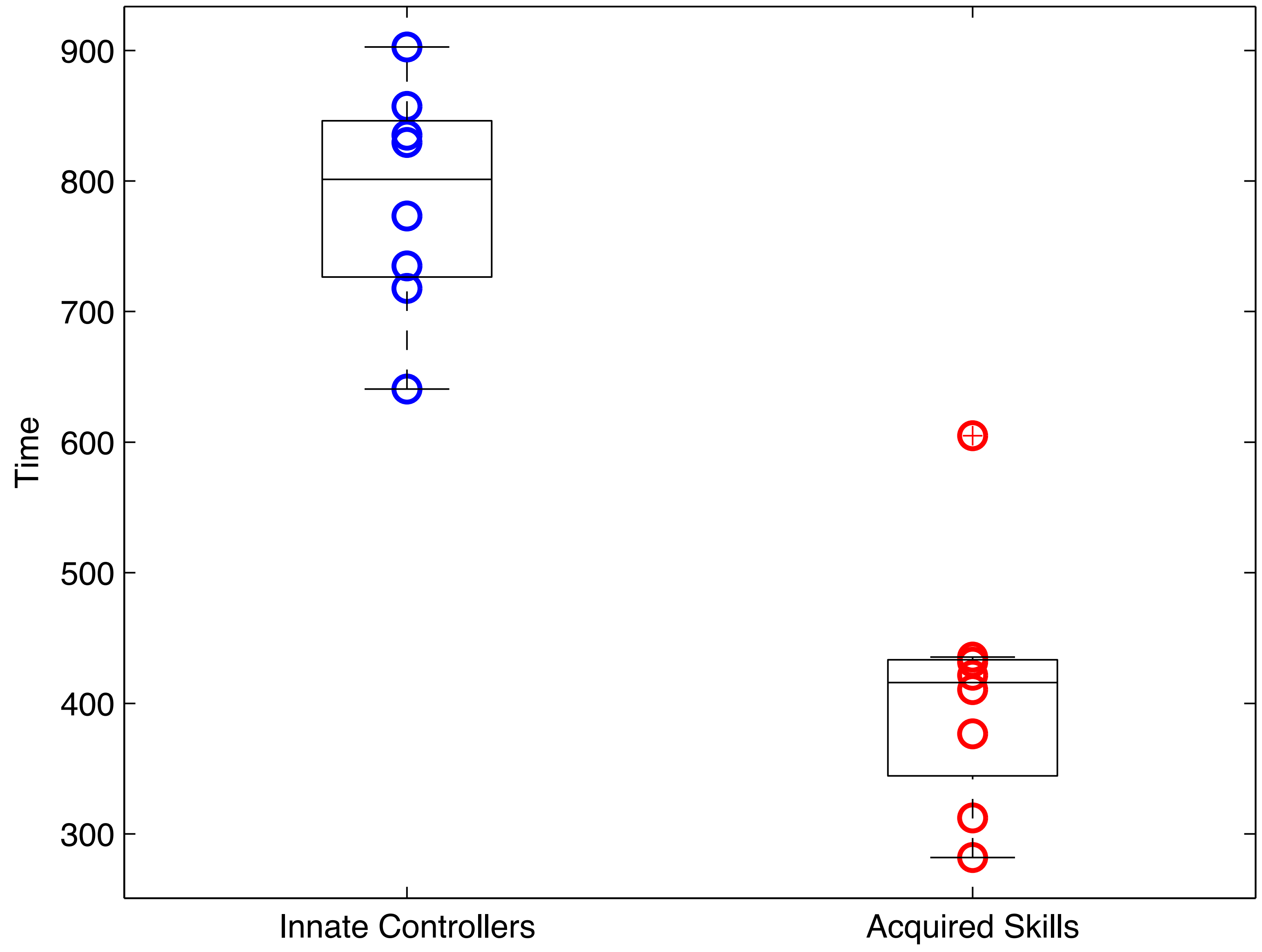


The Test Room

The Test Room

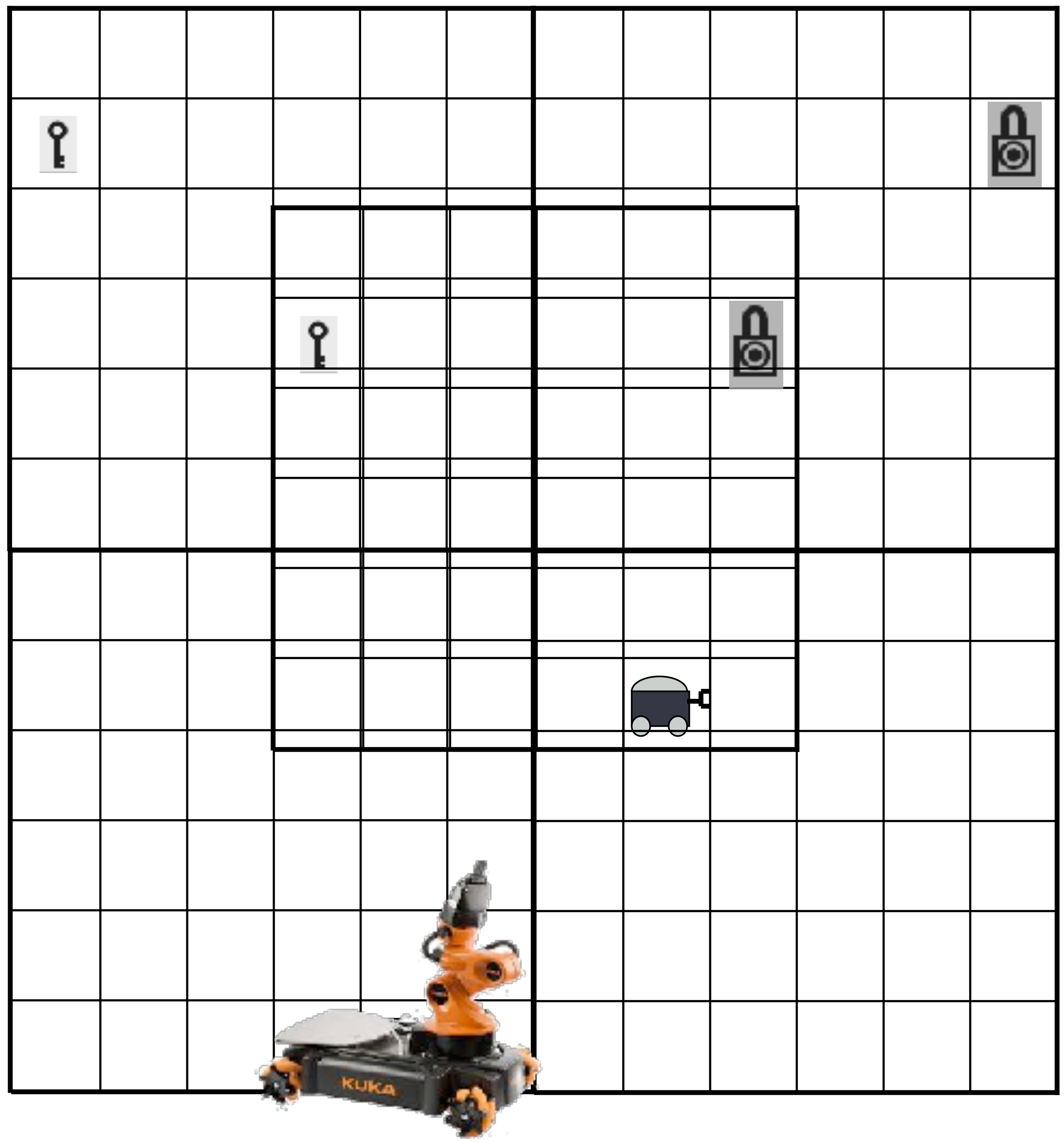


The Test Room



[Konidaris et al., 2011]

State Abstraction



Key Idea

How can we create a model of an environment that is maximally abstract but still allows the agent to plan?

What is the fundamental question of probabilistic planning?

Given a state and a sequence of high-level actions:

- What is the probability of being able to execute it?
- What is the expected reward?

[Konidaris et al., 2014, 2015]

Symbols for Planning

A plan $p = \{o_1, ..., o_n\}$ from a state distribution Z is a sequence of actions to be executed from a state drawn from Z.

Starting from the corridor ...

- GoToDoor
- TurnHandle
- PushDoorOpen
- EnterRoom ...

