CS 287: Advanced Robotics
Fall 2009

Lecture 1: Introduction

Pieter Abbeel
UC Berkeley EECS

WWW

- http://www.cs.berkeley.edu/~pabbeel/cs287-fa09

University of California at Berkeley
Dept of Electrical Engineering & Computer Sciences

CS 287: Advanced Robotics, Fall 2009

Instructor: Pieter Abbeel
Lectures: Tuesdays and Thursdays, 12:30pm-2:00pm, 405 Soda Hall
Office Hours: Thursdays 2:00-3:00pm (and by email arrangement) in 746 Sutardja Dai Hall

Announcements

- Announcements
- Assignments
- Course description
- Prerequisites
- Grading
- Assignment policy
- Syllabus and materials
- Related materials

Announcements

- Communication:
  - Announcements: webpage
  - Email: pabbeel@cs.berkeley.edu
  - Office hours: Thursday 2-3pm + by email arrangement, 746 SDH
- Enrollment:
  - Undergrads stay after lecture and see me

Class Details

- Prerequisites:
  - Familiarity with mathematical proofs, probability, algorithms, linear algebra, calculus.
  - Ability to implement algorithmic ideas in code.
  - Strong interest in robotics
- Work and grading
  - Four large assignments (4 * 15%)
  - One smaller assignment (5%)
  - Open-ended final project (35%)
- Collaboration policy: Students may discuss assignments with each other. However, each student must code up their solutions independently and write down their answers independently.

Class Goals

- Learn the issues and techniques underneath state of the art robotic systems
- Build and experiment with some of the prevalent algorithms
- Be able to understand research papers in the field
  - Main conferences: ICRA, IROS, RSS, ISER, ISRR
  - Main journals: IJRR, T-RO, Autonomous Robots
- Try out some ideas / extensions of your own

Lecture outline

- Logistics --- questions? [textbook slide forthcoming]
- A few sample robotic success stories
- Outline of topics to be covered

Driverless cars

- Darpa Grand Challenge
  - First long-distance driverless car competition
  - 2004: CMU vehicle drove 7.36 out of 150 miles
  - 2005: 5 teams finished, Stanford team won
- Darpa Urban Challenge (2007)
  - Urban environment: other vehicles present
  - 6 teams finished (CMU won)
- Ernst Dickmanns / Mercedes Benz: autonomous car on European highways
  - Human in car for interventions
  - Paris highway and 1758km trip Munich -> Odense, lane changes at up to 140km/h; longest autonomous stretch: 158km

Kalman filtering, Lyapunov, LQR, mapping, (terrain & object recognition)

Autonomous Helicopter Flight

[Coates, Abbeel & Ng]

Kalman filtering, model-predictive control, LQR, system ID, trajectory learning

Four-legged locomotion

[Kolter, Abbeel & Ng]

inverse reinforcement learning, hierarchical RL, value iteration, receding horizon control, motion planning

Two-legged locomotion

[Tedrake +al.]

TD learning, policy search, Poincare map, stability

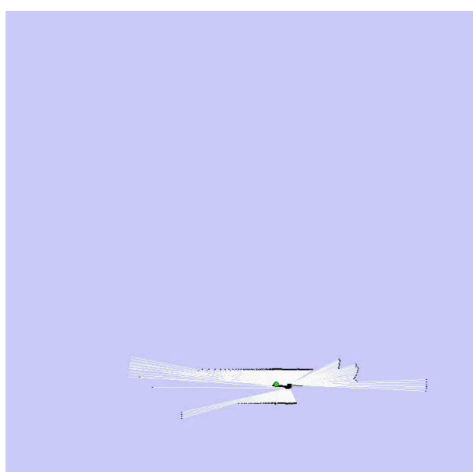
Mapping

[Video from W. Burgard and D. Haehnel]

"baseline" : Raw odometry data + laser range finder scans

Mapping

[Video from W. Burgard and D. Haehnel]



FastSLAM: particle filter + occupancy grid mapping

Mobile Manipulation

[Quigley, Gould, Saxena, Ng + al.]

SLAM, localization, motion planning for navigation and grasping, grasp point selection, (visual category recognition, speech recognition and synthesis)

Outline of Topics

- **Control:** underactuation, controllability, Lyapunov, dynamic programming, LQR, feedback linearization, MPC
- **Estimation:** Bayes filters, KF, EKF, UKF, particle filter, occupancy grid mapping, EKF slam, GraphSLAM, SEIF, FastSLAM
- **Manipulation and grasping:** force closure, grasp point selection, visual servo-ing, more sub-topics tbd
- **Reinforcement learning:** value iteration, policy iteration, linear programming, Q learning, TD, value function approximation, Sarsa, LSTD, LSPI, policy gradient, inverse reinforcement learning, reward shaping, hierarchical reinforcement learning, inference based methods, exploration vs. exploitation
- **Brief coverage of:** system identification, simulation, pomdps, k-armed bandits, separation principle
- **Case studies:** autonomous helicopter, Darpa Grand/Urban Challenge, walking, mobile manipulation.

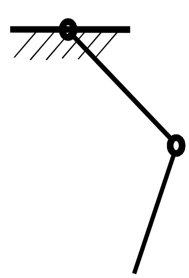
1. Control

- Overarching theme: mathematically capture
  - What makes control problems hard
  - What techniques do we have available to tackle the hard problems
- E.g.: “Helicopters have underactuated, non-minimum phase, highly non-linear and stochastic (within our modeling capabilities) dynamics.”

  → Hard or easy to control?

1. Control (ctd)

- Under-actuated vs. fully actuated
  - Example: acrobot swing-up and balance task



1. Control (ctd)

- Other mathematical formalizations of what makes some control problems easy/hard:
  - Linear vs. non-linear
  - Minimum-phase vs. non-minimum phase
  - Deterministic vs. stochastic
- Solution and proof techniques we will study:
  - Lyapunov, dynamic programming, LQR, feedback linearization, MPC

2. Estimation

- Bayes filters: KF, EKF, UKF, particle filter
- One of the key estimation problems in robotics: Simultaneous Localization And Mapping (SLAM)
- Essence: compute posterior over robot pose(s) and environment map given
  - (i) Sensor model
  - (ii) Robot motion model
- Challenge: Computationally impractical to compute exact posterior because this is a very high-dimensional distribution to represent
- [You will benefit from 281A for this part of the course.]

3. Grasping and Manipulation

- Extensive mathematical theory on grasping: force closure, types of contact, robustness of grasp
- Empirical studies showcasing the relatively small vocabulary of grasps being used by humans (compared to the number of degrees of freedom in the human hand)
- Perception: grasp point detection

4. Reinforcement learning

- Learning to act, often in discrete state spaces
- value iteration, policy iteration, linear programming, Q learning, TD, value function approximation, Sarsa, LSTD, LSPI, policy gradient, inverse reinforcement learning, reward shaping, hierarchical reinforcement learning, inference based methods, exploration vs. exploitation

5. Misc. Topics

- system identification: frequency domain vs. time domain
- Simulation / FEM
- Pomdps
- k-armed bandits
- separation principle
- …

Reading materials

- Control
  - Tedrake lecture notes 6.832: https://svn.csail.mit.edu/russt_public/6.832/underactuated.pdf
- Estimation
  - Probabilistic Robotics, Thrun, Burgard and Fox.
- Manipulation and grasping
  - -
- Reinforcement learning
  - Sutton and Barto, Reinforcement Learning (free online)
- Misc. topics
  - -

- Next lecture we will start with our study of control!

CS 287: Advanced Robotics
Fall 2009

Lecture 2: Control 1: Feedforward, feedback, PID, Lyapunov direct method

Pieter Abbeel
UC Berkeley EECS

Announcements

- Office hours: Thursdays 2-3pm + by email arrangement, 746 SDH
  - SDH 7th floor should be unlocked during office hours on Thursdays
- Questions about last lecture?

CS 287 Advanced Robotics

- **Control**
- Estimation
- Manipulation/Grasping
- Reinforcement Learning
- Misc. Topics
- Case Studies

Control in CS287

- Overarching goal:
  - Understand what makes control problems hard
  - What techniques do we have available to tackle the hard (and the easy) problems
- Any applicability of control outside robotics? Yes, many!
  - Process industry, feedback in nature, networks and computing systems, economics, …
    - [See, e.g., Chapter 1 of Astrom and Murray, http://www.cds.caltech.edu/~murray/amwiki/Main_Page, for more details---_optional_ reading. Fwiw: Astrom and Murray is a great read on mostly classical feedback control and is freely available at above link.]
  - We will not have time to study these application areas within CS287 [except for perhaps in your final project!]

Today's lecture

- Feedforward vs. feedback
- PID (Proportional Integral Derivative)
- Lyapunov direct method --- a method that can be helpful in proving guarantees about controllers
- Reading materials:
  - Astrom and Murray, 10.3
  - Tedrake, 1.2
  - Optional: Slotine and Li, Example 3.21.

Based on a survey of over eleven thousand controllers in the refining, chemicals and pulp and paper industries, 97% of regulatory controllers utilize PID feedback. L. Desborough and R. Miller, 2002 [DM02]. [Quote from Astrom and Murray, 2009]

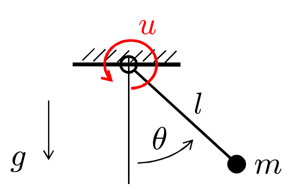
Today's lecture

- Practical result: can build a trajectory controller for a fully actuated robot arm

- Our abstraction: torque control input to motor, read out angle [in practice: voltages and encoder values]

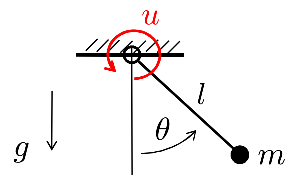
Intermezzo: Unconventional (?) robot arm use

Single link manipulator (aka the simple pendulum)



$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t) \qquad I = ml^2$$

Single link manipulator



$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

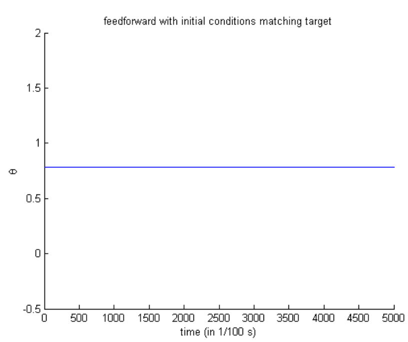
How to hold arm at θ = 45 degrees?

The Matlab code that generated all discussed simulations will be posted on www.

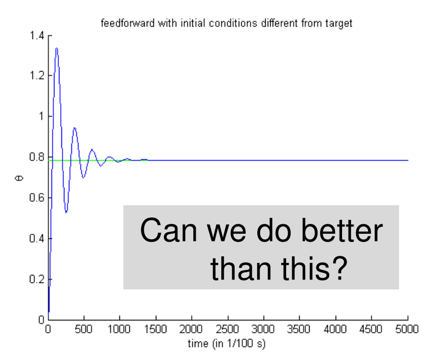
Single link manipulator

Simulation results:

$$I\ddot{\theta}(t) + c\dot{\theta}(t) + mgl\sin\theta(t) = u(t), \quad u = mgl\sin\tfrac{\pi}{4}$$



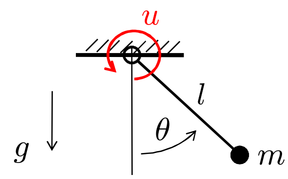
$\theta(0) = \frac{\pi}{4}, \dot{\theta}(0) = 0$



Can we do better than this?

