CS 545 Lecture 19 – Optimal Control 2

- Partially Observable Markov Decision Processes
- Solution Techniques
- Hidden Markov Models
- Baum-Welch
- Reinforcement Learning

Additional Readings

- *Probabilistic Robotics* – Thrun, Burgard, Fox
- *Reinforcement Learning: A Survey* – Kaelbling, Littman, Moore (1995)
- *Planning and Acting in Partially Observable Stochastic Domains* – Kaelbling, Littman, Cassandra (1998)
- *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition* – Rabiner (1989)

Partial Observability

- MDPs – Assume actions are stochastic but state is *fully observable*. ie. Robot always knows current location
- *Partial Observability* – state is not directly observable
  - Sensors may be noisy or provide incomplete information
  - Must be inferred from *observations*
- Combine tracking with decision-making

Partial Observability

- Current uncertainty vs. anticipated uncertainty

- POMDPs will take *information collecting actions*

Partially Observable Markov Decision Processes (POMDPs)

- $S = \{s_0, s_1, \dots, s_n\}$ – set of states
- $A = \{a_0, a_1, \dots, a_m\}$ – set of **stochastic** actions
- *T(s,a,s')* – transition function *P(s'|s,a)*

$$P(s_{t+1} = s' | s_t = s, a_t = a)$$

- $R(s,a): S \times A \to \mathbb{R}$ – reward function
- Υ – discount factor
- $Z = \{z_0, z_1, \dots, z_q\}$ - set of observations
- *O(s,a,z)* – observation function

$$P(z_t = z | s_t = s, a_t = a)$$

Belief State Tracking

- *History* maintains trace of agents actions over time

$$h_t = a_0, z_1, a_1, z_2, \dots, a_{t-1}, z_t$$

- Not practicable to maintain extensive history

- *Belief state*

$$b_{t+1}(s') = \frac{O(s', a_t, z_{t+1}) \sum_{s \in S} T(s, a_t, s') b_t(s)}{\Pr(z_{t+1} | a_t, b_t)}$$

- Sufficient representation for optimal decision-making

POMDP Planning

$$\pi: b \;\rightarrow a$$

$$V^*(b) = \sum_{s \in S} b(s) V^*(s)$$

$$V_{t+1} = \max_{a \in A} [\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} \Pr(z|a,b)\, V_t\big(b_{a,z}\big)]$$

POMDP Planning – MDP Approach

- Convert POMDP to MDP
  - Each MDP state is a “belief state” from the POMDP
- Run value iteration
- Problem: belief state is continuous and infinite!
- P-Space complete.
  - Algorithms trade off quality of plan for speed of computation

POMPD Planning – Greedy

Consider reward for taking an action + the value of the state it leads to, times the probability of ending up in that state. Value of current state is result of action which leads to the highest value

- Can use *greedy* methods
- Initially assume POMPD is fully observable and solve *Bellman* equation

$$v(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' | s,a) v(s') \right]$$

- Optimal action to execute in state *s*:

$$a(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' | s,a) v(s') \right]$$

POMDP Planning – Greedy

- *"most likely state" strategy* – execute action that is assigned to most likely state
- *"voting" strategy* – execute action with highest probability mass according to α

$$\arg\max_{a \in A} \sum_{s \in S | a(s) = a} \alpha(s)$$

Each state gets to vote for the action it thinks is optimal weighed by the probability of being in that state. If robot is not likely in your state, your state's vote doesn't mean much.

- *"completely observable after the first step" strategy* – execute action

$$\arg\max_{a \in A} \sum_{s \in S} \alpha(s)(r(s,a) + \gamma \sum_{s' \in S} p(s' | s,a) v(s'))$$

  – Can choose second best action if all disagree on first action but agree on second