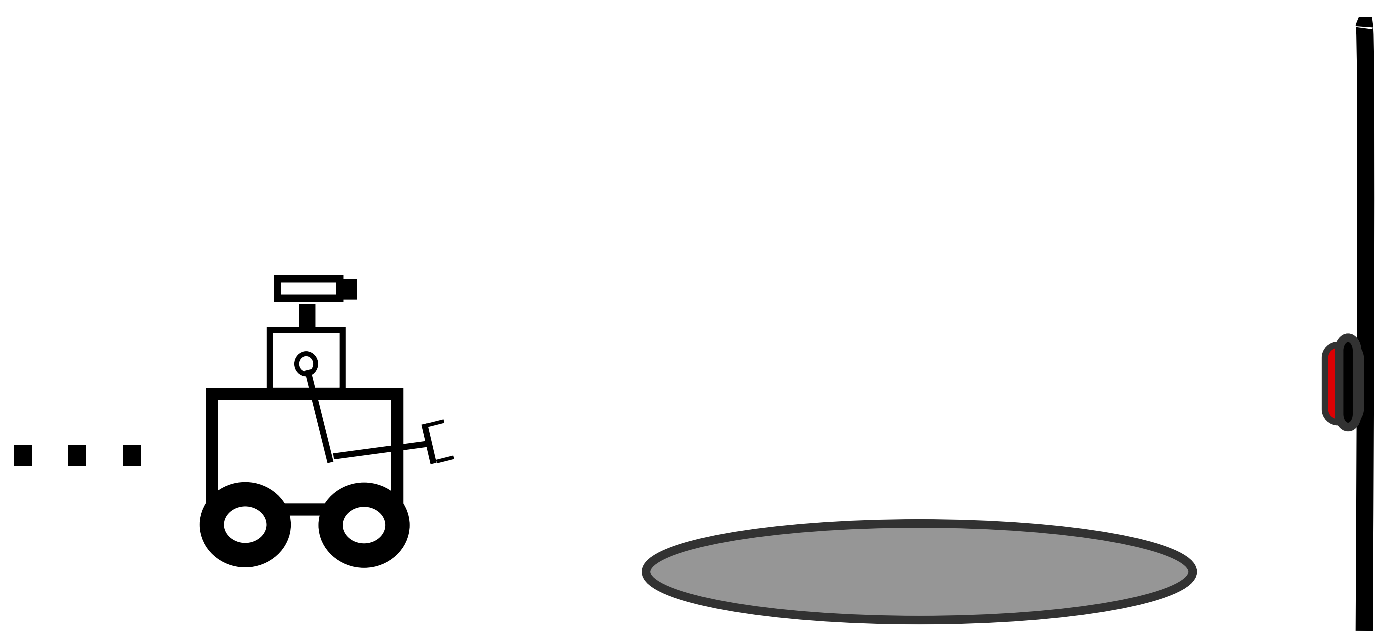
So:

- **Which mathematical objects do we need to determine the probability with which we can execute any plan p?**

Symbols for Planning

We need **one classifier** and one operator per skill.

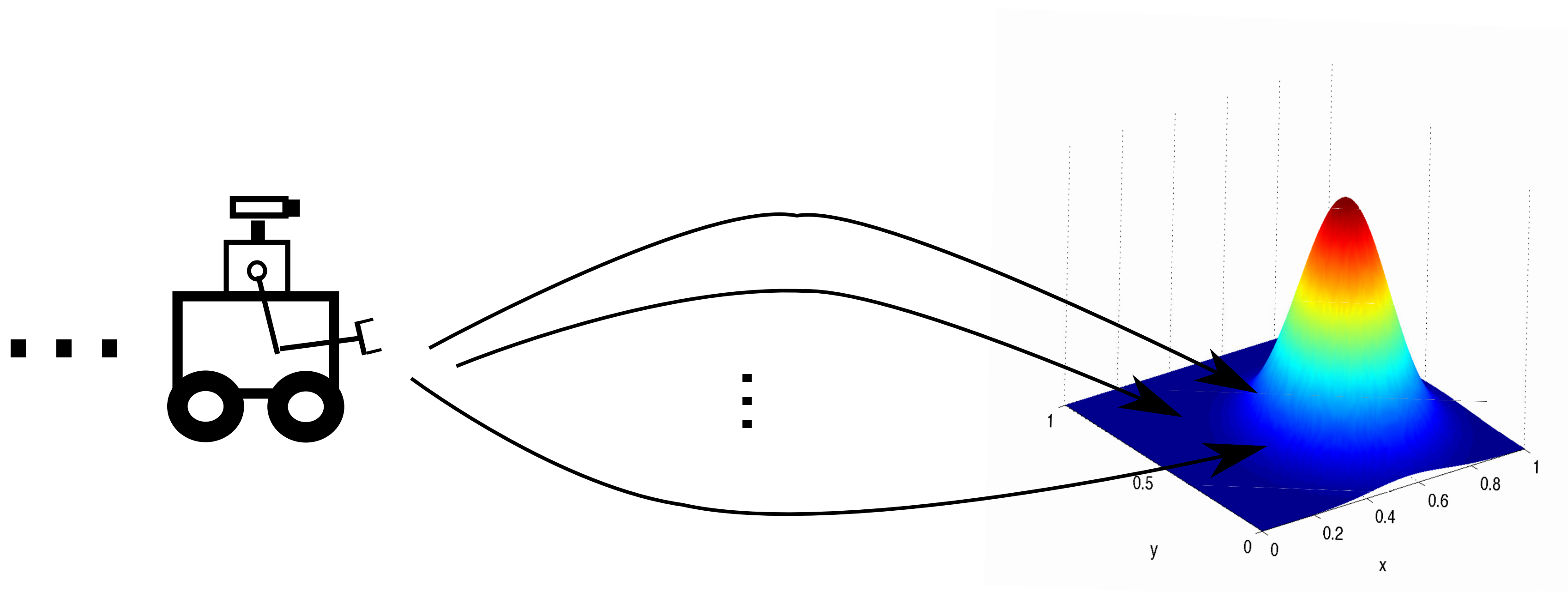
Initiation classifier:



Symbols for Planning

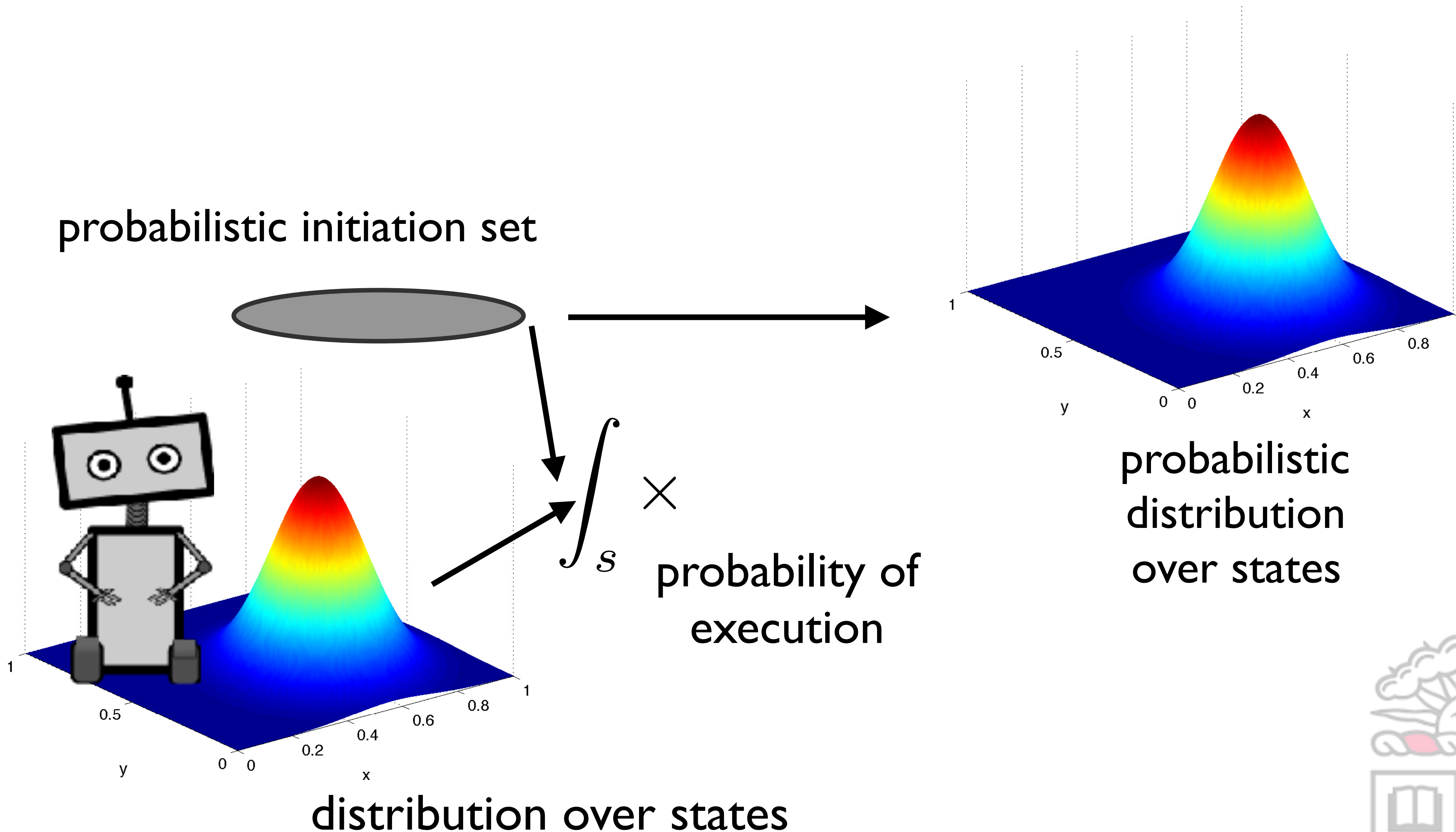
We need one classifier and **one operator** per skill.