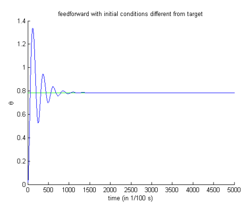
$\theta(0) = 0, \dot{\theta}(0) = 0$

Feedforward control



$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

How to make arm follow a trajectory θ^*(t) ?



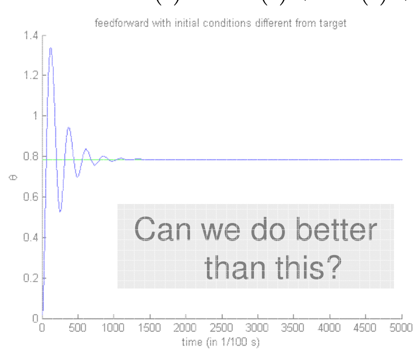
$\theta(0) = 0, \dot{\theta}(0) = 0$

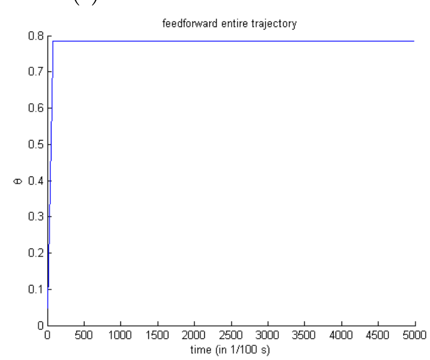
$$u(t) = I\ddot{\theta}^*(t) + c\dot{\theta}^*(t) + mgl\sin\theta^*(t)$$

Feedforward control

Simulation results:

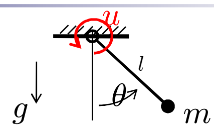
$$I\ddot{\theta}(t) + c\dot{\theta}(t) + mgl\sin\theta(t) = u(t) \qquad \theta(0) = 0, \dot{\theta}(0) = 0$$
$$u(t) = I\ddot{\theta}^*(t) + c\dot{\theta}^*(t) + mgl\sin\theta^*(t)$$





n DOF (degrees of freedom) manipulator?

- Thus far:



$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$

- n DOF manipulator: standard manipulator equations

$$H(q)\ddot{q} + C(q,\dot{q}) + G(q) = B(q)u$$

  - H : “inertial matrix,” full rank
  - B : identity matrix if every joint is actuated

- → Given trajectory $q(t)$, can readily solve for feedforward controls $u(t)$ for all times t

Fully-Actuated vs. Underactuated

- A system is fully actuated when in a certain state (q,\dot{q},t) if, when in that state, it can be controlled to instantaneously accelerate in any direction.
- Many systems of interest are of the form:

$$\ddot{q} = f_1(q,\dot{q},t) + f_2(q,\dot{q},t)u \qquad (1)$$

- Defn. **Fully actuated**: A control system described by Eqn. (1) is fully-actuated in state (q,\dot{q},t) if it is able to command an instantaneous acceleration in an arbitrary direction in q:

$$\mathrm{rank} f_2(q,\dot{q},t) = \dim q$$

- Defn. **Underactuated**: A control system described by Eqn. (1) is underactuated in configuration (q,\dot{q},t) if it is not able to command an instantaneous acceleration in an arbitrary direction in q:

$$\mathrm{rank} f_2(q,\dot{q},t) < \dim q$$

- [See also, Tedrake, Section 1.2.]

Fully-Actuated vs. Underactuated

$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)u$ fully actuated in $(q, \dot{q}, t)$ iff $\text{rank} f_2(q, \dot{q}, t) = \dim q$.

- Hence, for any fully actuated system, we can follow a trajectory by simply solving for $u(t)$:

$$u(t) = f_2^{-1}(q, \dot{q}, t)\,(\ddot{q} - f_1(q, \dot{q}, t))$$

- [We can also transform it into a linear system through a change of variables from u to v:

$$\ddot{q}(t) = v(t)$$
$$u(t) = f_2^{-1}(q, \dot{q}, t)\,(v(t) - f_1(q, \dot{q}, t))$$

The literature on control for linear systems is very extensive and hence this can be useful. This is an example of feedback linearization. More on this in future lectures.]

Fully-Actuated vs. Underactuated

$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)u$ fully actuated in $(q, \dot{q}, t)$ iff $\text{rank} f_2(q, \dot{q}, t) = \dim q$.

- **n DOF manipulator**

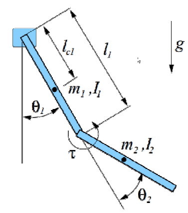
$$H(q)\ddot{q} + C(q, \dot{q}) + G(q) = B(q)u$$
$$f_2 = H^{-1}B,\ H \text{ full rank},\ B = I, \text{ hence } \text{rank}(H^{-1}B) = \text{rank}(B)$$

  - All joints actuated → rank(B) = n → fully actuated
  - Only $p < n$ joints actuated → rank(B) = p → underactuated

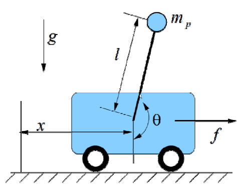
Example underactuated systems

- Car

- Acrobot

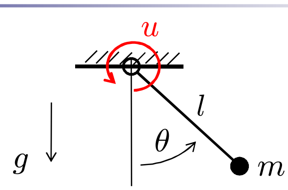


- Cart-pole



- Helicopter

Fully actuated systems: is our feedforward control solution sufficient in practice?

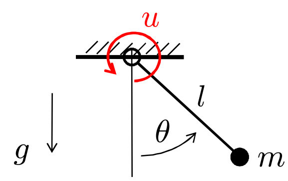


$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

Task: hold arm at 45 degrees.

- What if parameters off? --- by 5%, 10%, 20%, …
- What is the effect of perturbations?

Fully actuated systems: is our feedforward control solution sufficient in practice?

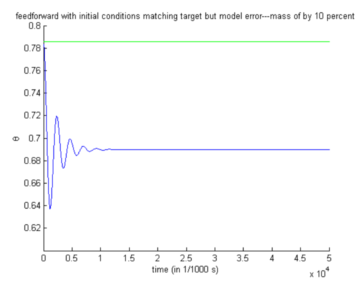


$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

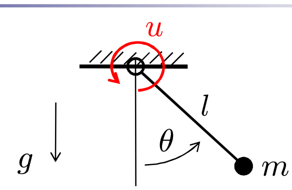
Task: hold arm at 45 degrees.

- Mass off by 10%:

→ steady-state error



Fully actuated systems: is our feedforward control solution sufficient in practice?



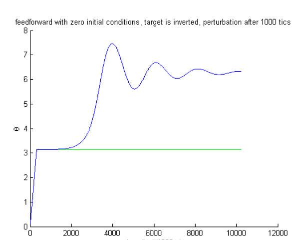
$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

Task: swing arm up to **180** degrees and hold there

- Perturbation after 1sec:

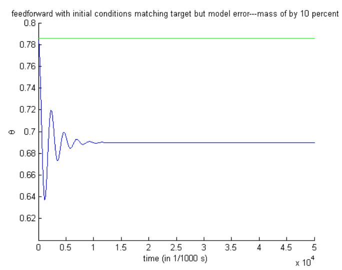
→ Does **not** recover

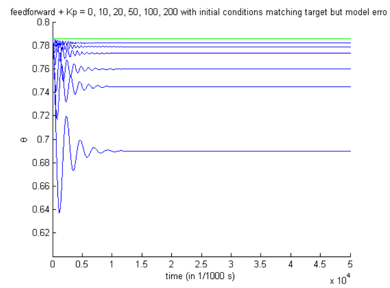
[θ = 180 is an “unstable” equilibrium point]



Proportional control

Task: hold arm at 45 degrees





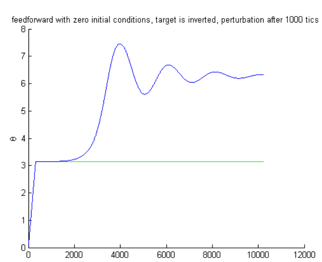
$$u(t) = u_{\text{feedforward}}(t)$$

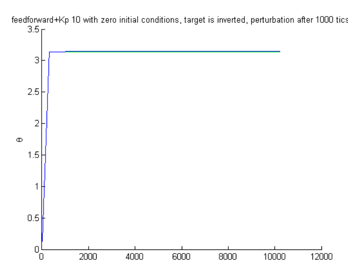
$$u(t) = u_{\text{feedforward}}(t) + K_p(q_{\text{desired}}(t) - q(t))$$

- Feedback can provide
  - Robustness to model errors
- However, still:
  - Overshoot issues --- ignoring momentum/velocity!
  - Steady-state error --- simply crank up the gain?

Proportional control

Task: swing arm up to **180** degrees and hold there





$$u(t) = u_{\text{feedforward}}(t)$$

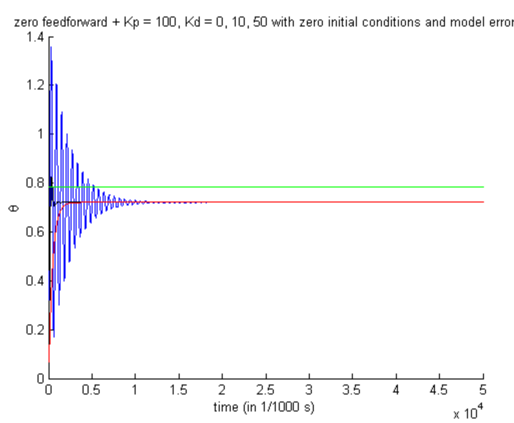
$$u(t) = u_{\text{feedforward}}(t) + K_p(q_{\text{desired}}(t) - q(t))$$

Current status

- Feedback can provide
  - *Robustness to model errors*
  - *Stabilization around states which are unstable in open-loop*
- Overshoot issues --- ignoring momentum/velocity!
- Steady-state error --- simply crank up the gain?