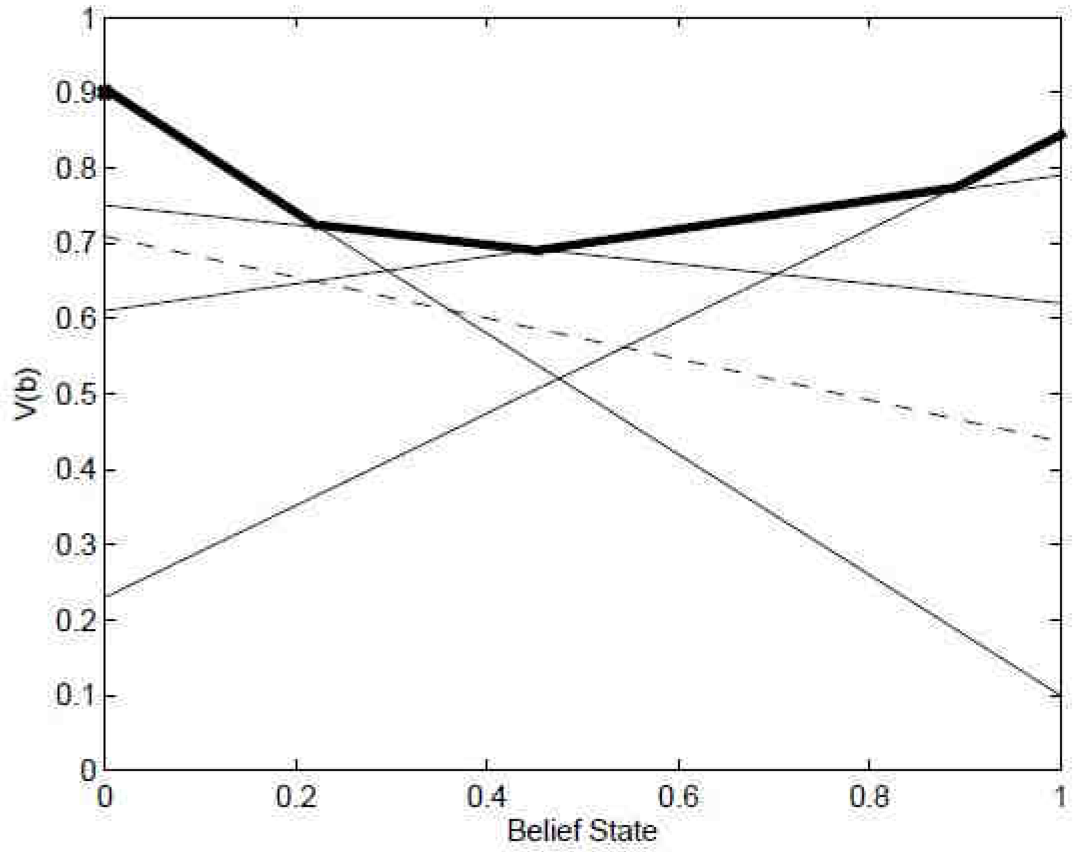
Value Function

- Resulting value function is piecewise linear, convex function over continuous belief state
- Can be represented as a set of α-vectors, each with an associated action

$$\Gamma = \{\alpha_0, \alpha_1, \dots, \alpha_m\}$$

$$V_t(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} \alpha(s) b(s)$$

Value Function



Value Iterations

- Express value function in terms of α-vectors

$$V_{t+1}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} \max_{\alpha \in \Gamma_t} \sum_{s \in S} \sum_{s' \in S} T(s,a,s'), O(s,a,z) V_t(b_{a,z}) \right]$$

- Initialize set of α-vectors for single step horizon

$$\Gamma_1^a = \{\alpha^{a,*} | \alpha^{a,*}(s) = R(s,a)\}$$

Value Iteration

- Construct intermediate alpha sets

$$\Gamma_{t+1}^{a,*} = \{\alpha^{a,*} | \alpha^{a,*}(s) = R(s,a)\}$$

$$\Gamma_{t+1}^{a,z} = \left\{ \alpha^{a,z} | \alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha'(s') \quad \forall \alpha' \in \Gamma_t \right\}$$

- Essentially, we are considering all possible outcomes of observations, *z*, when executing action, *a*, and selecting α-vectors from the previous iteration

Value Iteration

- Construct second intermediate set

$$\Gamma_{t+1}^{a} = \{\Gamma_{t+1}^{a,*} \oplus \Gamma_{t+1}^{a,z_1} \oplus \Gamma_{t+1}^{a,z_2} \oplus ... \oplus \Gamma_{t+1}^{a,z_{|Z|}}\} \quad \forall_{a \in A}$$

