Reinforcement Learning

Reinforcement Learning

- Assumptions we made so far:
  - Known state space S
  - Known transition model T(s, a, s’)
  - Known reward function R(s)
  - ➔not realistic for many real agents

Reinforcement Learning:

- Learn optimal policy with a priori unknown environment
- Assume fully observable state(i.e. agent can tell its state)
- Agent needs to explore environment (i.e. experimentation)

Passive Reinforcement Learning

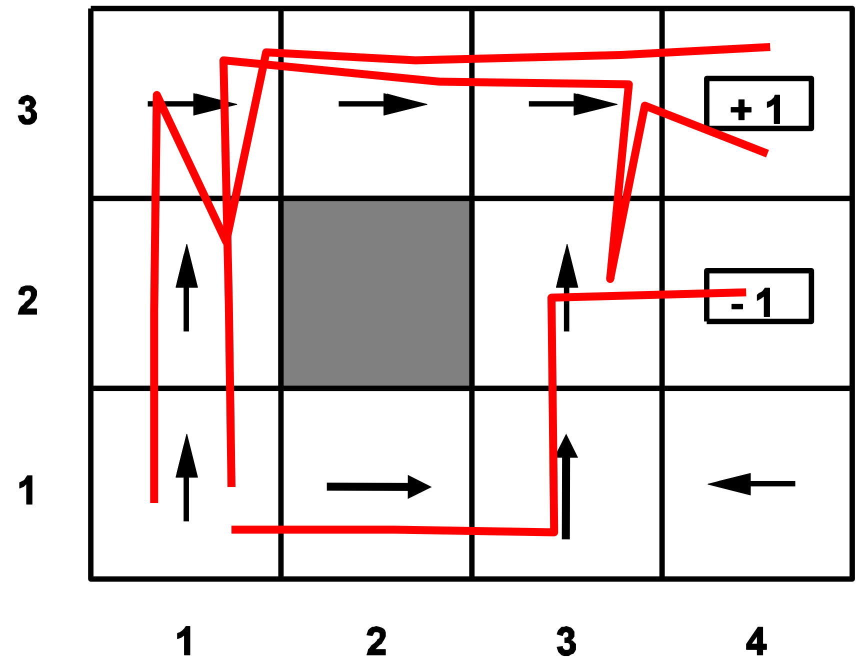
- Task: Given a policy π, what is the utility function U^{π} ?
  - Similar to Policy Evaluation, but unknown T(s, a, s’) and R(s)

Approach: Agent experiments in the environment
  - Trials: execute policy from start state until in terminal state.

$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04}$
$\rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04}$
$\rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04}$
$\rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$

$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04}$
$\rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04}$
$\rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04}$
$\rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$

$(1,1)_{-0.04} \rightarrow (2,1)_{-0.04}$
$\rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04}$
$\rightarrow (4,2)_{-1.0}$



Direct Utility Estimation

- Data: Trials of the form
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1.0}$
- Idea:
  - Average reward over all trials for each state independently
- **From data above, estimate U(1,1)**
  - **A=0.72 B= -1.16 C=0.28 D=0.55**

Direct Utility Estimation

- Data: Trials of the form
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1.0}$
- Idea:
  - Average reward over all trials for each state independently

– **From data above, estimate U(1,2)**

  – **A=0.76 B= 0.77 C=0.78 D=0.79**

Direct Utility Estimation

- Why is this less efficient than necessary?

  ➔ Ignores dependencies between states

  $U^{\pi}(s) = R(s) + \gamma \Sigma_{s'} T(s, \pi(s), s') U^{\pi}(s')$

•Adaptive Dynamic Programming (ADP)

- **Idea:**
  - Run trials to learn model of environment (i.e. T and R)
    - Memorize R(s) for all visited states
    - Estimate fraction of times action a from state s leads to s'
  - Use PolicyEvaluation Algorithm on estimated model
- **Data: Trials of the form**
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1.0}$

ADP

- $(1,1)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(2,3)_{-0.04}$ → $(3,3)_{-0.04}$ → $(4,3)_{1.0}$
- $(1,1)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(2,3)_{-0.04}$ → $(3,3)_{-0.04}$ → $(3,2)_{-0.04}$ → $(3,3)_{-0.04}$ → $(4,3)_{1.0}$
- $(1,1)_{-0.04}$ → $(2,1)_{-0.04}$ → $(3,1)_{-0.04}$ → $(3,2)_{-0.04}$ → $(4,2)_{-1.0}$

Estimate T[(1,3), *right*, (2,3)]

A=0 B=0.333 C=0.666 D=1.0

- **Problem?**
  - Can be quite costly for large state spaces
  - For example, Backgammon has 10^{50} states

➔ Learn and store all transition probabilities and rewards

➔ PolicyEvaluation needs to solve linear program with 10^{50} equations and variables.