PD control

$$u(t) = K_p(q_{\text{desired}}(t) - q(t)) + K_d(\dot{q}_{\text{desired}} - \dot{q}(t))$$

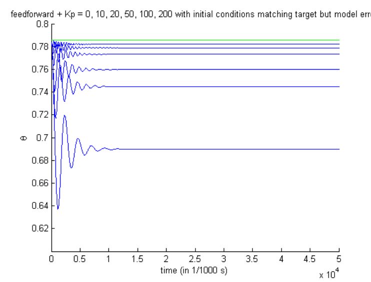


Eliminate steady state error by cranking up Kp ?

$$I\ddot{\theta}(t) + b\dot{\theta}(t) + mgl\sin\theta(t) = u(t)$$

$$u(t) = u_{\text{feedforward}}(t) + K_p(q_{\text{desired}}(t) - q(t))$$

Task: hold arm at 45 degrees



In steady-state, $\ddot{q} = \dot{q} = 0$ and we get:

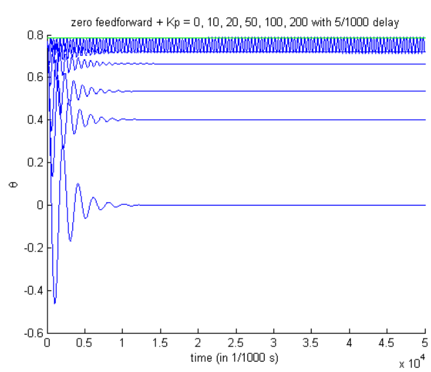
$$mgl\sin\theta = u_{\text{feedforward}} + K_p(\theta^* - \theta)$$

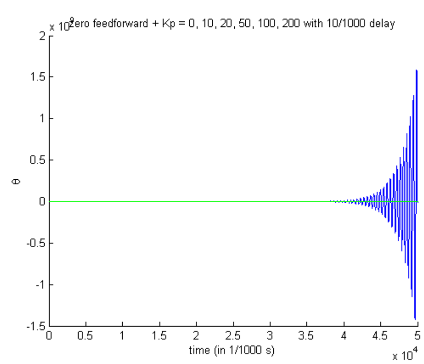
Using some trigoniometry and assuming θ is close to θ^* we get:

$$\theta - \theta^* = \frac{u_{\text{feedforward}} - mgl\sin\theta^*}{K_p + mgl\cos\theta^*}$$

Eliminate steady state error by cranking up Kp ?

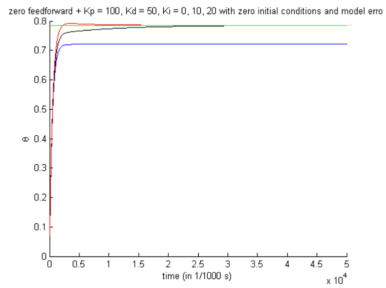
$$u(t + \delta t) = K_p(q_{\text{desired}}(t) - q(t))$$





PID

$$u(t) = K_p(q_{\text{desired}}(t) - q(t)) + K_d(\dot{q}_{\text{desired}} - \dot{q}(t)) + K_i \int_0^t (q_{\text{desired}}(\tau) - q(\tau))d\tau$$



- Zero error in steady-state: [assumes steady-state is achieved!]

$\ddot{q} = \dot{q} = 0$, $\dot{u} = 0$, hence, taking derivatives of above:

$$\begin{aligned} \dot{u} &= K_p(\dot{q}_{\text{desired}} - \dot{q}(t)) + K_d(\ddot{q}_{\text{desired}} - \ddot{q}(t)) + K_i(q_{\text{desired}}(t) - q(t)) \\ 0 &= K_k(q_{\text{desired}}(t) - q(t)) \end{aligned}$$

Recap so far

- Given a ***fully actuated system*** and a (smooth) target trajectory
  - Can solve dynamics equations for required control inputs = "feedforward controls"
  - Feedforward control is insufficient in presence of
    - Model inaccuracy
    - Perturbations + instability
  - Proportional feedback control can alleviate some of the above issues.
    - Steady state error reduced by (roughly) factor K_p, but large K_p can be problematic in presence of delay → Add integral term
    - Ignores momentum → Add derivative term
- **Remaining questions:**
  - How to choose PID constants? Aka "tuning"
  - Any guarantees?

PID tuning

- Typically done by hand (3 numbers to play with) [policy search should be able to automate this in many settings]
- Ziegler-Nichols method (1940s) provides recipe for starting point
  - Frequency response method
  - Step response method
- Recipe results from
  - (a) Extensively hand-tuning controllers for many settings
  - (b) Fitting a function that maps from easy to measure parameters to the three gains

[See also Astrom and Murray Section 10.3]

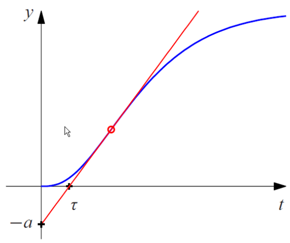
PID tuning: Ziegler-Nichols frequency domain method

- Set derivative and integral gain to zero
- Drive up the proportional gain until steady oscillation occurs, record the corresponding gain k_c and period T_c
- Use the following table to set the three gains:

Type	k_p	T_i	T_d
P	$0.5k_c$		
PI	$0.4k_c$	$0.8T_c$	
PID	$0.6k_c$	$0.5T_c$	$0.125T_c$

Notation: $K_I = \frac{k_p}{T_i}$, $K_D = k_p T_d$

PID tuning: Ziegler-Nichols step response method

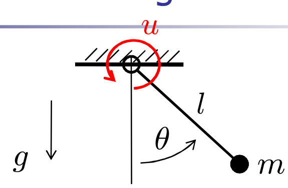


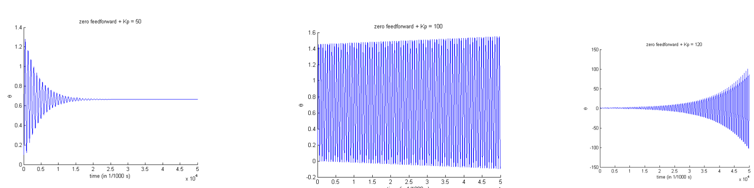
Type	k_p	T_i	T_d
P	1/a		
PI	0.9/a	3τ	
PID	1.2/a	2τ	0.5τ

1. Record open-loop step-response characteristics

2. Read gains out from above table

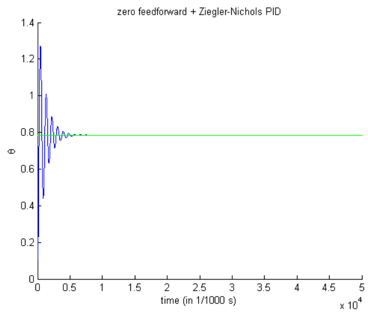
Frequency domain Ziegler-Nichols for single link

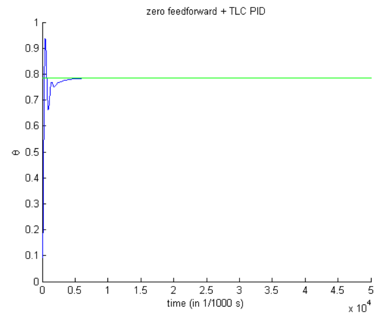




- Kc = 100;
- Tc = 0.63s;

ZN and TLC results





Type	k_p	T_i	T_d
P	$0.5k_c$		
PI	$0.4k_c$	$0.8T_c$	
PID	$0.6k_c$	$0.5T_c$	$0.125T_c$

Tyreus-Luyben tuning chart:
$Kp = k_c/2.2, T_i = 2.2T_c, T_d = T_c/6.3$
Tends to:
increase robustness,
decrease oscillation.

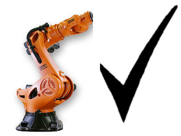
Aside: Integrator wind-up

- Recipe: Stop integrating error when the controls saturate
- Reason: Otherwise it will take a long time to react in the opposite direction in the future.
- Matters in practice!

[See also Astrom and Murray, Section 10.4]

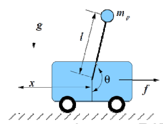
Recap of main points

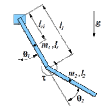
- To control a fully actuated system:
  - Compute feedforward by solving for $u(t)$
  - However, feedforward is insufficient when:
    - Model is imperfect (i.e., always when dealing with real systems)
    - System is unstable
  - Feedback can address these issues
    - Standard feedback: PID
  - In practice, often even only feedback (i.e., without feedforward) can already provide ok results → in these settings, no model needed, which can be very convenient



- In this lecture no solution provided for underactuated systems

?





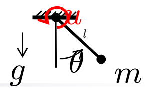
- Note: many underactuated systems do use PID type controllers in their core (e.g., helicopter governor, gyro)

CS 287: Advanced Robotics
Fall 2009

Lecture 3: Control 2: Fully actuated wrap-up/recap, Lyapunov direct method, Optimal control

Pieter Abbeel
UC Berkeley EECS

Fully actuated recap



$$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)u \text{ fully actuated in } (q, \dot{q}, t) \text{ iff } \mathrm{rank} f_2(q, \dot{q}, t) = \mathrm{dim} q.$$

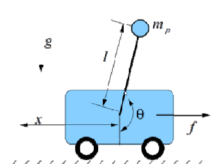
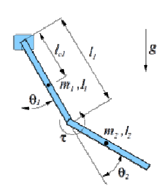
- Given a ***fully actuated system*** and a (smooth) target trajectory q^*
  - Can solve dynamics equations for required control inputs = "feedforward controls"
    $$u_{\mathrm{ff}}(t) = f_2^{-1}(q^*, \dot{q}^*, t)\,(\ddot{q}^* - f_1(q^*, \dot{q}^*, t))$$
  - Feedforward control is insufficient in presence of
    - Model inaccuracy
    - Perturbations + instability
  - Proportional feedback control can alleviate some of the above issues.
    - Steady state error reduced by (roughly) factor K_p, but large K_p can be problematic in presence of delay → Add integral term
    - Ignores momentum → Add derivative term

$$u(t) = u_{\mathrm{ff}}(t) + K_p(q^*(t) - q(t)) + K_d(\dot{q}^*(t) - \dot{q}(t)) + K_i \int_0^t (q^*(\tau) - q(\tau))d\tau$$

  - PID constants require tuning: often by hand, Ziegler-Nichols and TLC provide good starting points, (policy search could automate this)
  - If control inputs do not directly relate to the degrees of freedom, we can use feedback linearization to get that form:
    $$\ddot{q}(t) = v(t) \qquad u(t) = f_2^{-1}(q, \dot{q}, t)\,(v(t) - f_1(q, \dot{q}, t))$$

Today's lecture

- Fully actuated recap [done]
- Aside on integrator wind-up
- Lyapunov direct method --- a method that can be helpful in proving guarantees about controllers
- Optimal control

Readings for today's lecture

- --
- Optional:
  - Tedrake Chapter 3 [optional, nice read on energy pumping control strategies]
  - Slotine and Li, Example 3.21, Global asymptotic stability of a robot position controller [optional]

Aside: Integrator wind-up

$$u(t) = u_{\text{ff}}(t) + K_p(q^*(t) - q(t)) + K_d(\dot{q}^*(t) - \dot{q}(t)) + K_i \int_0^t (q^*(\tau) - q(\tau))d\tau$$

- Recipe: Stop integrating error when the controls saturate
- Reason: Otherwise it will take a long time to react in the opposite direction in the future.
- Matters in practice!