[1] The cross-sum ($\oplus$) operator is defined: for sets $A = \{a_1, a_2, \ldots, a_{|A|}\}$, $B = \{b_1, b_2, \ldots, b_{|B|}\}$, $A \oplus B = \{a_1 + b_1, a_1 + b_2, \ldots, a_1 + b_{|B|}, a_2 + b_1, \ldots, a_{|A|} + b_{|B|}\}$

- Construct final vectors

$$\Gamma_{t+1} = \bigcup_{a \in A} \Gamma_{t+1}^{a}$$

POMDP Solutions

- Exact solutions are very difficult: $O(|S|^2|A||\Gamma|^{|Z|})$ for each iteration
- Pruning
- Recent techniques use point-based sampling approaches (similar to particle filters)

POMDP-based Navigation Architecture

- Navigation layer gives directives to the obstacle avoidance layer
- “Go straight”, “Turn right”, “Turn left”
- Obstacle avoidance layer handles avoiding obstacles
  - Tries to follow directives
  - Tries to stay in middle of hall

Architecture

- *Motion Reports* – what the robot actually did (or thought it did)
- *Sensor Reports* – virtual sensors
  - Report what robot sees left, right, and in front
  - Derived from raw sensor data in occupancy grid
  - Currently uses only sonar, but easy to integrate more sensors through occupancy grid

The POMDP

- POMDP represents topological information
  - Hallways, rooms, doorways modeled
  - Length information
- Map discretized to 1 meter resolution
- Each location represent by four states
  - One state for each possible orientation at that location

Observation Probabilities

- $q_i(f|s)$ – probability of seeing feature f in state s
- Classes of states: *wall, near-wall, open, closed-door, open-door, door*
- Ex: left sensor:
  - $q_{left\ sensor}(wall|open) = 0.05$
  - $q_{left\ sensor}(small\_opening|open) = 0.20$
  - $q_{left\ sensor}(medium\_opening|open) = 0.40$
  - $q_{left\ sensor}(large\_opening|open) = 0.30$
  - $q_{left\ sensor}(unknown|open) = 0.05$

Modeling Actions

- Motion reports used to pose estimation
- Motion directives used for policy generation
- “turn left”, “turn right” reliably lead to same state
- “move forward” can lead to different states due to slippage and dead reckoning uncertainty

Modeling Corridors

- Topological edges – key to approach
- If exact length is known, simple Markov chain used
- If approximate length used, parallel chains used
  - Actually, not efficient
- Instead, combine both to form “come from” Markov model
- Actually use parallel chains in architecture