Temporal Difference (TD) Learning

- If policy led U(1,3) to U(2,3) all the time, we would expect that
  - $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$
- R(s) should be equal $U^{\pi}(s) - \gamma U^{\pi}(s')$, so
- $U^{\pi}(s) = U^{\pi}(s) + \alpha [R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)]$
  - – α is learning rate. α should decrease slowly over time, so that estimates stabilize eventually.

From observation,

U(1,3)=0.84 → U(2,3)=0.92

And R = -0.04

Is U(1,3) too low or too high?

A=Too Low B=Too high

Temporal Difference (TD) Learning

- Idea:
  - Do not learn explicit model of environment!
  - Use update rule that implicitly reflects transition probabilities.
- Method:
  - Init $U^\pi(s)$ with $R(s)$ when first visited
  - After each transition, update with
    $$U^\pi(s) = U^\pi(s) + \alpha \, [R(s) + \gamma \, U^\pi(s') - U^\pi(s)]$$
  - α is learning rate. α should decrease slowly over time, so that estimates stabilize eventually.
- Properties:
  - No need to store model
  - Only one update for each action (not full PolicyEvaluation)

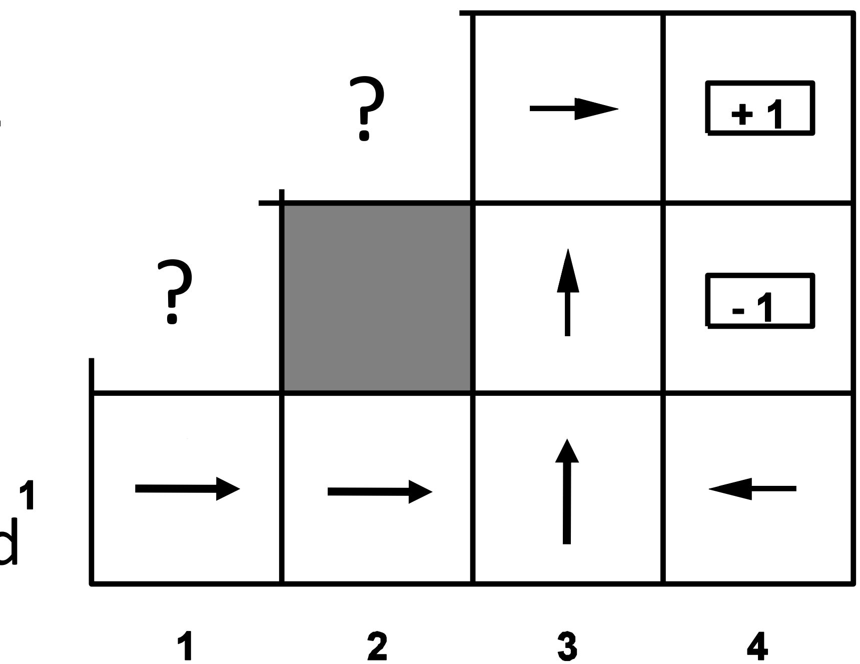
Active Reinforcement Learning

- Task: In an a priori unknown environment, find the optimal policy.
  - unknown T(s, a, s') and R(s)
  - Agent must experiment with the environment.
- Naïve Approach: "Naïve Active PolicyIteration"
  - Start with some random policy
  - ~~Follow policy~~ to learn model of environment and use ADP to estimate utilities.
  - Update policy using $\pi(s) \leftarrow \text{argmax}_a \, \Sigma_{s'} \, T(s, a, s') \, U^{\pi}(s')$
- Problem:
  - Can converge to sub-optimal policy!
  - By following policy, agent might never learn T and R everywhere.