[See also Astrom and Murray, Section 10.4]

Lyapunov

- Lyapunov theory is used to make conclusions about trajectories of a system without finding the trajectories (i.e., solving the differential equation)
- A typical Lyapunov theorem has the form:
  - if there exists a function $V: R^n \to R$ that satisfies some conditions on V and \dot{V}
  - then, trajectories of system satisfy some property
- If such a function V exists we call it a Lyapunov function
- Lyapunov function V can be thought of as generalized energy function for system

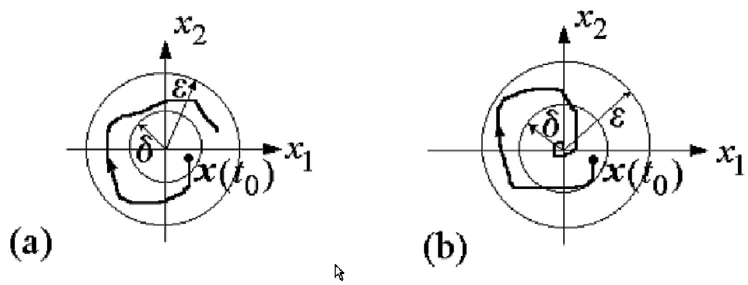
Guarantees?

Equilibrium state. A state x^* is an equilibrium state of the system $\dot{x} = f(x)$ if $f(x^*) = 0$.

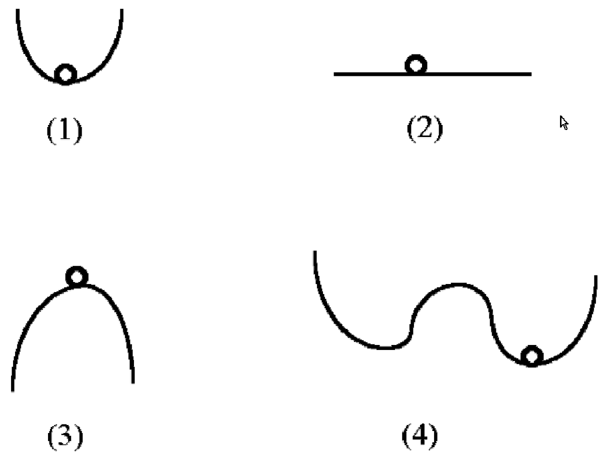
Stability. The equilibrium state x^* is said to be stable if, for any $R > 0$, there exists $r > 0$, such that if $\|x(0) - x^*\| < r$, then $\|x(t) - x^*\| < R$ for all $t \geq 0$. Otherwise, the equilibrium point is unstable.

Asymptotic stability. An equilibrium point x^* is asymptotically stable if it is stable, and if in addition there exists some $r > 0$ such that $\|x(0) - x^*\| < r$ implies that $x(t) \to x^*$ as $t \to \infty$.

Stability illustration



Simple physical examples



Proving stability

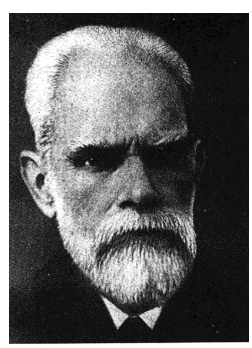
- To prove stability, we need to show something about the solution of a non-linear differential equation for all initial conditions within a certain radius of the equilibrium point.

- Challenge: typically no closed form solution to differential equation!

→ How to analyze / prove stability ??

Alexandr Mikhailovich Lyapunov

- In late 19th century introduced one of the most useful and general approaches for studying stability of non-linear systems.
- [Lyapunov's PhD thesis: 1892]



Александр Михайлович
ЛЯПУНОВ

A Lyapunov boundedness theorem

suppose there is a function V that satisfies

- all sublevel sets of V are bounded
- $\dot{V}(z) \le 0$ for all z

then, all trajectories are bounded, *i.e.*, for each trajectory x there is an R such that $\|x(t)\| \le R$ for all $t \ge 0$

in this case, V is called a Lyapunov function (for the system) that proves the trajectories are bounded

[from Boyd, ee363]

Proof:

to prove it, we note that for any trajectory x

$$V(x(t)) = V(x(0)) + \int_0^t \dot{V}(x(\tau))\, d\tau \le V(x(0))$$

so the whole trajectory lies in $\{z \mid V(z) \le V(x(0))\}$, which is bounded

also shows: every sublevel set $\{z \mid V(z) \le a\}$ is invariant

[from Boyd, ee363]

A Lyapunov global asymptotic stability theorem

suppose there is a function V such that

- V is positive definite
- $\dot{V}(z) < 0$ for all $z \neq 0$, $\dot{V}(0) = 0$

then, every trajectory of $\dot{x} = f(x)$ converges to zero as $t \to \infty$
($i.e.$, the system is globally asymptotically stable)

intepretation:

- V is positive definite generalized energy function
- energy is always dissipated, except at 0

[from Boyd, ee363]

Proof:

Suppose trajectory $x(t)$ does not converge to zero.

$V(x(t))$ is decreasing and nonnegative, so it converges to, say, ϵ as $t \to \infty$.

Since $x(t)$ does not converge to 0, we must have $\epsilon > 0$, so for all t, $\epsilon \leq V(x(t)) \leq V(x(0))$.

$C = \{z | \epsilon \leq V(z) \leq V(x(0))\}$ is closed and bounded, hence compact. So $\dot{V}$ (assumed continuous) attains its supremum on C, i.e., $\sup_{z \in C} \dot{V} = -a < 0$. Since $\dot{V}(x(t)) \leq -a$ for all t, we have

$$V(x(T)) = V(x(0)) + \int_0^T \dot{V}(x(t))dt \leq V(x(0)) - aT$$

which for $T > V(x(0))/a$ implies $V(x(T)) < 0$, a contradition.

So every trajectory $x(t)$ converges to 0, i.e., $\dot{x} = f(x)$ is globally asymptotically stable.

[from Boyd, ee363]

Global invariant set Theorem. Assume that

- $V(x) \to \infty$ as $\|x\| \to \infty$.
- $\dot{V}(x) \leq 0$ over the whole state space.

Let R be the set of all points where $\dot{V(x)} = 0$, and let M be the largest invariant set in R. Then all solutions globally asymptotically converge to M as $t \to \infty$.

Example 1

$$\ddot{q} + \dot{q} + \sin q = u$$

$$u = \sin q + K_p(q^* - q) + K_d(0 - \dot{q})$$

Example 1 (solution)

$$\ddot{q} + \dot{q} + g(q) = u$$
$$u = g(q) + K_d(q^* - q) + K_d(0 - \dot{q})$$

We choose $V = \frac{1}{2}K_p(q - q^*)^2 + \frac{1}{2}\dot{q}^2$.
This gives for $\dot{V}$:

$$\begin{aligned}
\dot{V} &= K_p(q - q^*)\dot{q} + \dot{q}\ddot{q} \\
&= K_p(q - q^*)\dot{q} + \dot{q}\left(K_p(q^* - q) - K_d\dot{q} - \dot{q}\right) \\
&= -(1 + K_d)\dot{q}
\end{aligned}$$

Hence V satisfies: (i) $V(q) \geq 0$ and $= 0$ iff $q = q^*$, (ii) $\dot{V} \leq 0$. Since the arm cannot get "stuck" at any position such that $q \neq 0$ (which can be easily shown by noting that acceleration is non-zero in such situations), the robot arm must settle down at $\dot{q} = 0$ and $q = 0$, according to the invariant set theorem. Thus the system is globally asymptotically stable.

Example 2: PD controllers are stable for fully actuated manipulators

- (Slotine and Li, Example 3.21.)

A converse Lyapunov G.E.S. theorem

suppose there is $\beta > 0$ and M such that each trajectory of $\dot{x} = f(x)$ satisfies

$$\|x(t)\| \leq Me^{-\beta t}\|x(0)\| \text{ for all } t \geq 0$$

(called *global exponential stability*, and is stronger than G.A.S.)

then, there is a Lyapunov function that proves the system is exponentially stable, $i.e.$, there is a function $V : \mathbf{R}^n \to \mathbf{R}$ and constant $\alpha > 0$ s.t.

- V is positive definite
- $\dot{V}(z) \leq -\alpha V(z)$ for all z

[from Boyd, ee363]

Proof of converse G.E.S. Lyapunov theorem

suppose the hypotheses hold, and define

$$V(z) = \int_0^\infty \|x(t)\|^2 \, dt$$

where $x(0) = z$, $\dot{x} = f(x)$

since $\|x(t)\| \leq Me^{-\beta t}\|z\|$, we have

$$V(z) = \int_0^\infty \|x(t)\|^2 \, dt \leq \int_0^\infty M^2 e^{-2\beta t}\|z\|^2 \, dt = \frac{M^2}{2\beta}\|z\|^2$$

(which shows integral is finite)

[from Boyd, ee363]

Finding Lyapunov functions

- there are many different types of Lyapunov theorems
- the key in all cases is to *find* a Lyapunov function and verify that it has the required properties
- there are several approaches to finding Lyapunov functions and verifying the properties

one common approach:

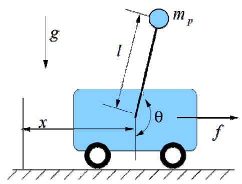
- decide form of Lyapunov function (*e.g.*, quadratic), parametrized by some parameters (called a *Lyapunov function candidate*)
- try to find values of parameters so that the required hypotheses hold

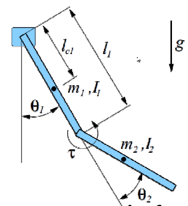
[from Boyd, ee363]

Lyapunov recap

- Enables providing stability guarantees w/o solving the differential equations for all possible initial conditions!
- Tricky part: finding a Lyapunov function
- A lot more to it than we can cover in ½ lecture, but this should provide you with a starting point whenever you might need something like this in the future

Intermezzo on energy pumping



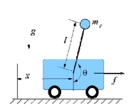


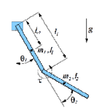
- An interesting and representative approach from non-linear control:
  - Energy pumping to swing up:
    - Write out energy of system E(q, \dot{q})
    - Write out time derivative of energy: \{dot}E (q, \dot{q}, u)
    - Choose u as a function of q and \dot{q} such that energy is steered towards required energy to reach the top
  - Local controller to stabilize at top --- we will see this aspect in detail later.

Energy pumping nicely described in Tedrake Chapter 3. Enjoy the optional read!