Image distribution:



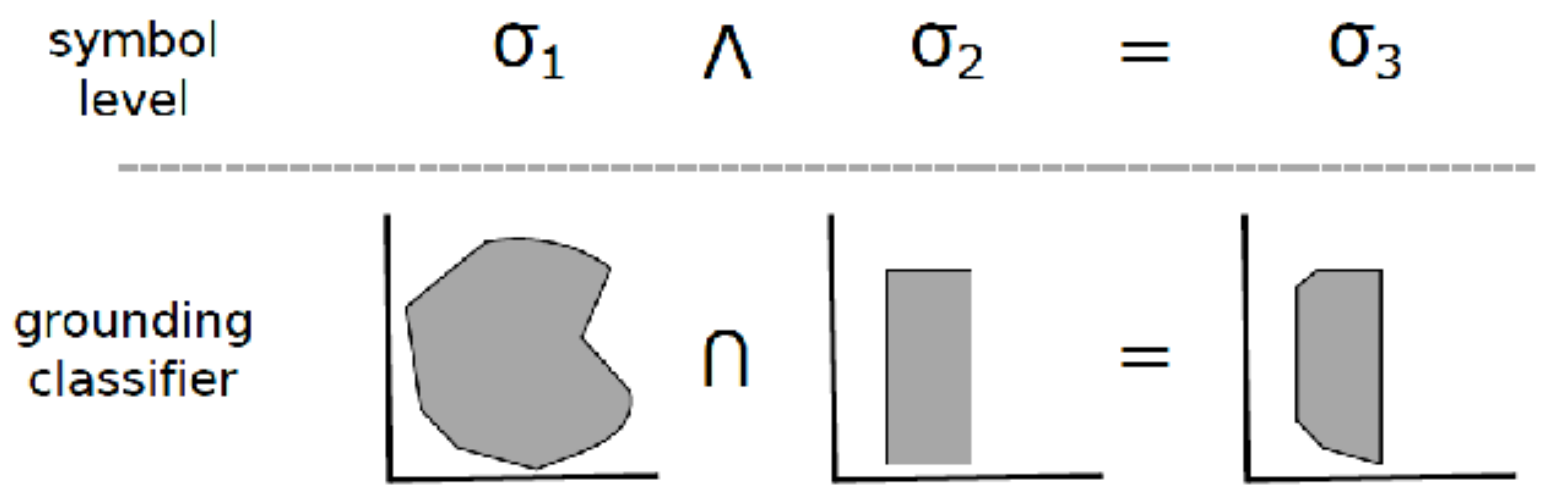
Probabilistic Planning

Must deal with *distributions over states* in the future.



Defining a Symbol

What do operations on our symbols mean?



(concrete boolean algebra)