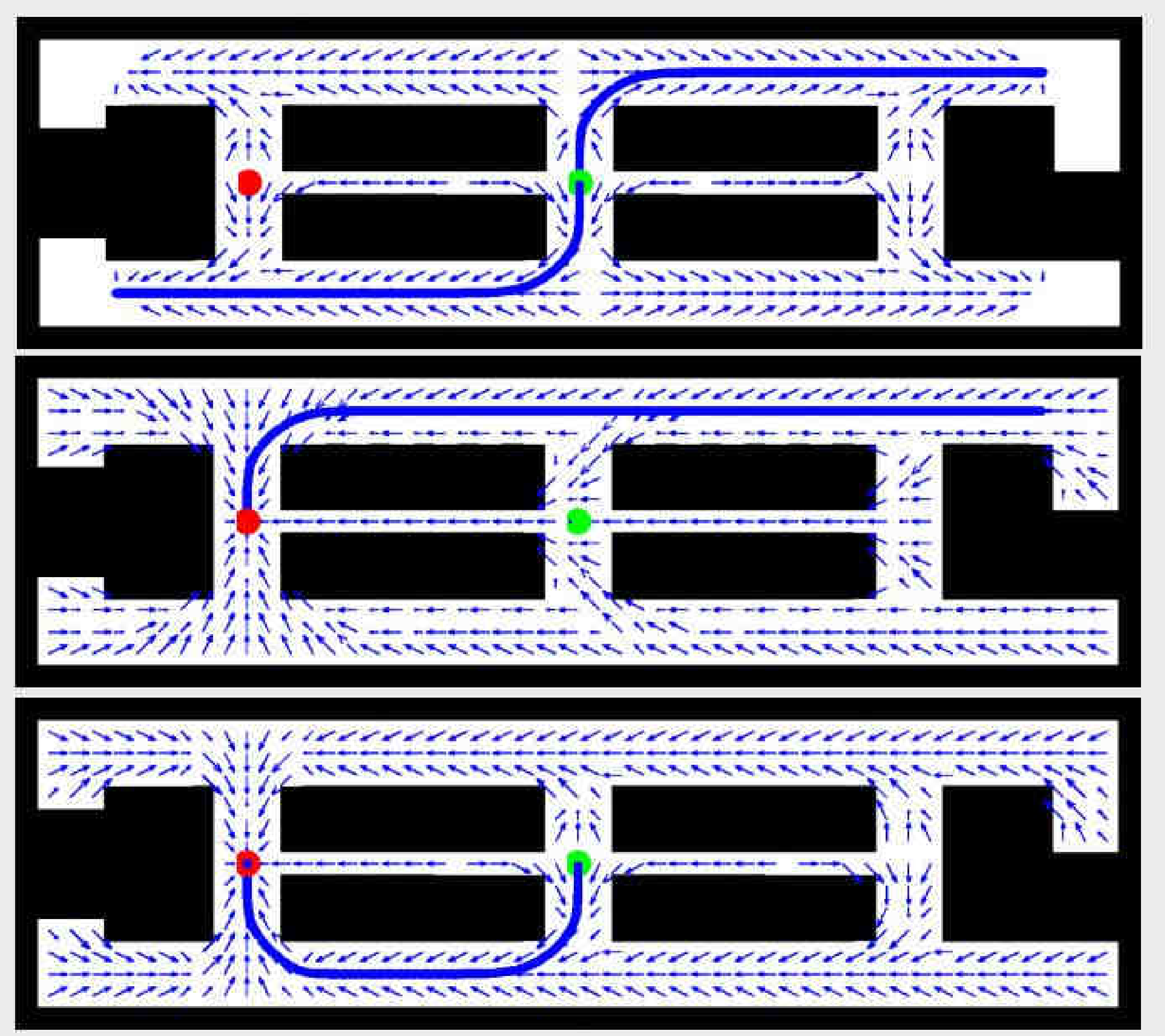
Using the POMDP

- Robot moves
- Update belief state using POMDP (state estimation)
- Motion reports tend to increase pose uncertainty
- Sensor reports tend to decrease pose uncertainty

Policy Generation and Directive Selection

- Uses decision theoretic planner to find path to minimize travel time
  - Takes into account that robot can miss turns and corridors can be blocked
- Navigation layer converts path into directives
- Use greedy policy for choosing directive during execution

Navigation



Other Examples

- Tiger
- Navigation
- Dialogue

POMDP Learning

- Uses Baum-Welch to learn POMDP during execution
- Three steps:
  - Calculate α, β, *scale* using forward-backward algorithm
  - Calculate γ values
  - Use frequency-counting re-estimation formulae to adjust probabilities

Step 1

Probability of being in state *s* after making a series of actions and observing a set of observations

$$\boxed{\alpha_t(s) = p(s_t = s \mid o_{1..t}, a_{1..t-1}) \quad \forall s \in S, t = 1..T}$$

$$scale'_1 := \sum_{s \in S} q(o_1 \mid s)\pi(s)$$

Initialize some stuff

$$\alpha'_1(s) := \frac{q(o_1 \mid s)\pi(s)}{scale} \quad \forall s \in S$$

For t:=1 to T-1

$$scale'_{t+1} := \sum_{s \in S} q(o_{t+1} \mid s) \sum_{s' \in S \mid a_t \in A(s')} p(s \mid s', a)\alpha'_t(s')$$

Probability of being in state *s* at time *t+1*. Consider all the states you could have been in, the probability of being in those states, and the probability of transitioning from those states to state *s* with the action taken (same as State Estimation slide)

$$\alpha'_{t+1}(s) := \frac{q(o_{t+1} \mid s) \sum_{s' \in S \mid a_t \in A} p(s \mid s', a_t)\alpha'_t(s')}{scale'_{t+1}} \quad \forall s \in S$$