Forthcoming lectures

- Optimal control: provides general computational approach to tackle control problems---both under- and fully actuated.
  - Dynamic programming
    - Discretization
  - Dynamic programming for linear systems
    - Extensions to nonlinear settings:
      - Local linearization
      - Differential dynamic programming
      - Feedback linearization
  - Model predictive control (MPC)
  - Examples:





CS 287: Advanced Robotics
Fall 2009

Lecture 4: Control 3: Optimal control---discretization (function approximation)

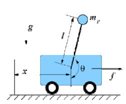
Pieter Abbeel
UC Berkeley EECS

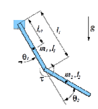
Announcement

- Tuesday Sept 15: **no** lecture

Today and forthcoming lectures

- Optimal control: provides general computational approach to tackle control problems---both under- and fully actuated.
  - Dynamic programming
    - Discretization
  - Dynamic programming for linear systems
    - Extensions to nonlinear settings:
      - Local linearization
      - Differential dynamic programming
      - Feedback linearization
  - Model predictive control (MPC)
  - Examples:





Today and Thursday

- Optimal control formalism [Tedrake, Ch. 6, Sutton and Barto Ch.1-4]
- Discrete Markov decision processes (MDPs)
  - Solution through value iteration [Tedrake Ch.6, Sutton and Barto Ch.1-4]
- Solution methods for continuous problems:
  - HJB equation [[[Tedrake, Ch. 7 (optional)]]]
  - Markov chain approximation method [Chow and Tsitsiklis, 1991; Munos and Moore, 2001] [[[Kushner and Dupuis 2001 (optional)]]]
- Continuous → discrete [Chow and Tsitsiklis, 1991; Munos and Moore, 2001] [[[Kushner and Dupuis 2001 (optional)]]]
- Error bounds:
  - Value function: Chow and Tsitsiklis; Kushner and Dupuis; function approximation [Gordon 1995; Tsitsiklis and Van Roy, 1996]
  - Value function close to optimal → resulting policy good
- Speed-ups and Accuracy/Performance improvements

Optimal control formulation

Given:

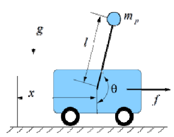
$$\text{dynamics}: \dot{x}(t) = f(x(t), u(t), t)$$

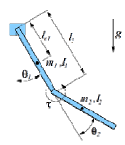
$$\text{cost function}: g(x, u, t)$$

Task: find a policy $u(t) = \pi(x, t)$ which optimizes:

$$J^{\pi}(x_0) = h(x(T)) + \int_0^T g(x(t), u(t), t) dt$$

Applicability: g and f often easier to specify than π





Finite horizon discrete time

- Markov decision process (MDP) (S, A, P, H, g)
  - S: set of states
  - A: set of actions
  - P: dynamics model $P(x_{t+1} = x' | x_t = x, u_t = u)$
  - H: horizon
  - g: S x A → R cost function
- Policy $\pi = (\mu_0, \mu_1, \ldots, \mu_H)$, $\mu_k : S \rightarrow A$
- Cost-to-go of a policy π: $J^{\pi}(x) = \mathrm{E}[\sum_{t=0}^{H} g(x(t), u(t)) | x_0 = x, \pi]$
- Goal: find $\pi^* \in \arg\min_{\pi \in \Pi} J^{\pi}$

Dynamic programming (aka value iteration)

Let $J_k^* = \min_{\mu_k,\ldots,\mu_H} \mathrm{E}[\sum_{t=k}^{H} g(x_t, u_t)]$, then we have:

$$
\begin{aligned}
J_H^*(x) &= \min_u g(x(H), u(H)) \\
J_{H-1}^*(x) &= \min_u g(x,u) + \sum_{x'} P(x'|x,u) J_H^*(x') \\
& \ldots \\
J_k^*(x) &= \min_u g(x,u) + \sum_{x'} P(x'|x,u) J_{k+1}^*(x') \\
& \ldots \\
J_0^*(x) &= \min_u g(x,u) + \sum_{x'} P(x'|x,u) J_1^*(x')
\end{aligned}
$$

And

$$\mu_k^*(x) = \arg\min_u g(x,u) + \sum_{x'} P(x'|x,u) J_{k+1}^*(x');$$

- Running time: $O(|S|^2\ |A|\ H)$ vs. naïve search over all policies would require evaluation of $|A|^{|S|H}$ policies

Discounted infinite horizon

- Markov decision process (MDP) (S, A, P, γ, g)
  - γ: discount factor
- Policy $\pi = (\mu_0, \mu_1, \ldots),\ \mu_k : S \to A$
- Value of a policy π: $J^\pi(x) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t g(x(t), u(t)) | x_0 = x, \pi]$
- Goal: find $\pi^* \in \arg\min_{\pi \in \Pi} V^\pi$

Discounted infinite horizon

- **Dynamic programming (DP) aka Value iteration (VI):**

  For i=0,1, …

  For all s ∈ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} \sum_{s'} P(s'|s,u) \left(g(s,a) + \gamma J^{(i)}(s') \right)$$

- **Facts:**

$J^{(i)} \rightarrow J^* \text{ for } i \rightarrow \infty$

There is an optimal stationary policy: $\pi^* = (\mu^*, \mu^*, \ldots)$ which satisfies:

$$\mu^*(x) = \arg\min_u g(x,u) + \gamma \sum_{x'} P(x'|x,u) J^*(x)$$

Continuous time and state-action space

- Hamilton-Jacobi-Bellman equation / approach:
  - Continuous equivalent of discrete case we already discussed

  → We will see 2 slides.
- Variational / Markov chain approximation method:
  - Numerically solve a continuous problem by directly approximating the continuous MDP with a discrete MDP

  → We will study this approach in detail.

Hamilton-Jacobi-Bellman (HJB) [*]

The Hamilton-Jacobi-Bellman Equation.
Let's develop the continuous time form of the cost-to-go function recursion by taking the limit as the time between control updates goes to zero.

$$J^*(\mathbf{x}, T) = h(\mathbf{x})$$

$$\begin{aligned} J^*(\mathbf{x}, t) &= \min_{[\mathbf{u}(t) \ldots \mathbf{u}(T)]} \left[h(\mathbf{x}(T)) + \int_t^T g(\mathbf{x}(t), \mathbf{u}(t)) dt \right], \quad \mathbf{x}(t) = \mathbf{x}, \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ &= \lim_{dt \to 0} \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) dt + J(\mathbf{x}(t + dt), t + dt) \right] \\ &\approx \lim_{dt \to 0} \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) dt + J^*(\mathbf{x}, t) + \frac{\partial J^*}{\partial \mathbf{x}} \dot{\mathbf{x}} dt + \frac{\partial J^*}{\partial t} dt \right] \end{aligned}$$

Simplifying, we are left with

$$\boxed{0 = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t} \right].} \tag{7.1}$$

This equation is well-known as the Hamilton-Jacobi-Bellman (HJB) equation.

Sufficiency theorem. The HJB equation assumes that the cost-to-go function is continuously differentiable in $\mathbf{x}$ and t, which is not necessarily the case. It therefore cannot be satisfied in all optimal control problems. It does, however, provide a sufficient condition for optimality.

56 © *Russ Tedrake, 2009*

Hamilton-Jacobi-Bellman (HJB) [*]

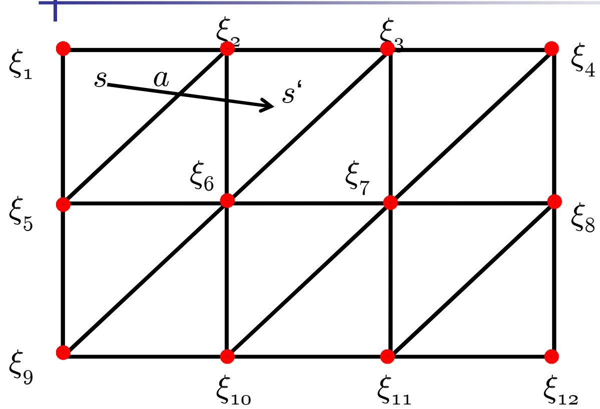
- Can also derive HJB equation for the stochastic setting. Keywords for finding out more: Controlled diffusions / diffusion jump processes.
  - For special cases, can assist in finding / verifying analytical solutions
  - However, for most cases, need to resort to numerical solution methods for the corresponding PDE --- or directly approximate the control problem with a Markov chain
- References:
  - Tedrake Ch. 7; Bertsekas, "Dynamic Programming and Optimal Control."
  - Oksendal, "Stochastic Differential Equations: An Introduction with Applications"
  - Oksendal and Sulem, "Applied Stochastic Control of Jump Diffusions"
  - Michael Steele, "Stochastic Calculus and Financial Applications"
  - Markov chain approximations: Kushner and Dupuis, 1992/2001

Markov chain approximation ("discretization")

- Original MDP (S, A, P, R, γ)

- Discretized MDP:
  - Grid the state-space: the vertices are the discrete states.
  - Reduce the action space to a finite set.
    - Sometimes not needed:
      - When Bellman back-up can be computed exactly over the continuous action space
      - When we know only certain controls are part of the optimal policy (e.g., when we know the problem has a "bang-bang" optimal solution)
  - Transition function remains to be resolved!

Discretization: example 1



Discrete states: { ξ_1 , ..., ξ_{12} }

$P(\xi_2|s,a) = p_A;$
$P(\xi_3|s,a) = p_B;$
$P(\xi_6|s,a) = p_C;$
s.t. $s' = p_A\xi_2 + p_B\xi_3 + p_C\xi_6$

- Results in discrete MDP, which we know how to solve.
- Policy when in "continuous state":

$$\pi(s) = \arg\min_a g(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_i P(\xi_i; s') J(\xi_i)$$

- Note: need not be triangular. [See also: Munos and Moore, 2001.]

Discretization: example 1 (ctd)

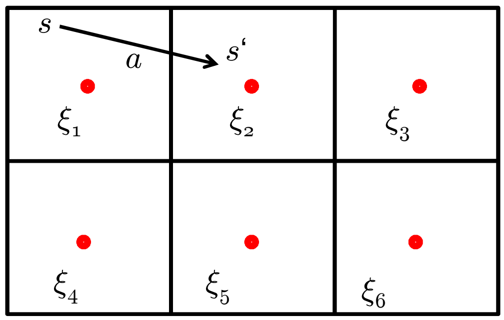
- Discretization turns deterministic transitions into stochastic transitions
- If MDP already stochastic
  - Repeat procedure to account for all possible transitions and weight accordingly
- If a (state, action) pair can results in infinitely many different next states:
  - Sample next states from the next-state distribution

Discretization: example 1 (ctd)

- Discretization results in finite state stochastic MDP, hence we know value iteration will converge
- Alternative interpretation: the Bellman back-ups in the finite state MDP are
  - (a) back-ups on a subset of the full state space
  - (b) use linear interpolation to compute the required "next-state cost-to-go functions" whenever the next state is not in the discrete set

  = value iteration with function approximation

Discretization: example 2



Discrete states: $\{ \xi_1, \ldots, \xi_6 \}$

$P(\xi_2 | s, a) = 1;$

Similarly define transition probabilities for all ξ_i

- Results in discrete MDP, which we know how to solve.
- Policy when in "continuous state":

$$\pi(s) = \arg\min_a g(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_i P(\xi_i; s') J(\xi_i)$$

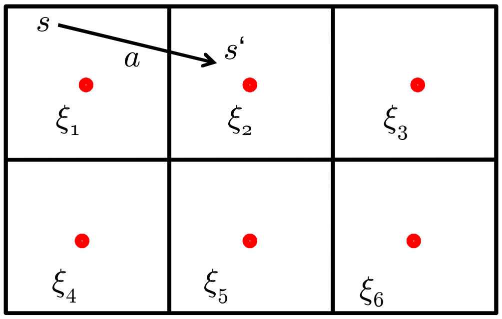
- This is nearest neighbor; could also use weighted combination of nearest neighbors.