➔ Need for exploration!

Exploration vs. Exploitation

- Exploration:
  - Take actions that explore the environment
  - Hope: possibly find areas in the state space of higher reward
  - Problem: possibly take suboptimal steps
- Exploitation:
  - Follow current policy
  - Guaranteed to get certair expected reward
- Approach:
  - Sometimes take rand steps
  - Bonus reward for states that have not been visited often yet



Q-Learning

- Problem: Agent needs model of environment to select action via

  $\text{argmax}_a \sum_{s'} T(s, a, s') U^{\pi}(s')$

- Solution: Learn action utility function Q(a,s), not state utility function U(s). Define Q(a,s) as

  $U(s) = \max_a Q(a,s)$

  ➔ Bellman equation with Q(a,s) instead of U(s)

  $Q(a,s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a',s')$

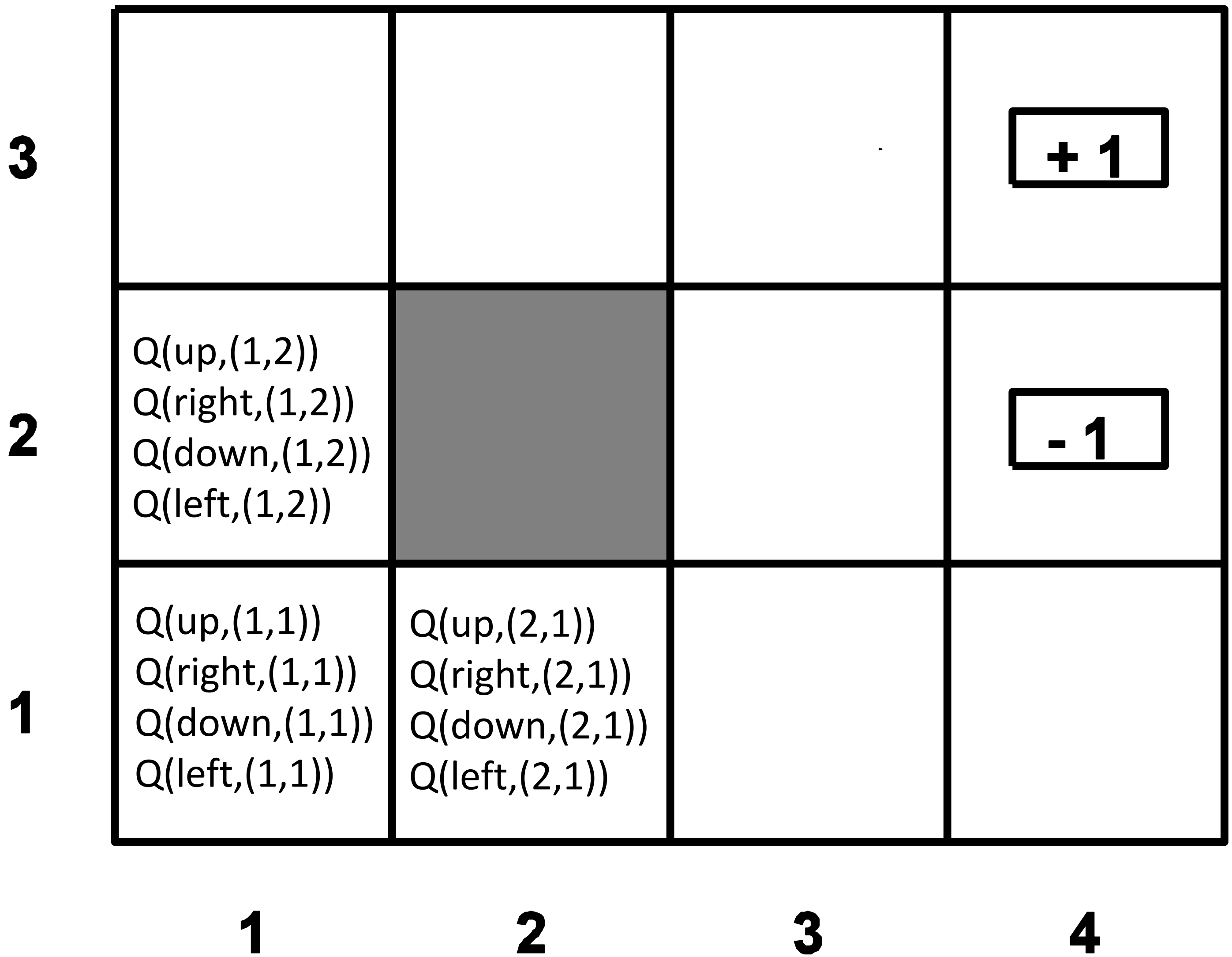
  ➔ TD-Update with Q(a,s) instead of U(s)