Probabilistic Symbols

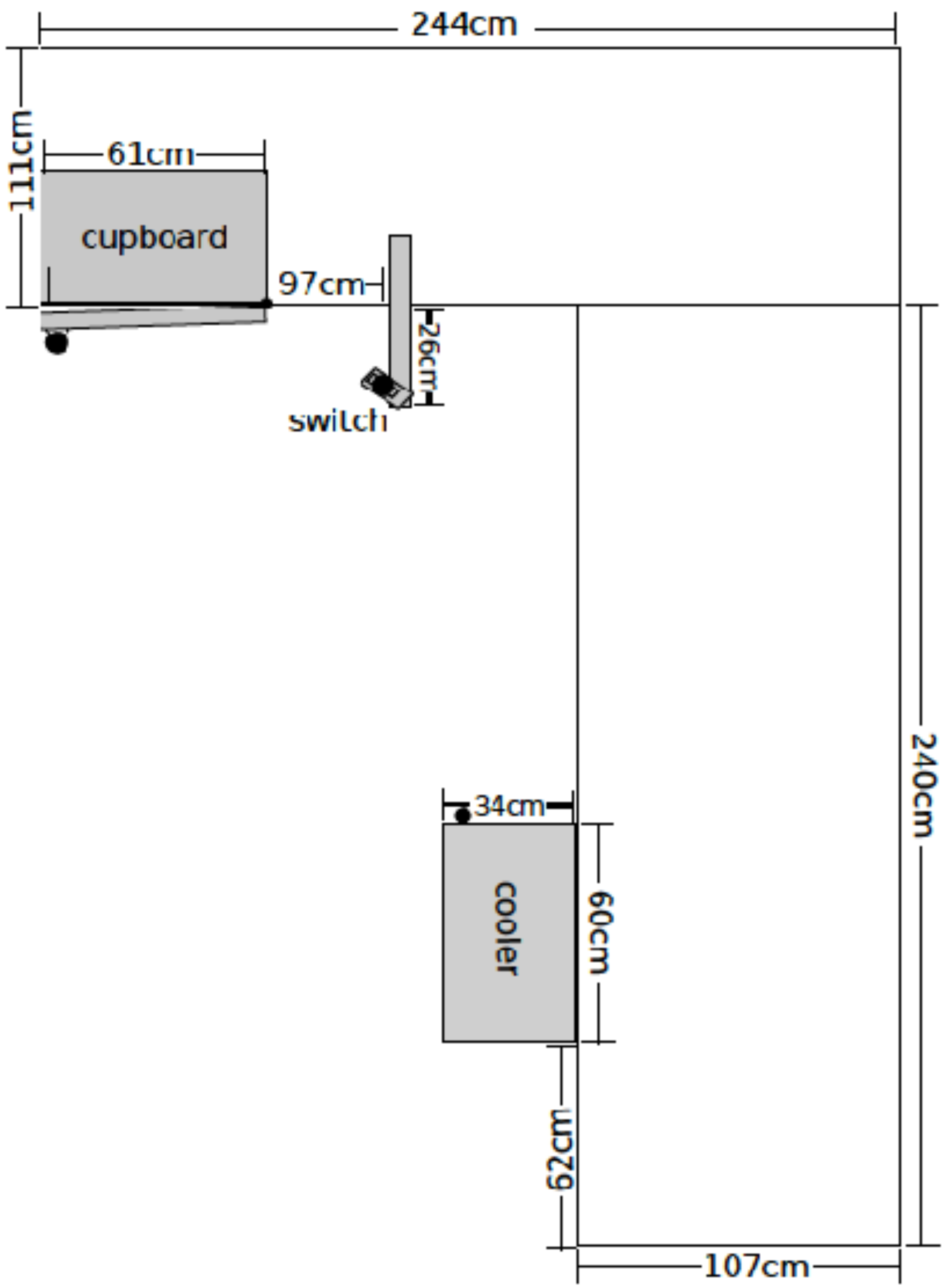
Learning symbolic representations

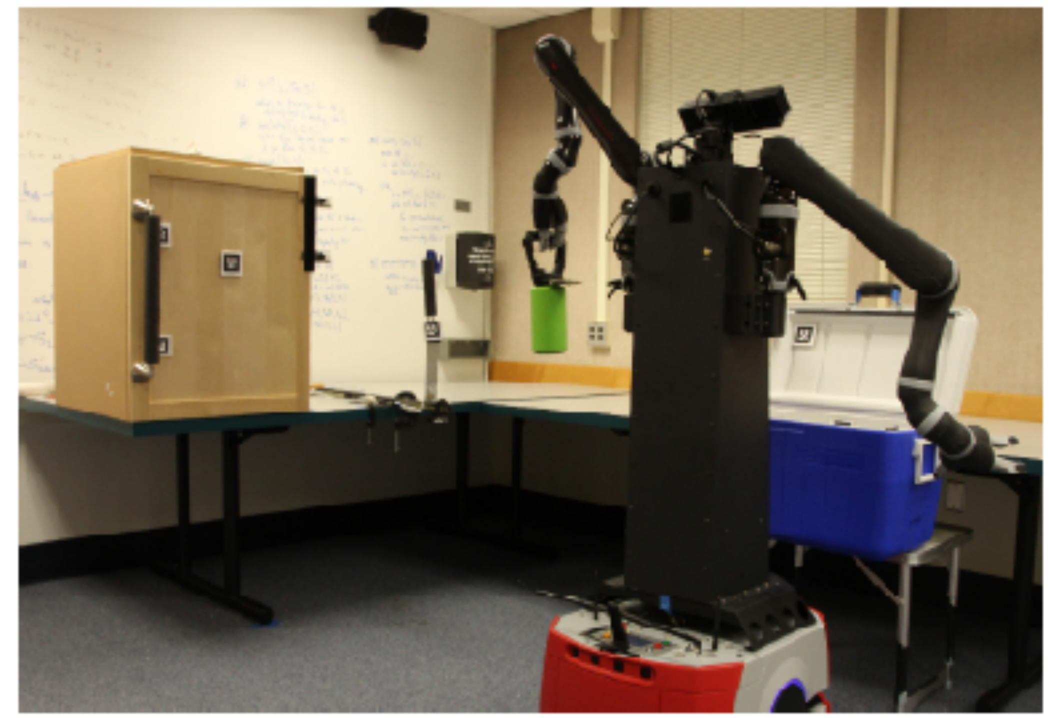
- Execute options and get some data

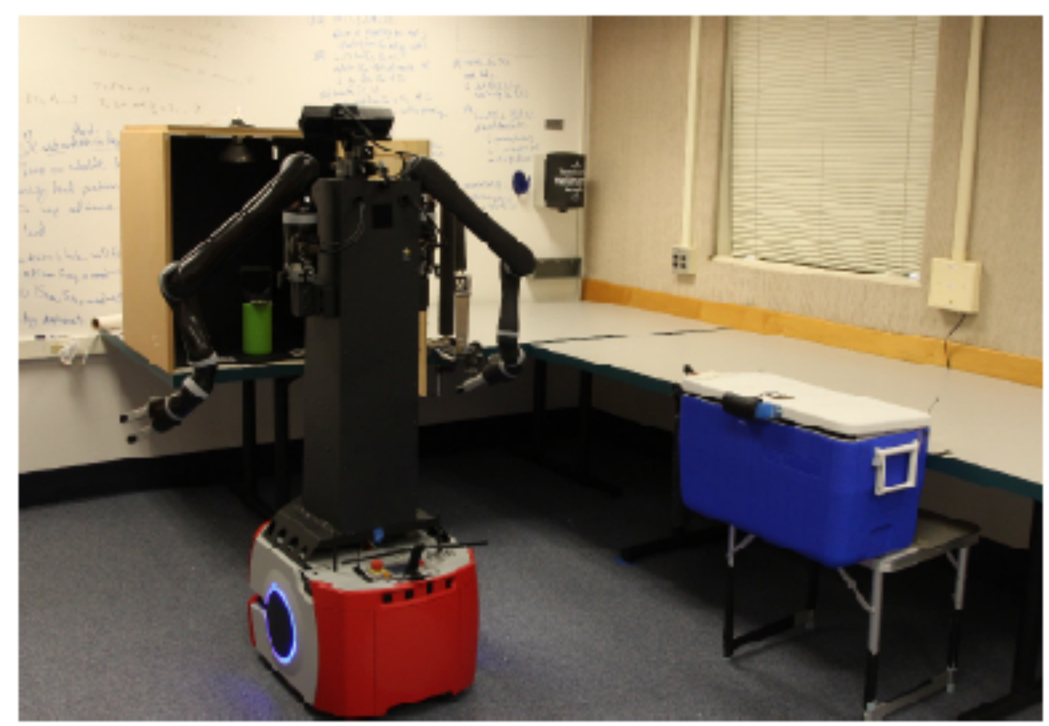
  (s, o, s', r) $(s, I_o?)$

- For each option:
  - Partition into ~abstract subgoal options
  - For each partitioned option:
    - Probabilistic classifier for init distribution
    - Density estimator for image distribution
    - Regression for reward model