Step 1 (cont)

$$\beta_t(s) = \frac{1}{scale_t} \quad \forall s \in S, t = 1..T$$

For t:=T-1 downto 1

$$\beta_t(s) := \begin{cases} \dfrac{\sum_{s' \in S} p(s' \mid s, a_t) q(o_{t+1} \mid s') \beta_{t+1}}{scale} & \forall s \in S, a_t \in A(s) \\ 0 & \forall s \in S, a_t \notin A(s) \end{cases}$$

Probability of being in state *s* at time *t* and then observing the remaining observations, i.e. finishing off the sequence

Step 2

$$\gamma_t(s,s') = p(s_t = s, s_{t+1} = s' \mid o_{1..T}, a_{1..T-1}) \quad t = 1..T-1, s, s' \in S, a_t \in A(s)$$

$$\gamma_t(s) = p(s_t = s \mid o_{1..T}, a_{1..T-1}) \quad t = 1..T, s \in S$$

Probability that you moved from state *s* to state *s′* at time *t* given all the observations and actions taken

Probability of being in state *s* at time *t* given all the observations and actions taken

$$\gamma_t(s,s') := \alpha_t(s) p(s' \mid s, a_t) q(o_{t+1} \mid s') \beta_{t+1}(s') \quad t = 1..T-1, s, s' \in S, a_t \in A(s)$$

α gets you to state *s*

Move between *s* and *s′* with action a_t

Make the correct observation

Finish off the observations correctly

$$\gamma_t(s) := scale_t \alpha_t(s) \beta_t(s) \quad t = 1..T, s \in S$$

Step 3 – Update Estimates

$$\pi^*(s) := \gamma_1(s)$$

Update initial state estimates

$$p^*(s'|s,a) := \frac{\sum_{t=1..T-1|a_t=a} \gamma_t(s,s')}{\sum_{t=1..T-1|a_t=a} \gamma_t(s)} \quad \forall s, s' \in S, a \in A(s)$$

Frequency-counting. Count the number of times you were in state s, executed action *a*, and moved to state *s'* versus the number of times you were in state *s* and executed action *a*.

$$q^*(o|s) := \frac{\sum_{t=1..T|o_t=o} \gamma_t(s)}{\sum_{t=1..T} \gamma_t(s)} \quad \forall s \in S, \forall o \in O$$

Frequency-counting. Count the number of times you see observation *o* in state *s* versus the number of times you were in state *s*

Most Likely Path

- Determine most likely state sequence from observations
- Use Viterbi algorithm to compute most likely path

$$scale'_1 := \sum_{s \in S} q(o_1 \mid s)\pi(s)$$

$$\alpha'_1(s) := \frac{q(o_1 \mid s)\pi(s)}{scale} \quad \forall s \in S$$

For t:=1 to T-1

$$scale'_{t+1} := \sum_{s \in S} q(o_{t+1} \mid s) \max_{s' \in S \mid a \in A(s')} p(s \mid s', a)\alpha'_t(s')$$

$$\alpha'_{t+1}(s) := \frac{q(o_{t+1} \mid s) \max_{s' \in S \mid a_t \in A} p(s \mid s', a_t)\alpha'_t(s')}{scale'_{t+1}} \quad \forall s \in S$$