  $Q(a,s) \leftarrow Q(a,s) + \alpha \, [R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)]$

- Result: With Q-function, agent can select action without model of environment

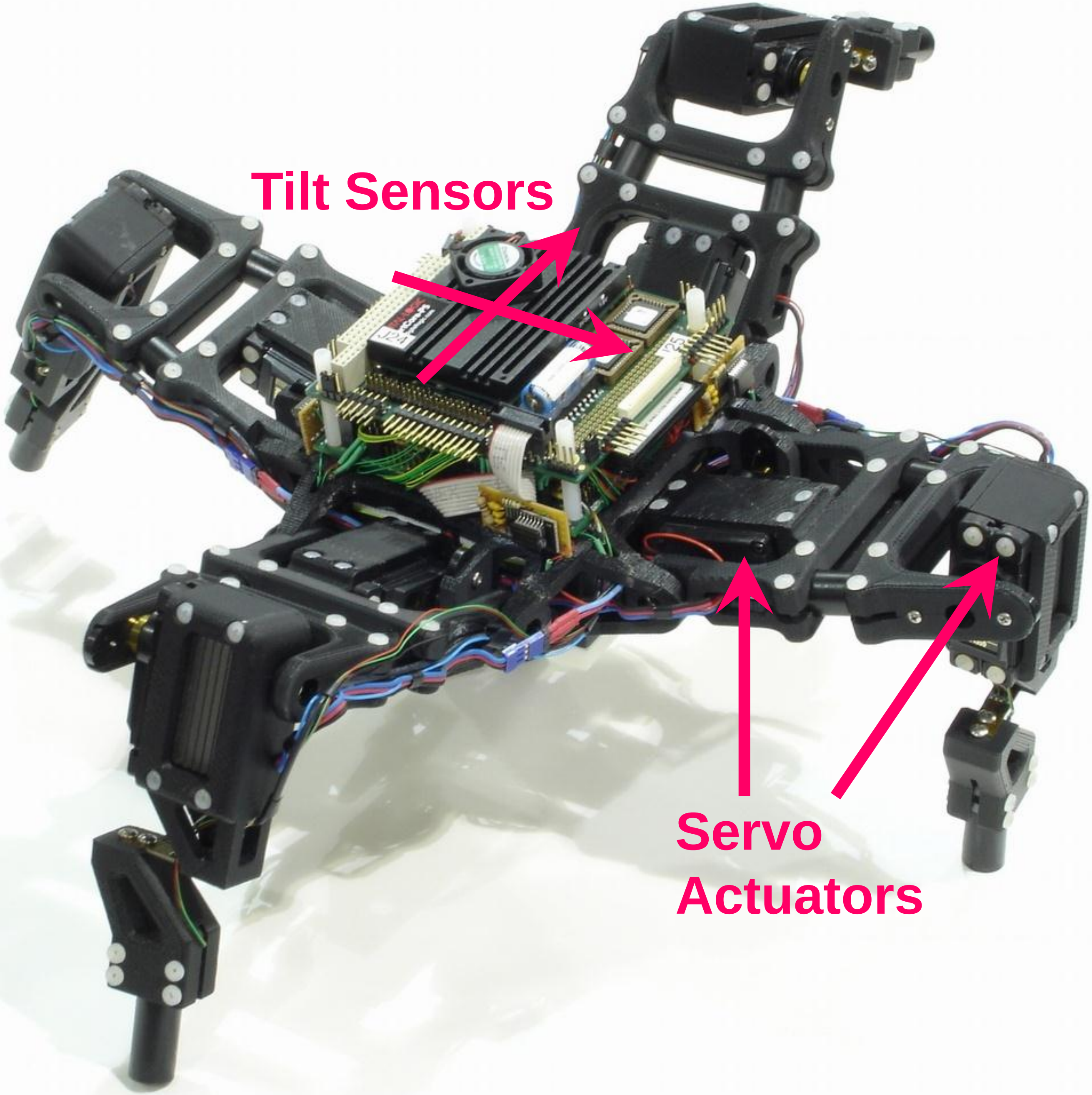
  $\text{argmax}_a Q(a,s)$

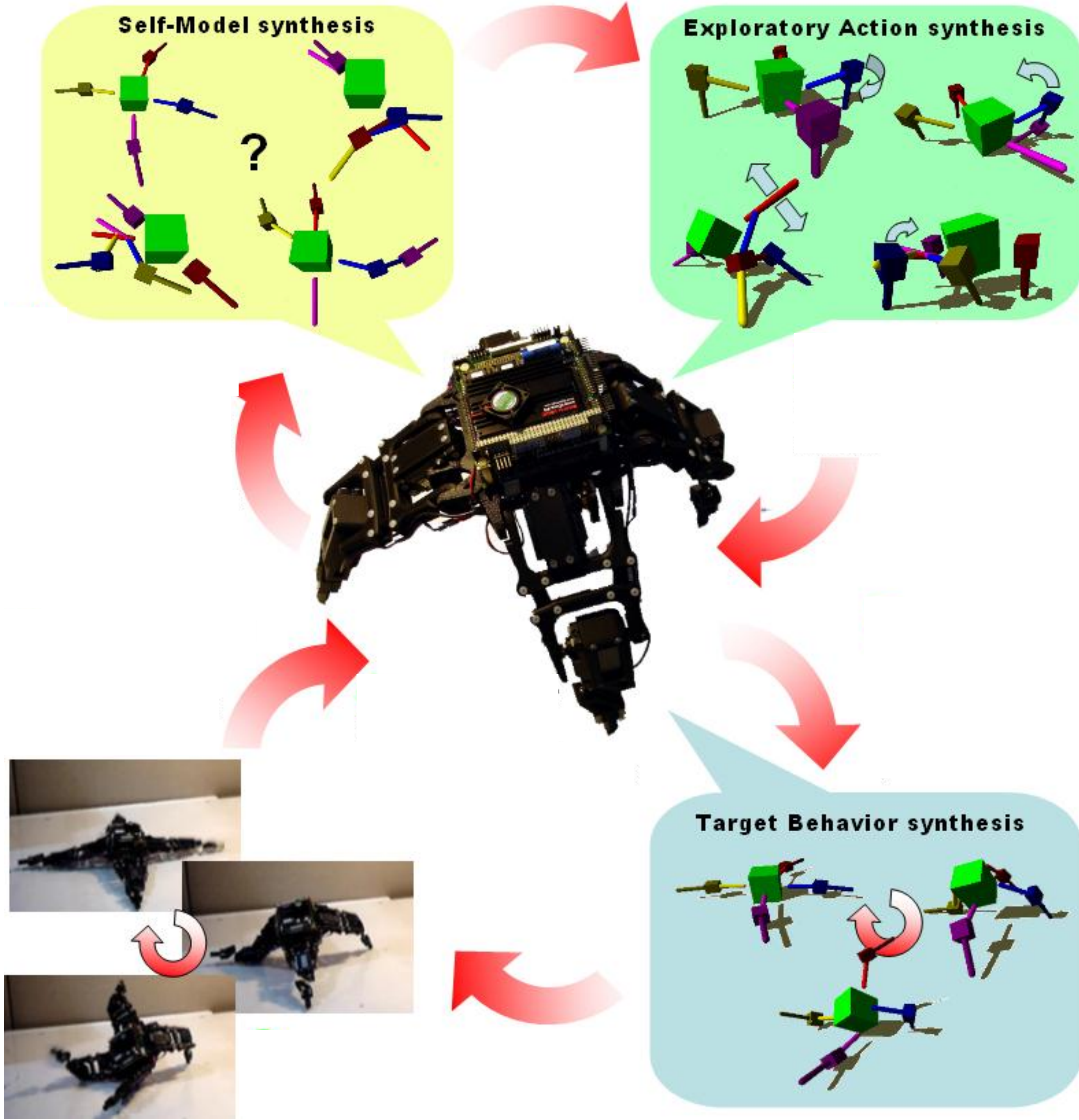