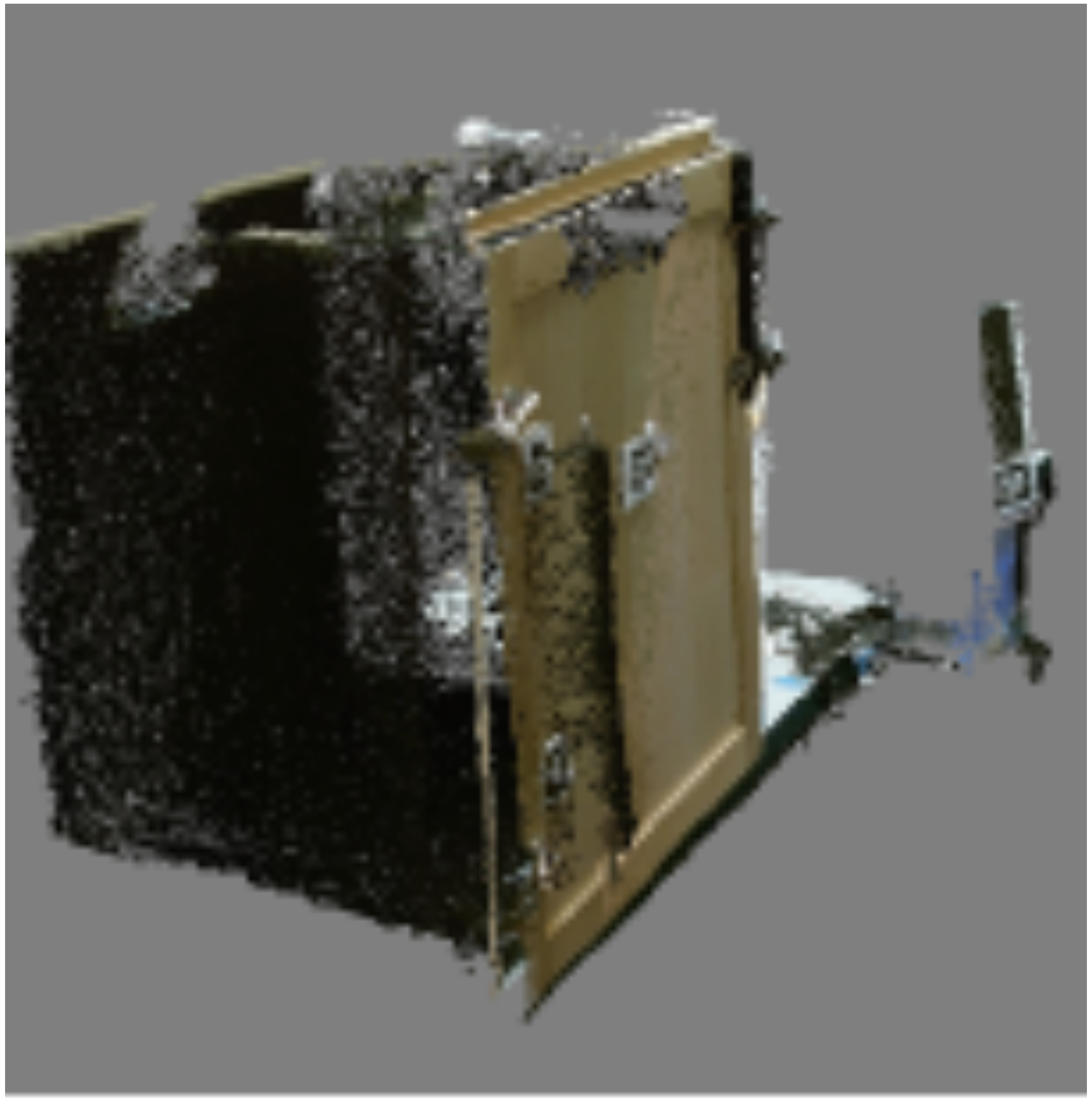
Learning Symbolic Representations



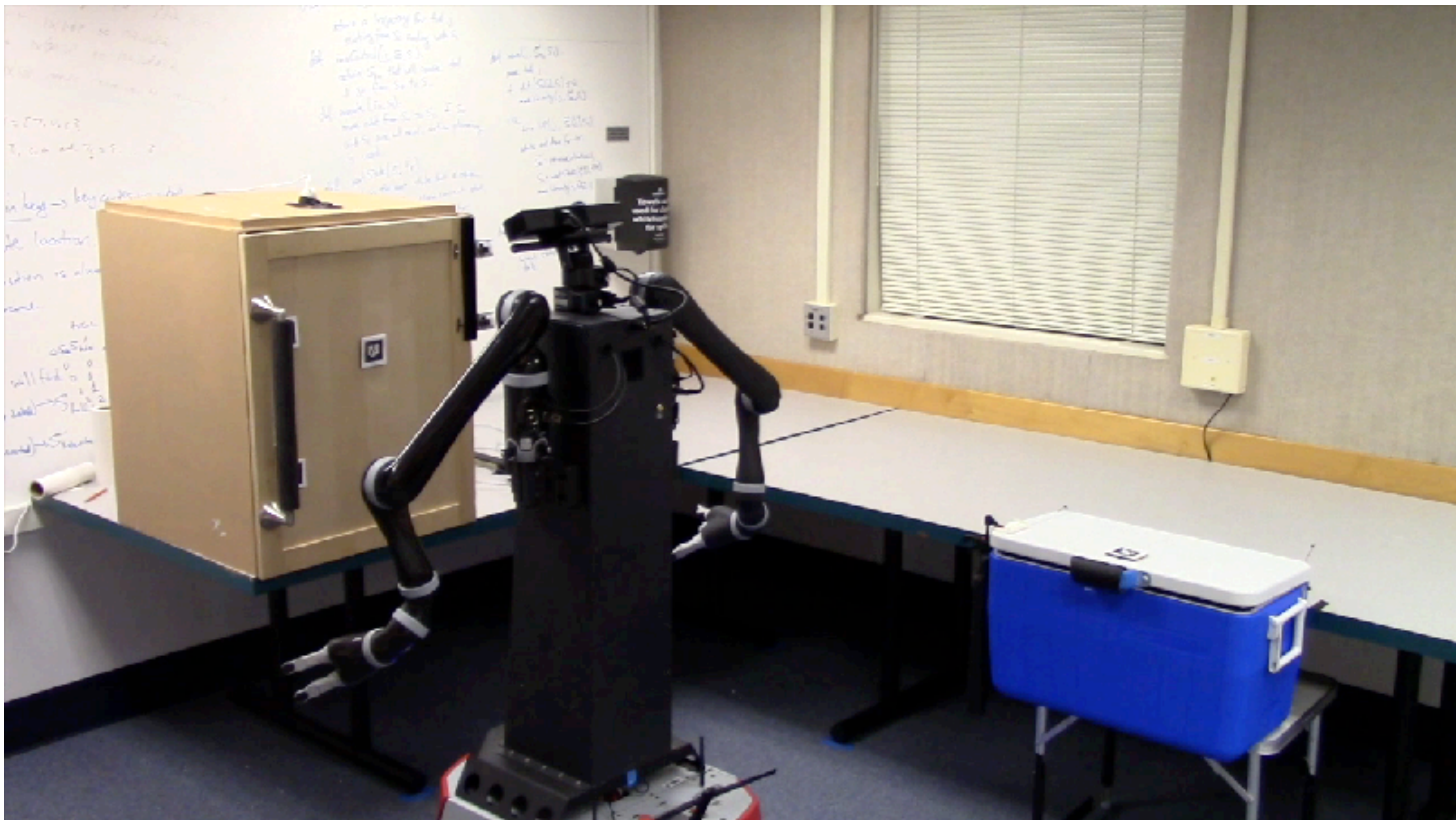




Symbolic Planning

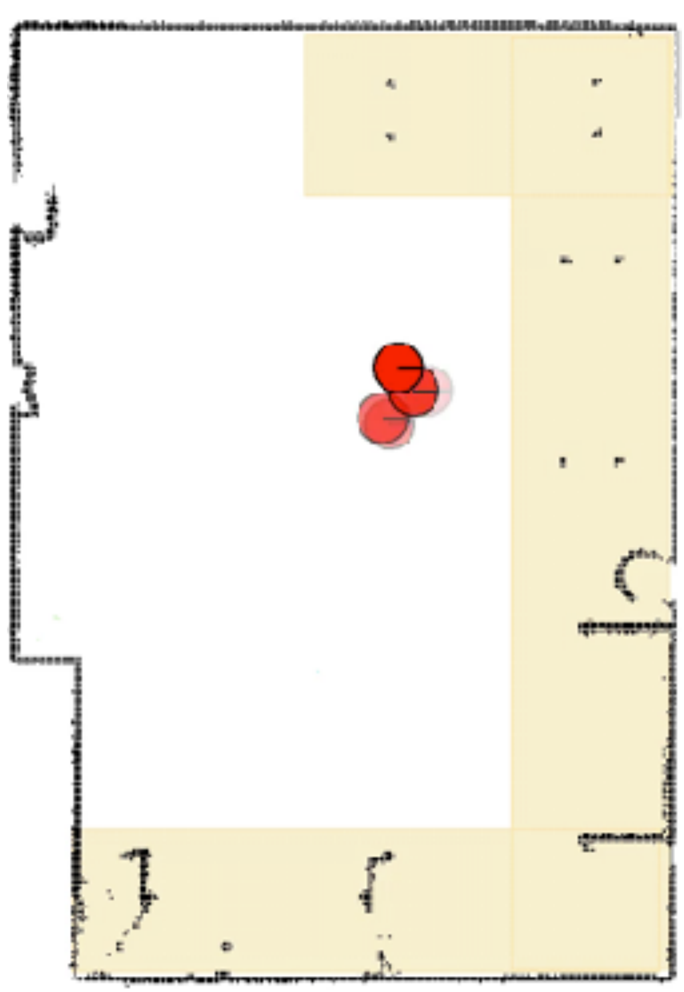


Learning Symbolic Representations

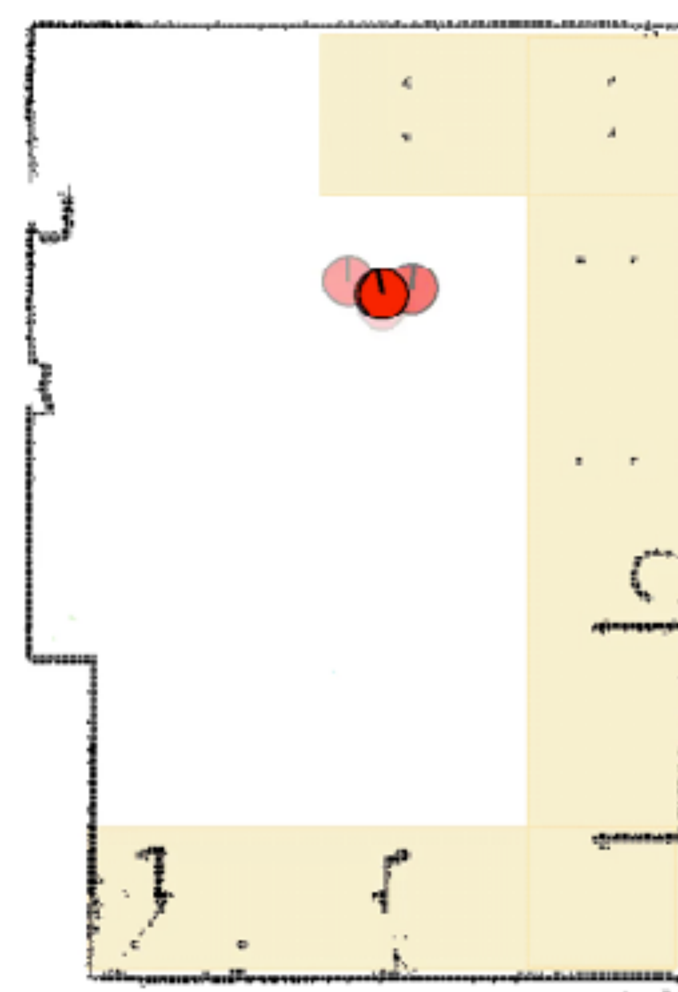


Symbolic Representations

```
(:action nav_to_cooler1
 :parameters ()
 :precondition (and (symbol1))
 :effect       (and (symbol0) (not (symbol1))
                    (decrease (reward) 37.25))
)
```



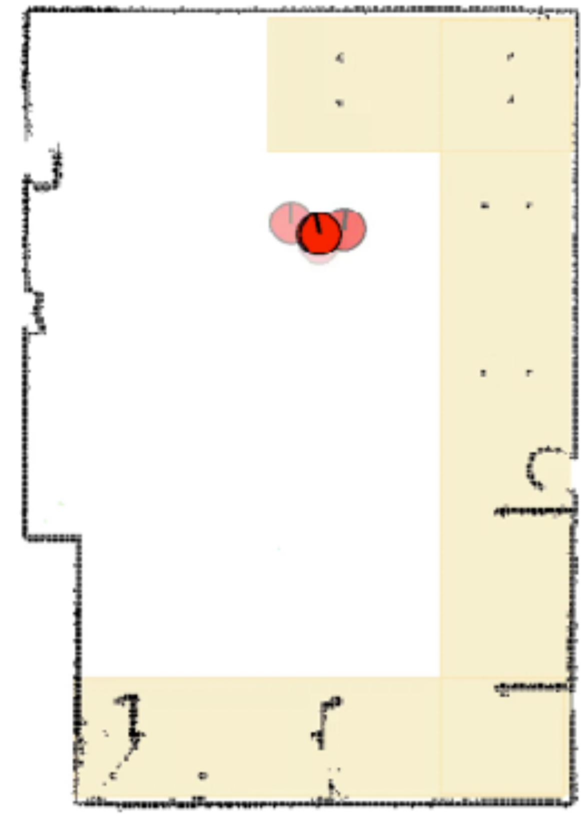
symbol0



symbol1

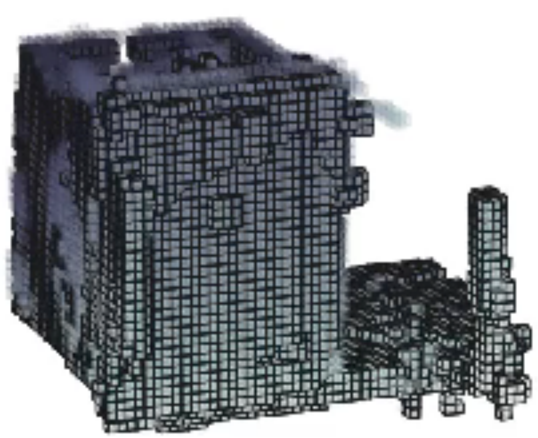
Symbolic Representations

```
(:action cupboard_open1
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect       (and (symbol5) (not (symbol4))
                    (decrease (reward) 67.44))
)
```