Discretization: example 2 (ctd)

- Discretization results in finite state (stochastic) MDP, hence we know value iteration will converge
- Alternative interpretation: the Bellman back-ups in the finite state MDP are
  - (a) back-ups on a subset of the full state space
  - (b) use nearest neighbor interpolation to compute the required "next-state cost-to-go functions" whenever the next state is not in the discrete set

  = value iteration with function approximation

Discretization: example 3



Discrete states: { ξ_1 , …, ξ_6 }

$$P(\xi_i|\xi_j, u) = \frac{\int_{s \in \xi_j} P(s'|s,u) 1\{s' \in \xi_i\} ds}{\int_{s \in \xi_j} P(s'|s,u) ds}$$

After entering a region, the state gets uniformly reset to any state from that region.

[Chow and Tsitsiklis, 1991]

Discretization: example 3 (ctd)

- Discretization results in a similar MDP as for example 2
  - Main difference: transition probabilities are computed based upon a region rather than the discrete states

Continuous time

- One might want to discretize time in a variable way such that one discrete time transition roughly corresponds to a transition into neighboring grid points/regions
- Discounting: $$\exp(-\beta \delta t)$$
  δt depends on the state and action

See, e.g., Munos and Moore, 2001 for details.

Note: Numerical methods research refers to this connection between time and space as the CFL (Courant Friedrichs Levy) condition. Googling for this term will give you more background info.

!! 1 nearest neighbor tends to be especially sensitive to having the correct match [Indeed, with a mismatch between time and space 1 nearest neighbor might end up mapping many states to only transition to themselves no matter which action is taken.]

Example: Double integrator---minimum time

- Continuous time: $\ddot{q} = u, \quad \forall t : u(t) \in [-1, +1]$
- Objective: reach origin in minimum time

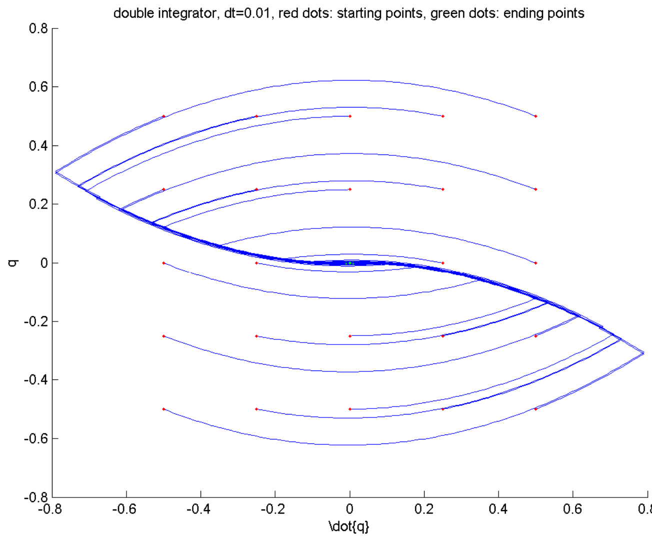
$$g(q, \dot{q}, u) = \left[\begin{array}{ll} 0 & \text{if } q = \dot{q} = 0 \\ 1 & \text{otherwise} \end{array} \right.$$

- Can be solved analytically: optimal policy is bang-bang: the control system should accelerate maximally towards the origin until a critical point at which it should hit the brakes in order to come perfectly to rest at the origin. This results in:

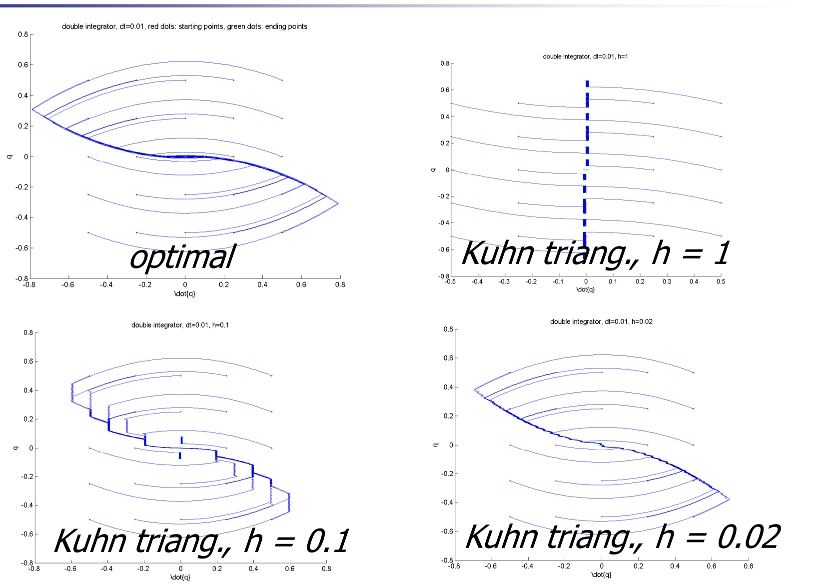
$$u = \left[\begin{array}{ll} 1 & \text{if } \dot{q} \leq -\text{sign}(q)\sqrt{2\text{sign}(q) q} \\ -1 & \text{otherwise} \end{array} \right.$$

[See Tedrake 6.6.3 for further details.]

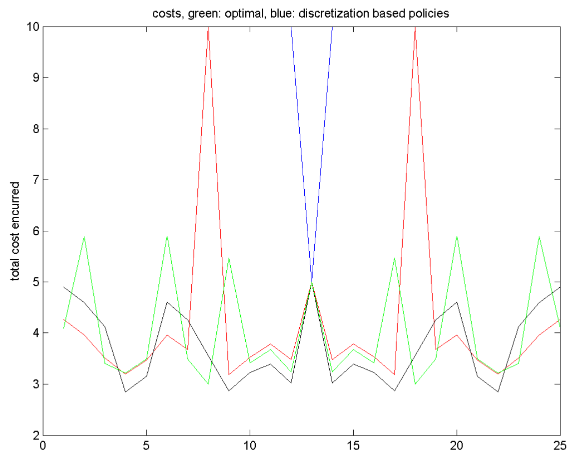
Example: Double integrator---minimum time---optimal solution



Example: Double integrator---minimum time



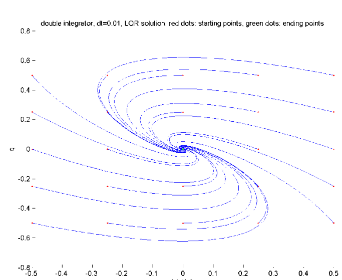
Resulting cost, Kuhn triang.



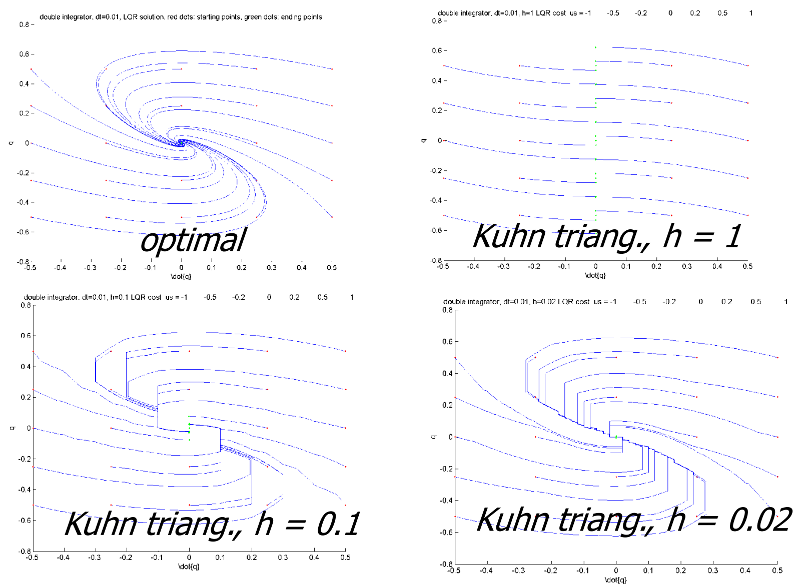
Green = continuous time optimal policy for mintime problem
For simulation we used: dt = 0.01; and goal area = within .01 of zero for q and \dot{q}.
This results in the continuous time optimal policy not being exactly optimal for the discrete time case.

Example: Double integrator---quadratic cost

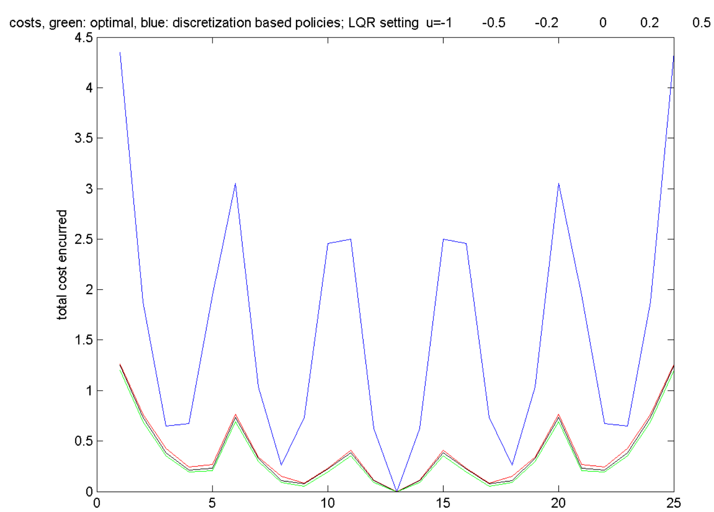
- Continuous time: $\ddot{q} = u$
- In discrete time:
$$
\begin{aligned}
q_{t+1} &= q_t + \dot{q}_t \delta t \\
\dot{q}_{t+1} &= \dot{q}_t + u \delta_t
\end{aligned}
$$
- Cost function: $g(q, \dot{q}, u) = q^2 + u^2$



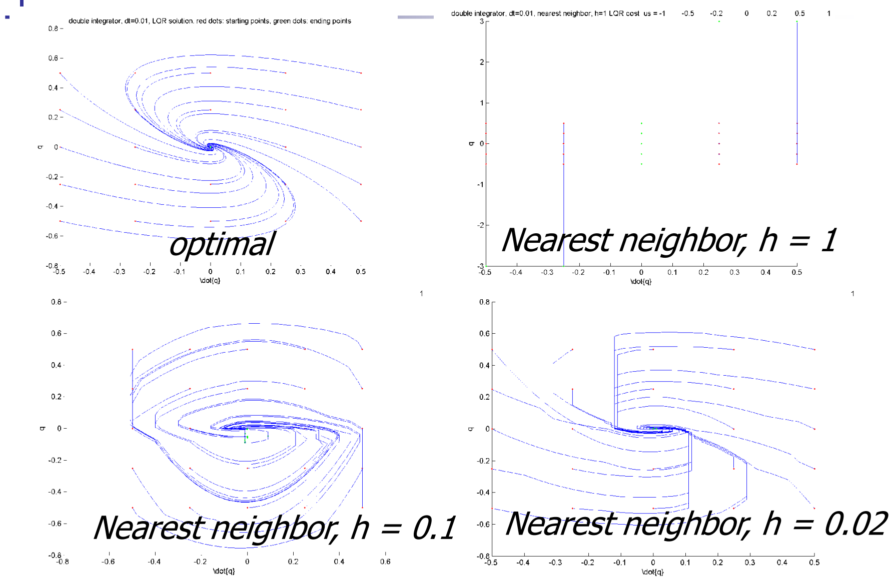
Example: Double integrator---quadratic cost