Q-Learning Illustration



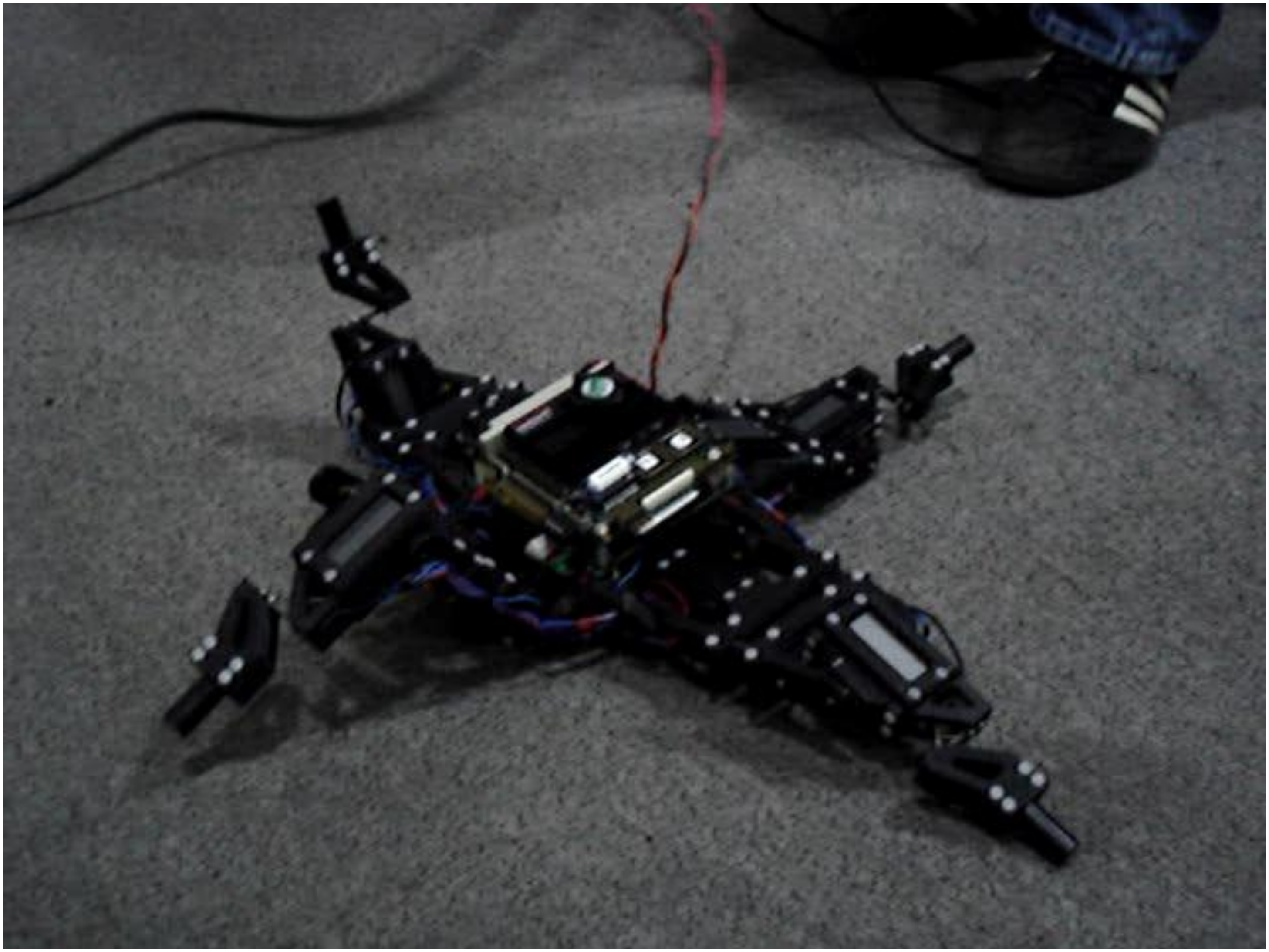
Function Approximation

- Problem:
  - Storing Q or U,T,R for each state in a table is too expensive, if number of states is large
  - Does not exploit "similarity" of states (i.e. agent has to learn separate behavior for each state, even if states are similar)
- Solution:
  - Approximate function using parametric representation $U(s) = \vec{w} \cdot \Phi(s)$
  - For example:
    - Φ(s) is feature vector describing the state
      - "Material values" of board
      - Is the queen threatened?
      - …