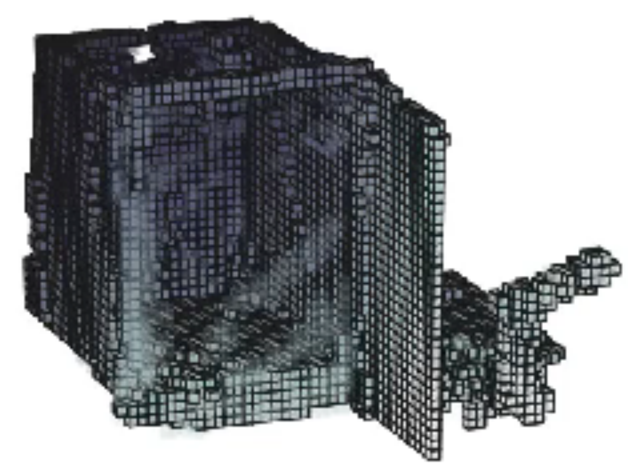


symbol1

symbol3



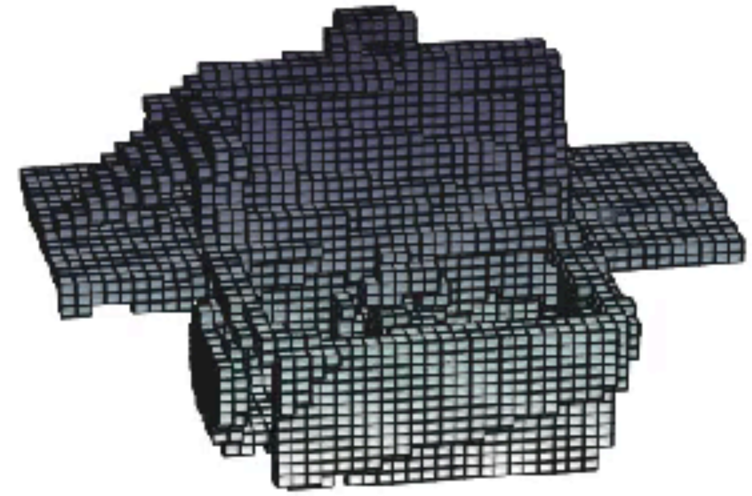
symbol4



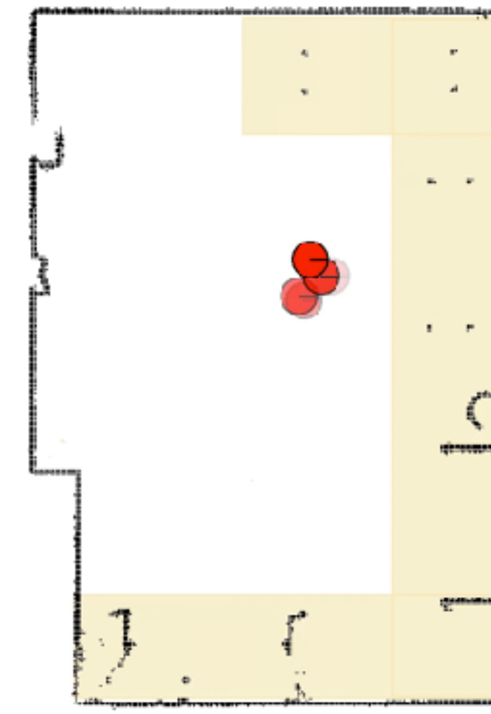
symbol5

Symbolic Representations

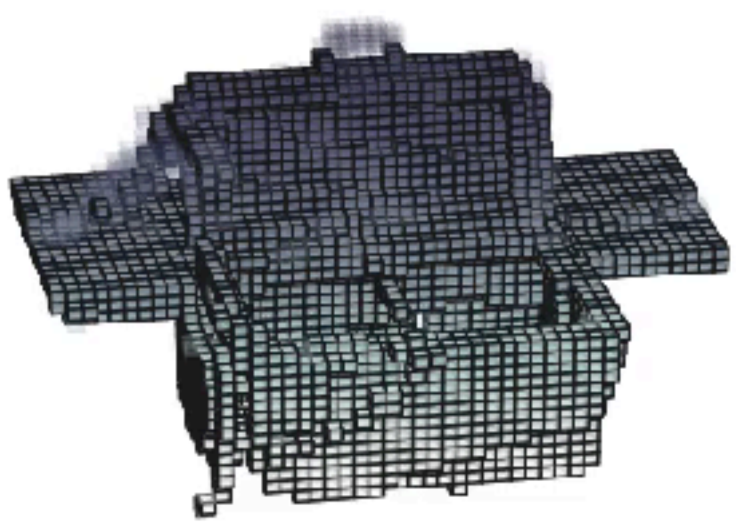
```
(:action pick_up1
 :parameters ()
 :precondition (and (symbol0) (symbol8)
                    (symbol12))
 :effect (and (symbol11) (symbol2)
              (not (symbol3)) (not (symbol12))
              (decrease (reward) 52.62))
)
```