Resulting cost, Kuhn

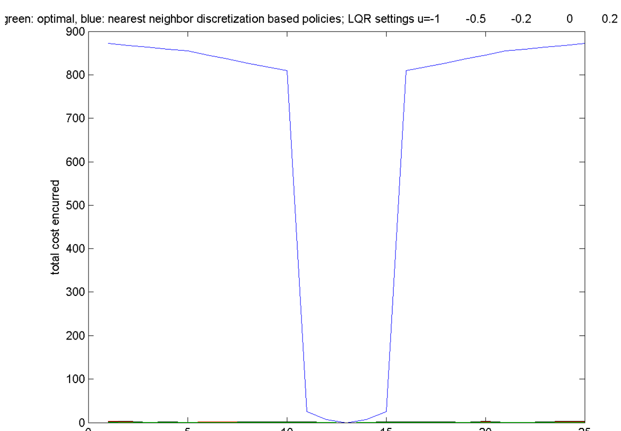


Example: Double integrator---quadratic cost

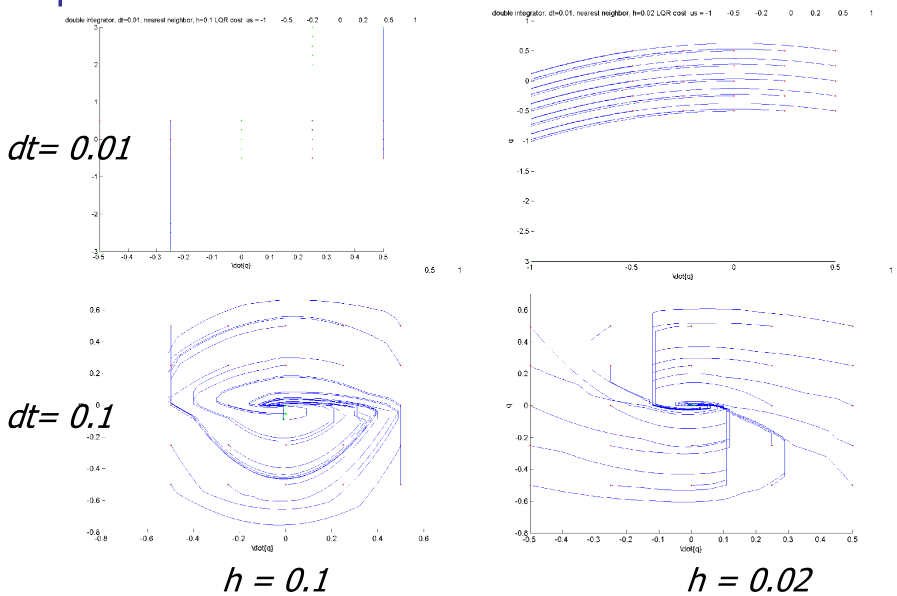


dt=0.1

Resulting cost, nearest neighbor



Nearest neighbor quickly degrades when time and space scale are mismatched



Discretization guarantees

- Typical guarantees:
  - Assume: smoothness of cost function, transition model
  - For h → 0, the discretized value function will approach the true value function
- Combine with:
  - Greedy policy w.r.t. value function V which is close to V* is a policy that attains value close to V*

Discretization proof techniques

- Chow and Tsitsiklis, 1991:
  - Show that one discretized back-up is close to one "complete" back-up + then show sequence of back-ups is also close
- Kushner and Dupuis, 2001:
  - Show that sample paths in discrete stochastic MDP approach sample paths in continuous (deterministic) MDP [also proofs for stochastic continuous, bit more complex]
- Function approximation based proof
  - Applies more generally to solving large-scale MDPs
  - Great descriptions: Gordon, 1995; Tsitsiklis and Van Roy, 1996

Example result (Chow and Tsitsiklis,1991)

A.1: $|g(x, u) - g(x', u')| \le K \|(x, u) - (x', u')\|_\infty$, for all $x, x' \in S$ and $u, u' \in C$;

A.2: $|P(y \mid x, u) - P(y' \mid x', u')| \le K \|(y, x, u) - (y', x', u')\|_\infty$, for all $x, x', y, y' \in S$ and $u, u' \in C$;

A.3: for any $x, x' \in S$ and any $u' \in U(x')$, there exists some $u \in U(x)$ such that $\|u - u'\|_\infty \le K \|x - x'\|_\infty$;

A.4: $0 \le P(y \mid x, u) \le K$ and $\int_S P(y \mid x, u)\, dy = 1$, for all $x, y \in S$ and $u \in C$.

Theorem 3.1: There exist constants K_1 and K_2 (depending only on the constant K of assumptions A.1–A.4) such that for all $h \in (0, 1/2K]$ and all $J \in \mathscr{B}(S)$

$$\|TJ - \tilde{T}_h J\|_\infty \le (K_1 + \alpha K_2 \|J\|_S) h. \qquad (3.6)$$

Furthermore,

$$\|J^* - \tilde{J}_h^*\|_\infty \le \frac{1}{1-\alpha} (K_1 + \alpha K_2 \|J^*\|_S) h. \qquad (3.7)$$

Function approximation

- General idea
  - Value iteration back-up on some states → V_{i+1}
  - Fit parameterized function to V_{i+1}

Discretization as function approximation

- Nearest neighbor discretization = piecewise constant
- Piecewise linear over “triangles” discretization

CS 287: Advanced Robotics
Fall 2009

Lecture 5: Control 4: Optimal control / Reinforcement learning--- function approximation in dynamic programming

Pieter Abbeel
UC Berkeley EECS

Today

- Optimal control/Reinforcement learning provide a general approach to tackle temporal decision making problems.
- Often the state space is too large to perform exact Dynamic Programming (DP) / Value Iteration (VI)

→ Today: Dynamic programming with function approximation

Great references:

Gordon, 1995, "Stable function approximation in dynamic programming"

Tsitsiklis and Van Roy, 1996, "Feature based methods for large scale dynamic programming"

Bertsekas and Tsitsiklis, "Neuro-dynamic programming," Chap. 6

Recall: Discounted infinite horizon

- Markov decision process (MDP) (S, A, P, γ, g)
  - γ: discount factor
- Policy $\pi = (\mu_0, \mu_1, \ldots),\ \mu_k : S \rightarrow A$
- Value of a policy π: $J^\pi(x) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t g(x(t), u(t)) | x_0 = x, \pi]$
- Goal: find $\pi^* \in \arg\min_{\pi \in \Pi} V^\pi$

Recall: Discounted infinite horizon

- **Dynamic programming (DP) aka Value iteration (VI):**

For i=0,1, …

For all s $\in$ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

- **Facts:**

$J^{(i)} \rightarrow J^*$ for $i \rightarrow \infty$

There is an optimal stationary policy: $\pi^* = (\mu^*, \mu^*, \ldots)$ which satisfies:

$$\mu^*(s) = \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^*(s)$$

- **Issue in practice: Bellman's curse of dimensionality: number of states grows exponentially in the dimensionality of the state space**

DP/VI with function approximation

Pick some $S' \subseteq S$ [typically the idea is that $|S'| << |S|$].
Iterate for $i = 0, 1, 2, \ldots$:

$$\text{back-ups:} \forall s \in S' : \bar{J}^{(i+1)}(s) \leftarrow \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) \hat{J}_{\theta^{(i)}}(s')$$

$$\text{projection: find some } \theta^{(i+1)} \text{ such that } \forall s \in S' \quad \hat{J}_{\theta^{(i+1)}}(s) \approx \bar{J}^{(i+1)}(s)$$

Projection enables generalization to $s \in S \setminus S'$, which in turn enables the Bellman back-ups in the next iteration.

θ parameterizes the class of functions used for approximation of the cost-to-go function

Function approximation examples

- Piecewise linear over triangles (tetrahedrons, etc.)
- Piecewise constant over sets of states (=nearest neighbor, often called state aggregation)
- Fit a neural net to $\bar{J}$

$$\theta^{(i+1)} = \arg\min_\theta \sum_{s \in S'} \text{loss}(\bar{J}^{(i+1)}(s) - f_\theta(s))$$

- Least squares fit to $\bar{J}$

$$\theta^{(i+1)} = \arg\min_\theta \sum_{s \in S'} (\bar{J}^{(i+1)}(s) - \phi(s)^\top \theta)^2$$

- Bezier patches (=particular choice of convex weighting)
- [[TODO: work out examples in more detail.]]

Potential guarantees?

- If we have bounded error during function approximation in each iteration, i.e.,

$$\forall i : \|\bar{J}^{(i)} - \hat{J}_{\theta^{(i)}}\| \leq \epsilon$$

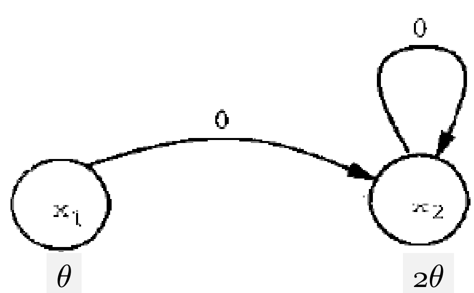
can we provide any guarantees?

- If we have a function approximation architecture that can capture the true cost-to-go function within some approximation error, i.e.,

$$\exists \theta^* : f_{\text{loss}}(J^*, \hat{J}_{\theta^*}) \leq \epsilon$$

can we provide any guarantees?

Simple example



Function approximator: [1 2] * θ

Simple example

$$\bar{J}_\theta = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \theta$$

$$\begin{aligned} \bar{J}^{(1)}(x_1) &= 0 + \gamma \hat{J}_{\theta^{(0)}}(x_2) = 2\gamma\theta^{(0)} \\ \bar{J}^{(1)}(x_2) &= 0 + \gamma \hat{J}_{\theta^{(0)}}(x_2) = 2\gamma\theta^{(0)} \end{aligned}$$

Function approximation with least squares fit:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \theta^{(1)} \approx \begin{bmatrix} 2\gamma\theta^{(0)} \\ 2\gamma\theta^{(0)} \end{bmatrix}$$

Least squares fit results in:

$$\theta^{(1)} = \frac{6}{5}\gamma\theta^{(0)}$$

Repeated back-ups and function approximations result in:

$$\theta^{(i)} = \left(\frac{6}{5}\gamma\right)^i \theta^{(0)}$$

which diverges if $\gamma > \frac{5}{6}$ even though the function approximation class can represent the true value function.]