Similar to state estimation

$$s^*_T := \max_{s \in S} \alpha'_T(s)$$

For t:=T-1 to 1

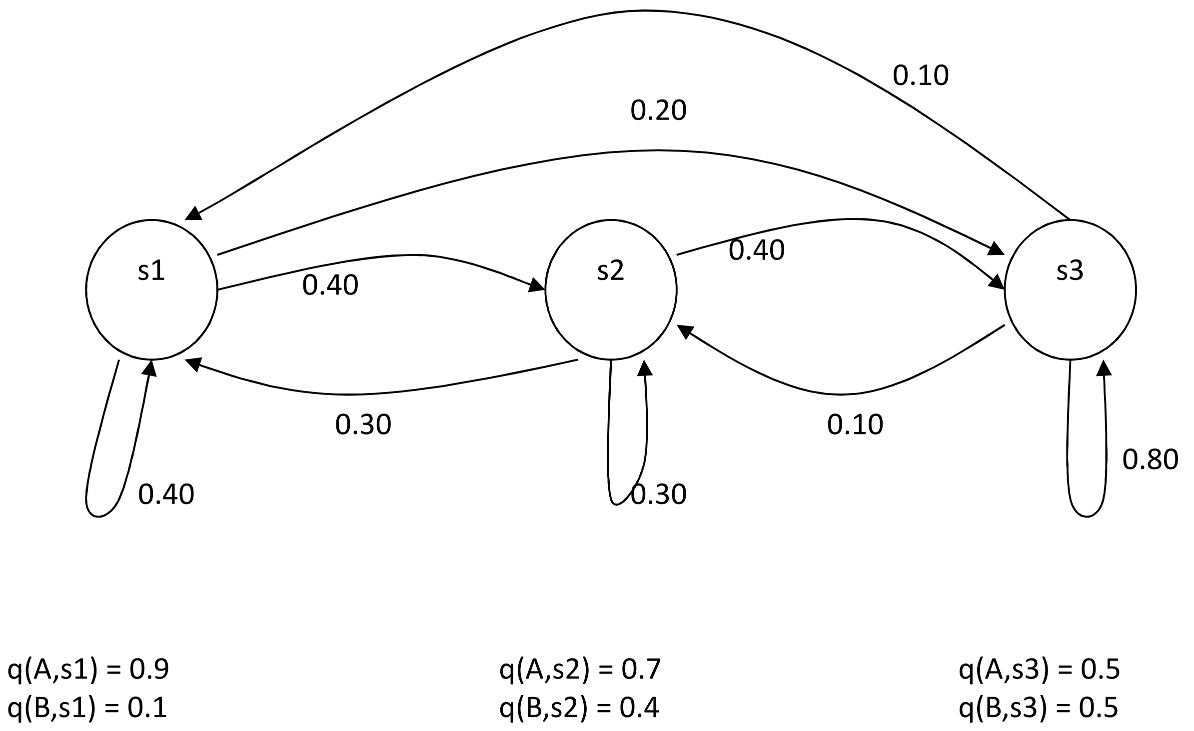
$$s^*_t = \arg\max_{s' \in S \mid a_t \in A(s')} p(s^*_{t+1} \mid s', a)\alpha'_t(s')$$

Find most likely state at time *t*

Hidden Markov Models

- Probabilistic models used to represent non-deterministic processes in partially observable domains
- Set of states S
- Set of observations O
- Initial state distribution π
- $p(s'|s)$ – Probability of transitioning from s to s'
- $q(o|s)$ – Probability of making observation o in state s
- *Markov Property* revisited – Observation in current state depends only on current state, not how the current state was reached

Example – HMMs as Process



Three Questions

- Given a sequence of observations, what is the most likely state at time at time *t*. (state estimation)
- Given a sequence of observations, what is the most likely sequence of states through the HMM. (Viterbi algorithm)
- Training!!! (Baum-Welch)

HMMs in Use

- Good for noisy sensors
- Time Variant Processes
- Speech
  - Phonemes sound similar
  - Accents
  - Stretch words
- Robot Localization
  - Noisy sensors
  - Noisy actuators
- Gesture Recognition
  - Vision
  - Gestures performed at different speeds

Reinforcement Learning

- Assume world is MDP but we don't have models. (Don't have *T(s,a,s')* or *R(s,a)*

- Need to determine policy through execution

TD(λ)

- Estimate value function

$$V(s) = V(s) + \alpha\big(R + \gamma V(s') - V(s)\big)$$

- TD(0) – only update current state
- TD(λ) – update states visited recently

- On-policy, model-free

Q-learning

- Estimate Q value

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + max_{a'}\, Q(s',a') - Q(s,a))$$

- Off-policy, model-free
- Will converge to optimal policy with enough data

- Exploration vs. Exploitation
  - Need to explore
  - One approach: select random action with ε likelihood

Model-based methods

- Gather statistics
- Estimate $T(s,a,s')$, $R(s,a)$ from experience
- Run value iterations