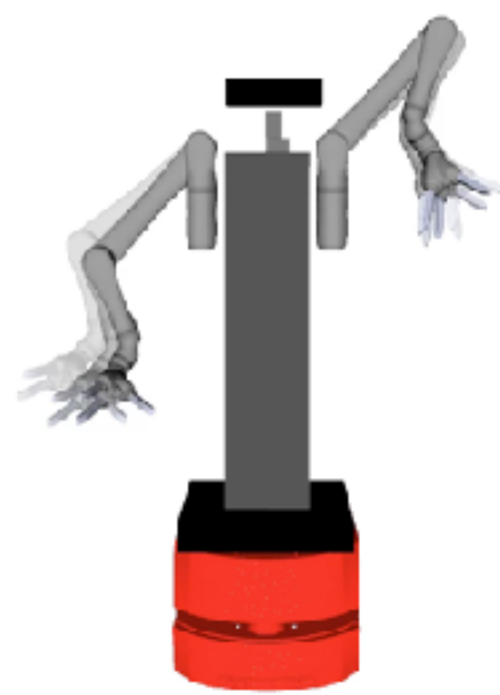
symbol8 and symbol12



symbol0



symbol8 and symbol11

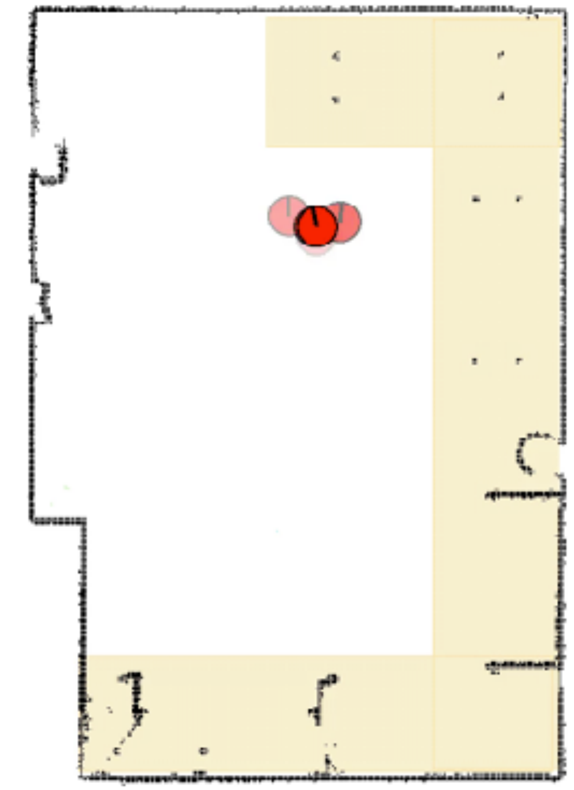


symbol2

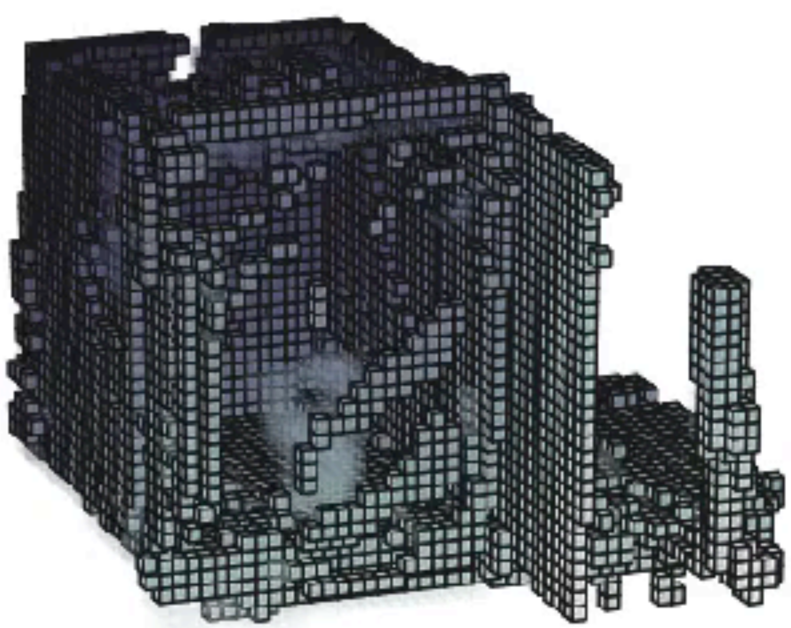
symbol3

Symbolic Representations

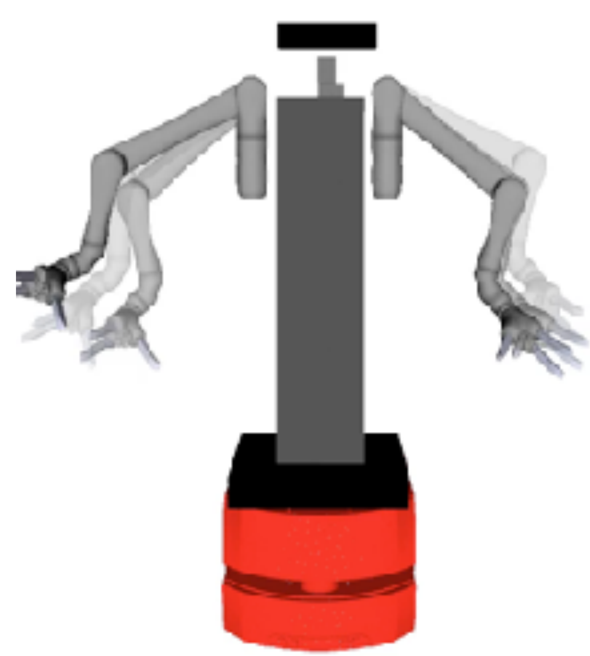
```
(:action pick_up2
 :parameters ()
 :precondition (and (symbol1) (symbol3)
                    (symbol5) (symbol6) (symbol11))
 :effect (probabilistic
    0.0559 (and)
    0.9441 (and (symbol2) (not (symbol3))
                (decrease (reward) 53.42))
        )
)
```