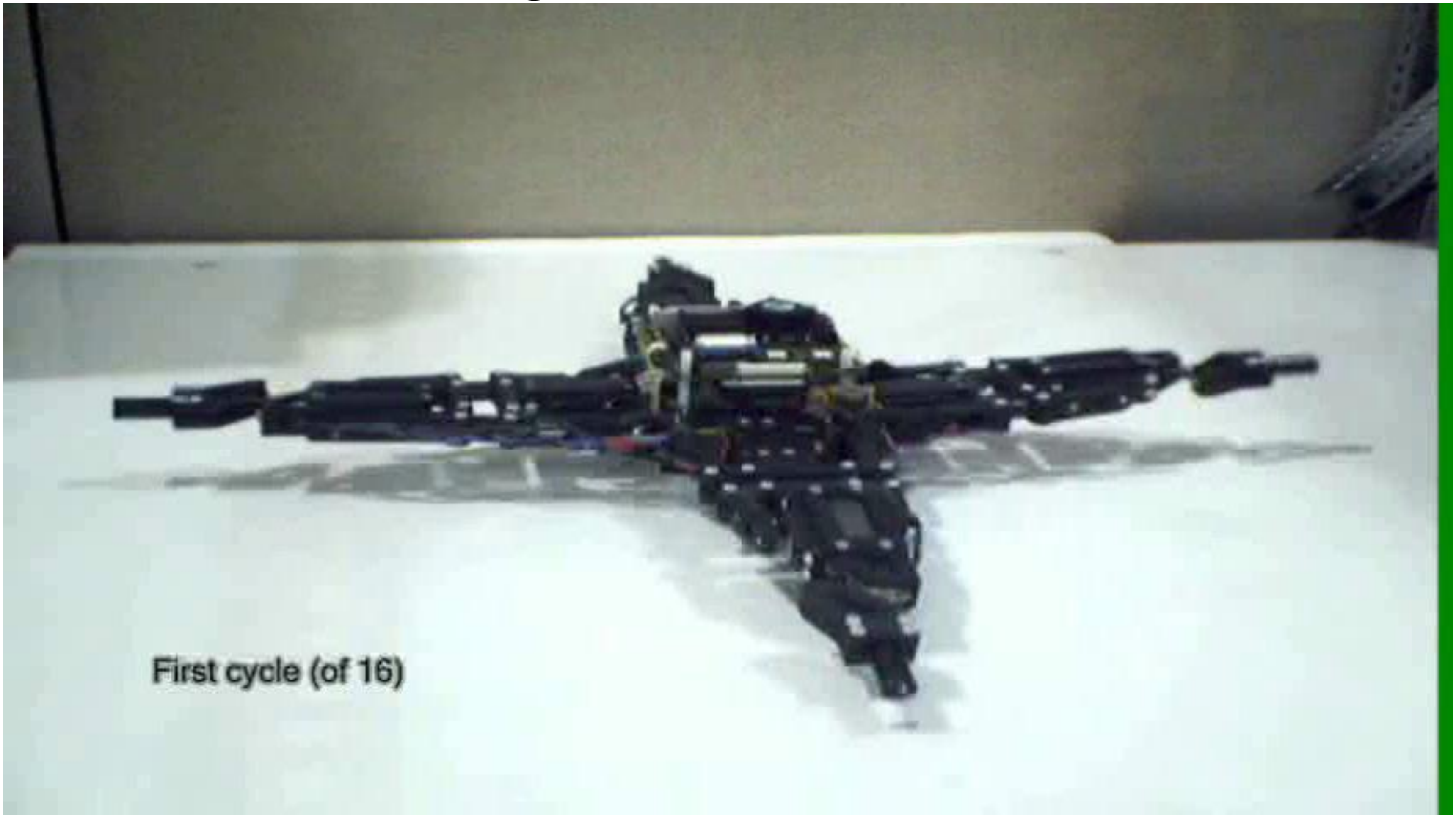

Tilt Sensors
Servo
Actuators


Self-Model synthesis
?
Exploratory Action synthesis
Target Behavior synthesis

Morphological Estimation



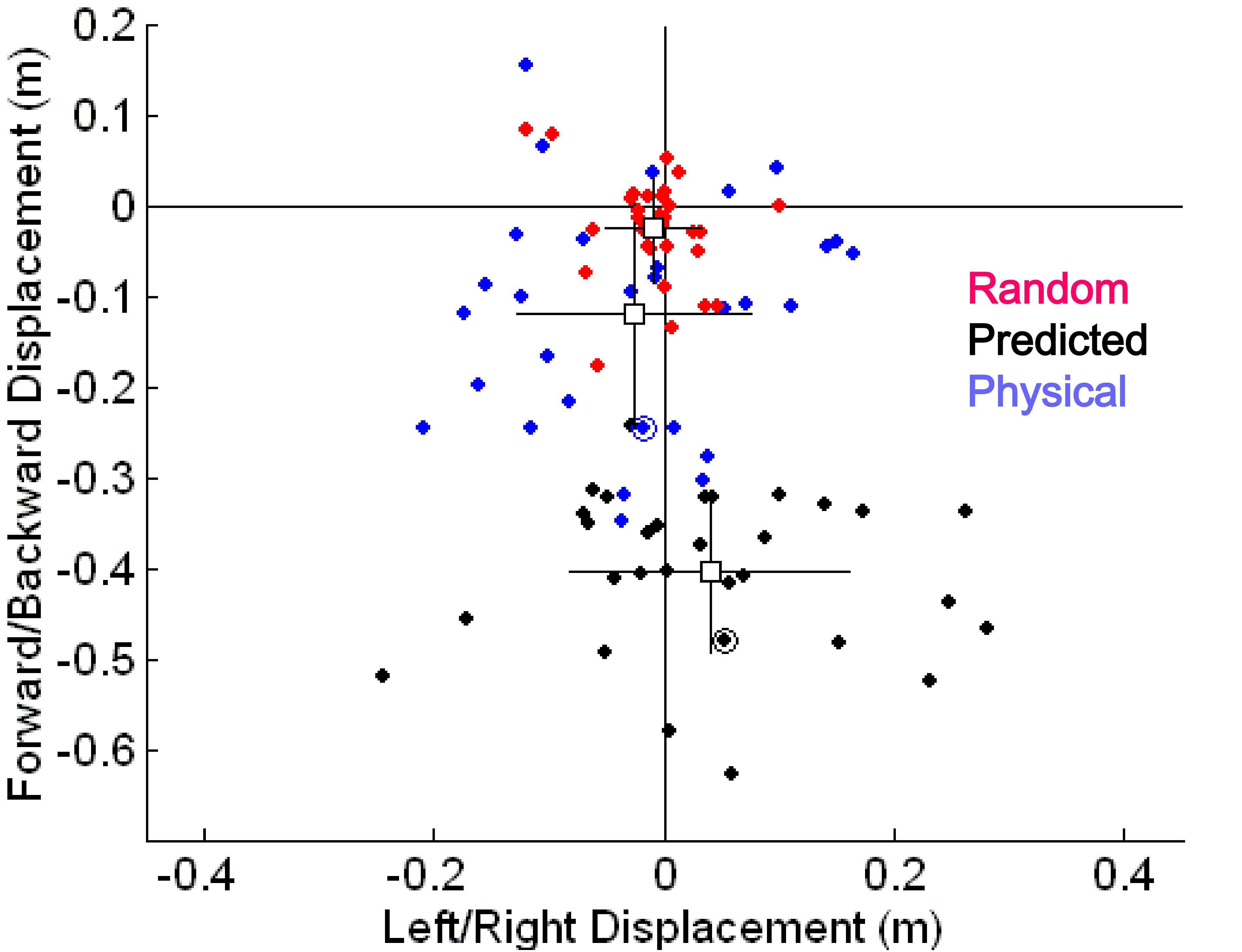
Emergent Self-Model



With Josh Bongard and Victor Zykov, Science 2006

Damage Recovery

With Josh Bongard and Victor Zykov, Science 2006



Random
Predicted
Physical
Forward/Backward Displacement (m)
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.4
-0.2
0
0.2
0.4
Left/Right Displacement (m)