symbol1



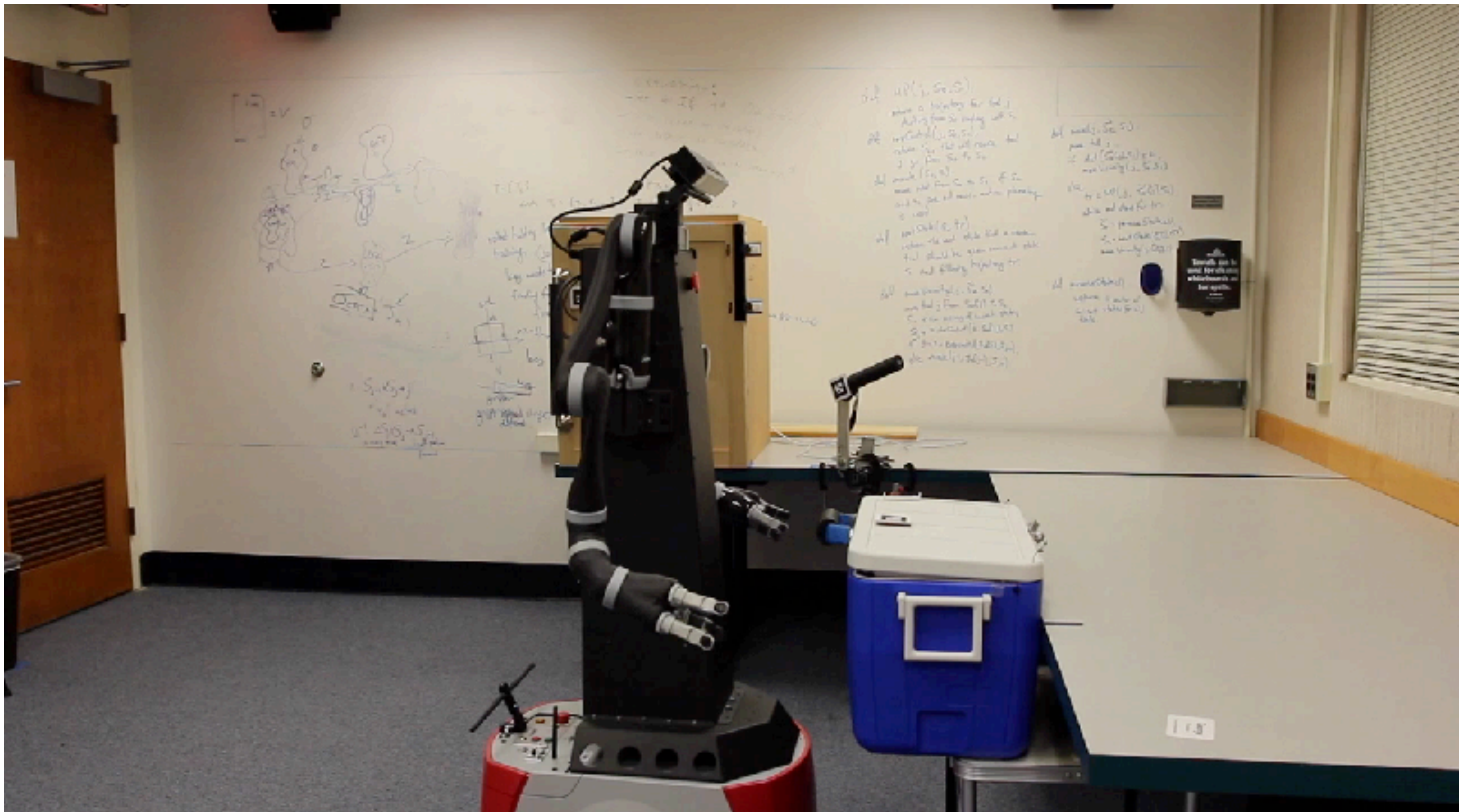
symbol5 and symbol6



symbol3

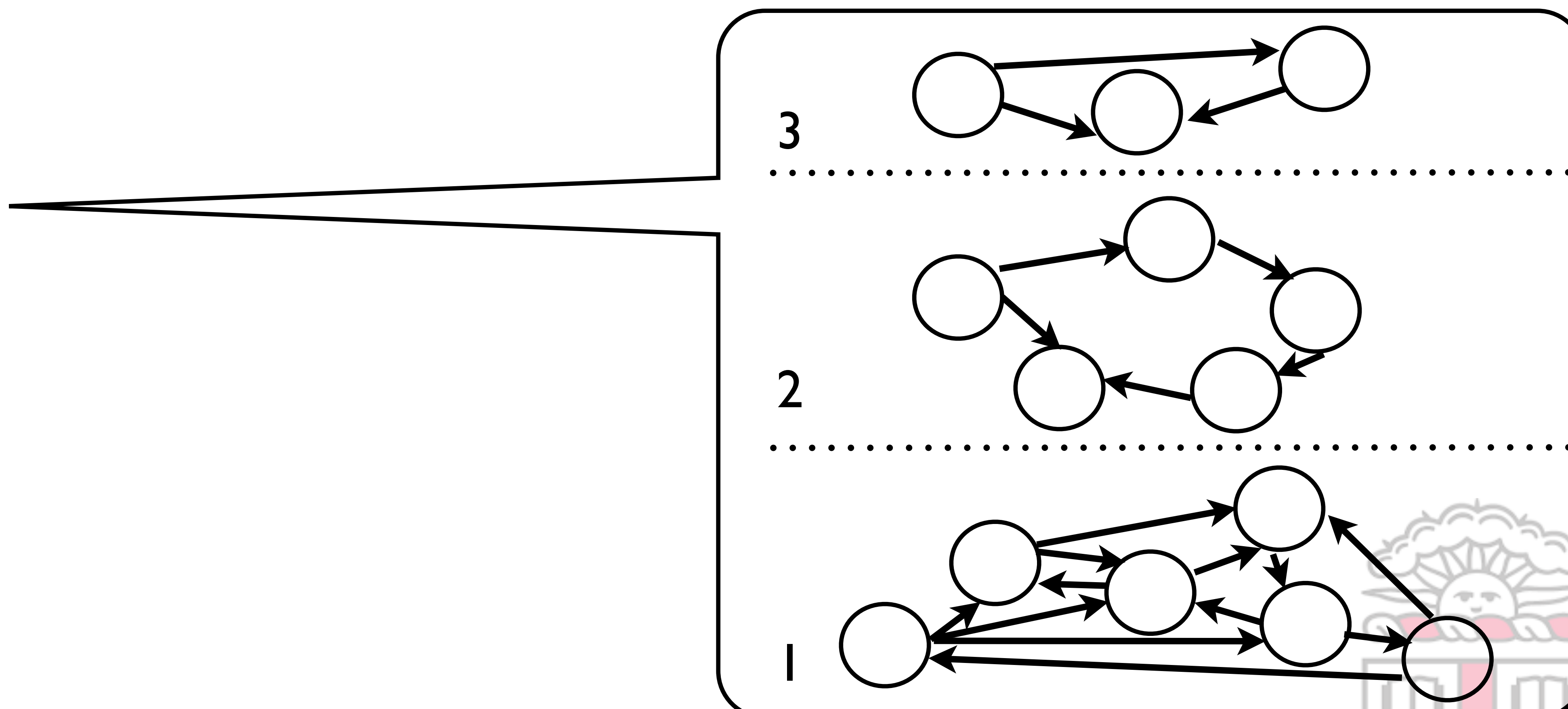
symbol2

Symbolic Planning



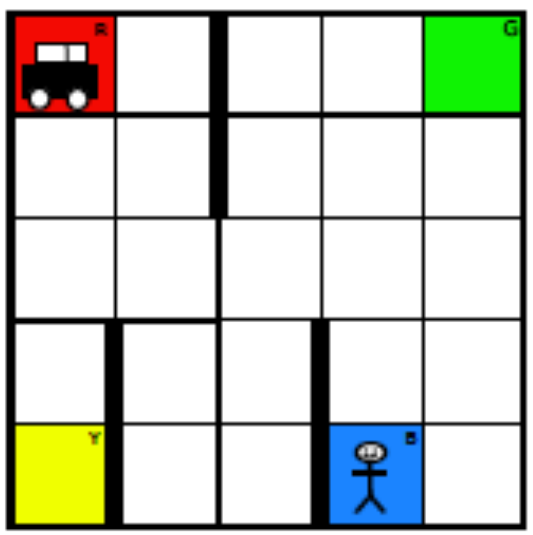
True Abstraction Hierarchies

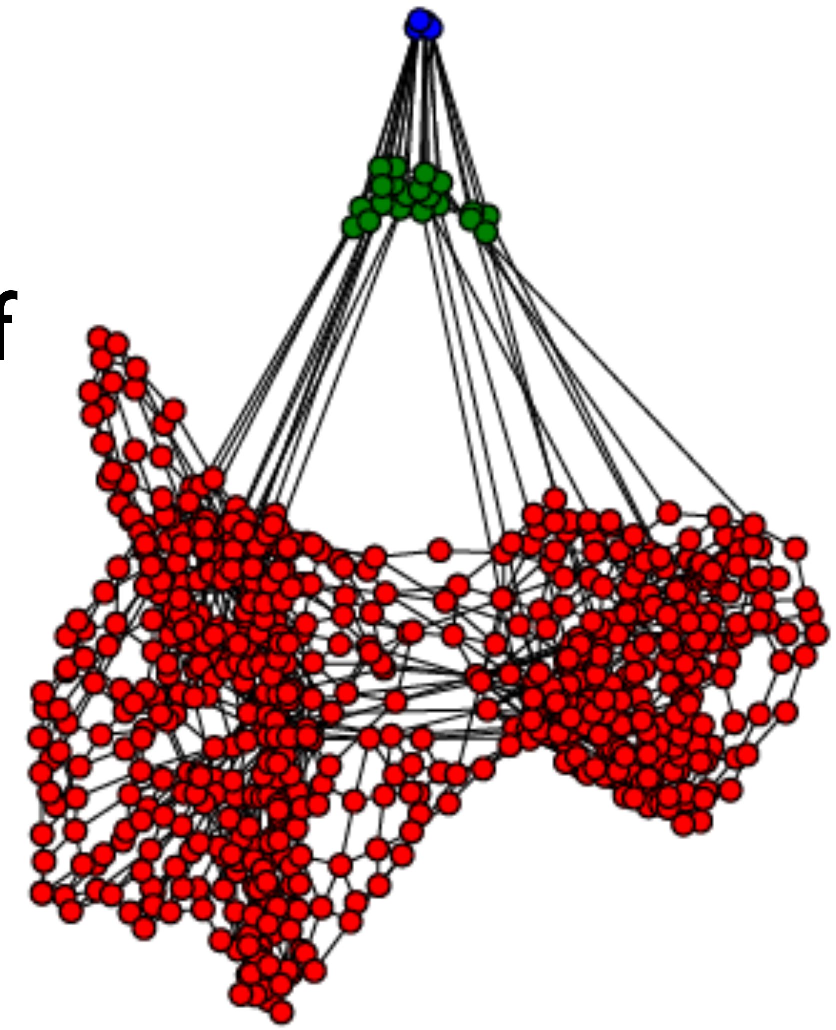
Base MDP: $M_0 = \{S_0, A_0, R_0, P_0\}$
Successive MDPs: $M_i = \{S_i, A_i, R_i, P_i\}$



Taxi

Options:
1. up, down, left, right, pick up, drop off
2. drive to each depot, pick up, drop off
3. passenger-to-depot

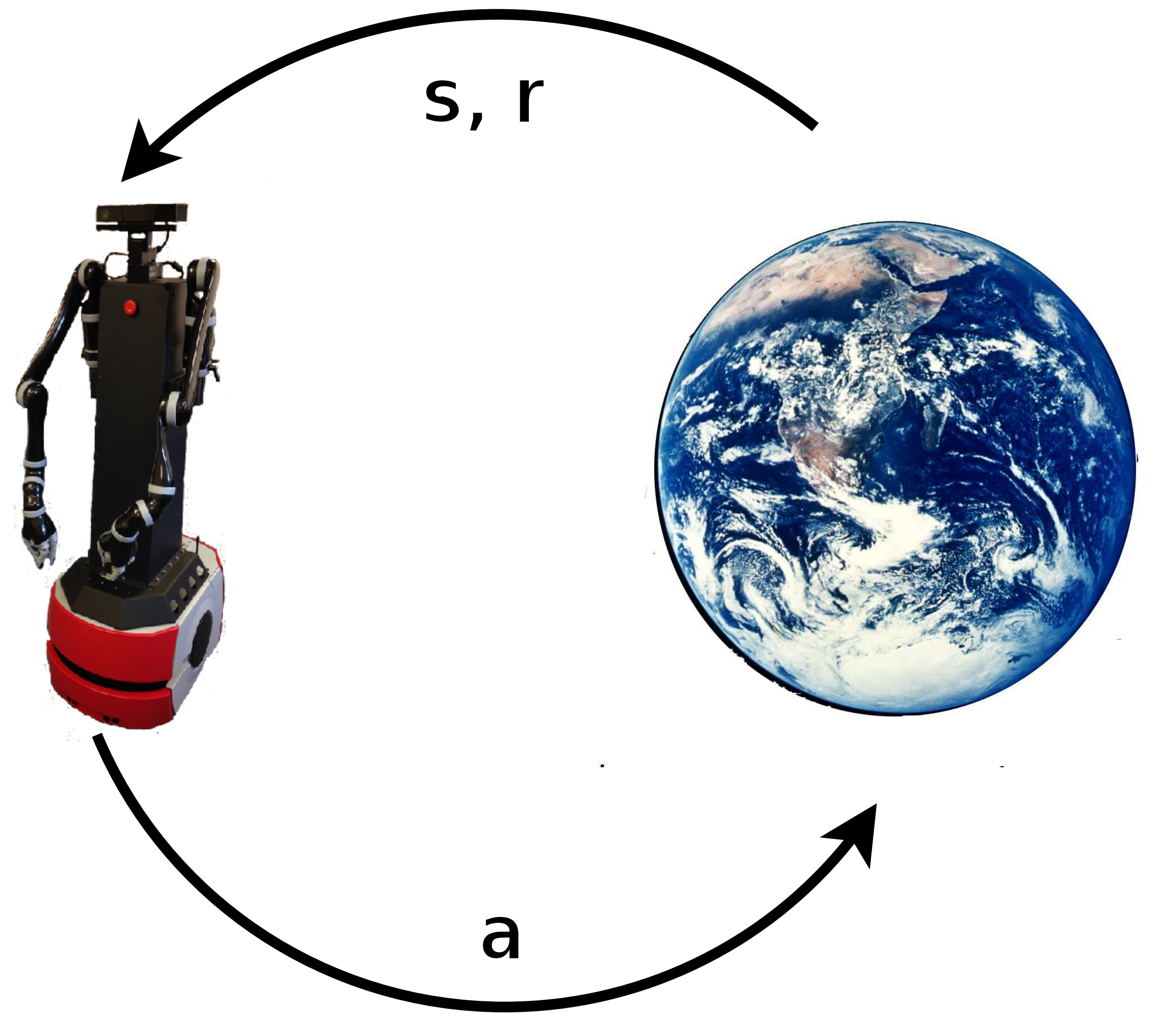




[Konidaris, IJCAI 2016]

		Hierarchical Planning				
Query	Level	Matching	Planning	Total	Base + Options	Base MDP
1	2	<1	<1	**<1**	770.42	1423.36
2	1	<1	10.55	**11.1**	1010.85	1767.45
3	0	12.36	1330.38	1342.74	**1174.35**	**1314.94**

Reinforcement Learning



Thank you!

Questions?