Bellman operator

- **Dynamic programming (DP) aka Value iteration (VI):**

  Set $J^{(0)} = 0$

  For i=0,1, …

  For all s ∈ S

  $$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} \sum_{s'} P(s'|s,u) \Big(g(s,a) + \gamma J^{(i)}(s') \Big)$$

- **Bellman operator T**

  $T : \Re^{|S|} \rightarrow \Re^{|S|}$ is defined as:

  $$(TJ)(s) = \min_{u \in A} g(s,a) + \gamma \sum_{s'} P(s'|s,u) J(s')$$

- **Hence running value iteration can be compactly written as:**

  Set $J = 0$

  Repeat $J \leftarrow TJ$.

- **Note: T is a *non-linear* operator (b/c of the min).**

Contractions

Definition. The operator F is a α-contraction w.r.t. some norm $\|\cdot\|$ if

$$\forall X, \overline{X}: \quad \|FX - F\overline{X}\| \leq \alpha \|X - \overline{X}\|$$

Theorem 1. The sequence $X, FX, F^2X, ...$ converges for every X.

Theorem 2. F has a unique fixed point X^* which satisfies $FX^* = X^*$ and all sequences $X, FX, F^2X, ...$ converge to this unique fixed point X^*.

Proof of Theorem 1

Useful fact.

Cauchy sequences: If for $x_0, x_1, x_2, \ldots$, we have that

$$\forall \epsilon, \exists K : \|x_M - x_N\| < \epsilon \;\; for \;\; M, N > K$$

then we call $x_0, x_1, x_2, ...$ a Cauchy sequence.

If $x_0, x_1, x_2, \ldots$ is a Cauchy sequence, and $x_i \in \Re^n$, then there exists $x^* \in \Re^n$ such that $\lim_{i \to \infty} x_i = x^*$.

Proof.

Assume $N > M$.

$$\begin{aligned}
\|F^M X - F^N X\| &= \|\textstyle\sum_{i=M}^{N-1} (F^i X - F^{i+1} X)\| \\
&\leq \textstyle\sum_{i=M}^{N-1} \|F^i X - F^{i+1} X\| \\
&\leq \textstyle\sum_{i=M}^{N-1} \alpha^i \|X - FX\| \\
&= \|X - FX\| \textstyle\sum_{i=M}^{N-1} \alpha^i \\
&= \|X - FX\| \frac{\alpha^M}{1-\alpha}.
\end{aligned}$$

As $\|X - FX\| \frac{\alpha^M}{1-\alpha}$ goes to zero for M going to infnity, we have that for any $\epsilon > 0$ for $\|F^M X - F^N X\| \leq \epsilon$ to hold for all $M, N > K$, it suffices to pick K large enough.

Hence $X, FX, \ldots$ is a Cauchy sequence and converges.

Proof of uniqueness of fixed point

Suppose F has two fixed points. Let's say

$$\begin{aligned} FX_1 &= X_1, \\ FX_2 &= X_2, \end{aligned}$$

this implies,

$$\|FX_1 - FX_2\| = \|X_1 - X_2\|.$$

At the same time we have from the contractive property of F

$$\|FX_1 - FX_2\| \leq \alpha \|X_1 - X_2\|.$$

Combining both gives us

$$\|X_1 - X_2\| \leq \alpha \|X_1 - X_2\|.$$

Hence,

$$X_1 = X_2.$$

Therefore, the fixed point of F is unique.

The Bellman operator is a contraction

Definition. The infinity norm of a vector $x \in \Re^n$ is defined as

$$\|x\|_\infty = \max_i |x_i|$$

Fact. The Bellman operator T is a γ-contraction with respect to the infinity norm, i.e.,

$$\|TJ_1 - TJ_2\|_\infty \leq \gamma \|J_1 - J_2\|_\infty$$

Corollary. From any starting point, value iteration/Dynamic programming converges to a unique fixed point J^* which satisfies $J^* = TJ^*$.

Proof Bellman operator is a contraction

Lemma 1 *Sup-Norm Contraction*

$$||TJ - T\bar{J}||_\infty \leq \alpha ||J - \bar{J}||_\infty.$$

Note that for $y \in \mathbb{R}^{|S|}$, the sup-norm is defined as $||y||_\infty = \max_{s \in S} |y_s|$.

Proof Consider the following statement:

$$|\max_z f(z) - \max_z h(z)| \leq \max_z |f(z) - h(z)|.$$

To show that this is true:

$$\begin{aligned} \max_z f(z) - \max_z h(z) &= f(z^*) - \max_z h(z) \\ &\leq f(z^*) - h(z^*) \\ &\leq \max_z |f(z) - h(z)|. \end{aligned}$$

Applying this to $TJ(x)$ we have:

$$\begin{aligned} &|TJ(x) - T\bar{J}(x)| \\ &= |\max_{u \in U(x)} g(x,u) + \alpha \sum_{y \in S} P_{xy}(u) J(y) - \max_{u \in U(x)} g(x,u) + \alpha \sum_{y \in S} P_{xy}(u) \bar{J}(y)| \\ &\leq \max_{u \in U(x)} |g(x,u) + \alpha \sum_{y \in S} P_{xy}(u) J(y) - g(x,u) - \alpha \sum_{y \in S} P_{xy}(u) \bar{J}(y)| \\ &\leq \max_{u \in U(x)} \alpha |\sum_{y \in S} P_{xy}(u)(J(y) - \bar{J}(y))| \\ &\leq \alpha \max_{y \in S} |J(y) - \bar{J}(y)| \\ &= \alpha ||J - \bar{J}||_\infty. \end{aligned}$$

Value iteration variations

- Gauss-Seidel value iteration

For i=1, 2, …

for s=1,…,|S|

$$J(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J(s')$$

Compare to regular value iteration:

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

Exercise: Show that Gauss-Seidel value iteration converges to J*. [Hint: proceed by showing the combined operator which does the sequential update for all states s=1,…,|S| is a infinity norm contraction.]

Value iteration variations

- Asynchronous value iteration

Pick an infinite sequence of states,

$$s^{(0)}, s^{(1)}, s^{(2)}, \ldots$$

such that every state $s \in S$ occurs infinitely often. Define the operators $T_{s^{(k)}}$ as follows:

$$(T_{s^{(k)}} J)(s) = \left[\begin{array}{ll} (TJ)(s), & \text{if } \; s^{(k)} = s \\ J(s), & \text{otherwise} \end{array} \right.$$

Asynchronous value iteration initializes J and then applies, in sequence, $T_{s^{(0)}}, T_{s^{(1)}}, \ldots$.

Exercise: Show that asynchronous value iteration converges to J*.

DP/VI with function approximation

Pick some $S' \subseteq S$ [typically the idea is that $|S'| << |S|$].
Iterate for $i = 0, 1, 2, \ldots$:

$$\text{back-ups:} \forall s \in S' : \bar{J}^{(i+1)}(s) \leftarrow \min_{u \in A} \sum_{s'} P(s'|s,u) \left(g(s,u) + \gamma \hat{J}_{\theta^{(i)}}(s') \right)$$

$$\text{projection: find some } \theta^{(i+1)} \text{ such that } \forall s \in S' \;\; \hat{J}_{\theta^{(i+1)}}(s) \approx \bar{J}^{(i+1)}(s)$$

- New notation: projection operator Π maps from $\Re^n$ into the subset of $\Re^n$ which can be represented by the function approximator class

$$\bar{J}^{(i+1)} \leftarrow \Pi T \bar{J}^{(i)}$$

- While theoretical convergence analysis does not depend on this, the projection operator Π has to operate based upon only knowing J at the points s $\in$ S' , otherwise not practically feasible for large scale problems

Composing operators

- **Definition.** An operator G is a *non-expansion* with respect to a norm || . || if

$$\|GJ_1 - GJ_2\| \leq \|J_1 - J_2\|$$

- **Fact.** If the operator F is a γ contraction with respect to a norm || . || and the operator G is a non-expansion with respect to the same norm, then the sequential application of the operators G and F is a γ-contraction, i.e., $\|GFJ_1 - GFJ_2\| \leq \gamma \|J_1 - J_2\|$
- **Corollary.** The operator ΠT is a γ-contraction w.r.t. the infinity norm if Π is a non-expansion w.r.t. the infinity norm.

Averager function approximators are non-expansions

DEFINITION: A real valued function approximation scheme is an *averager* if every fitted value is the weighted average of zero or more target values and possibly some predetermined constants. The weights involved in calculating the fitted value $\hat{Y}_i$ may depend on the sample vector X_0, but may not depend on the target values Y. More precisely, for a fixed X_0, if Y has n elements, there must exist n real numbers k_i, n^2 nonnegative real numbers β_{ij}, and n nonnegative real numbers β_i, so that for each i we have $\beta_i + \sum_j \beta_{ij} = 1$ and $\hat{Y}_i = \beta_i k_i + \sum_j \beta_{ij} Y_j$.

- Examples:
  - nearest neighbor (aka state aggregation)
  - linear interpolation over triangles (tetrahedrons, …)

Averager function approximators are non-expansions

Theorem. The mapping Π associated with any averaging method is a nonexpansion in the infinity norm.

Proof: Let J_1 and J_2 be two vectors in $\Re^n$. Consider a particular entry s of ΠJ_1 and ΠJ_2:

$$
\begin{aligned}
|(\Pi J_1)(s) - (\Pi J_2)(s)| &= |\beta_{s0} + \sum_{s'} \beta_{ss'} J_1(s') - \beta_{s0} + \sum_{s'} \beta_{ss'} J_2(s')| \\
&= |\sum_{s'} \beta_{ss'} (J_1(s') - J_2(s'))| \\
&\leq \max_{s'} |J_1(s') - J_2(s')| \\
&= \|J_1 - J_2\|_\infty
\end{aligned}
$$

This holds true for all s, hence we have

$$\|\Pi J_1 - \Pi J_2\|_\infty \leq \|J_1 - J_2\|_\infty$$

Linear regression ☹

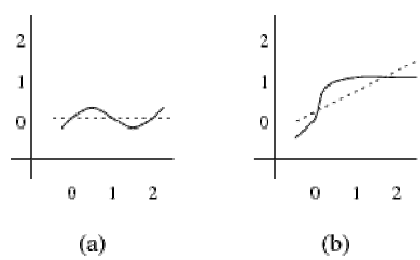


Figure 2: The mapping associated with linear regression when samples are taken at the points $x = 0, 1, 2$. In (a) we see a target value function (solid line) and its corresponding fitted value function (dotted line). In (b) we see another target function and another fitted function. The first target function has values $y = 0, 0, 0$ at the sample points; the second has values $y = 0, 1, 1$. Regression exaggerates the difference between the two functions: the largest difference between the two target functions at a sample point is 1 (at $x = 1$ and $x = 2$), but the largest difference between the two fitted functions at a sample point is $\frac{7}{6}$ (at $x = 2$).

[Example taken from Gordon, 1995.]

Guarantees for fixed point

Theorem. Let J^* be the optimal value function for a finite MDP with discount factor γ. Let the projection operator Π be a non-expansion w.r.t. the infinity norm and let $\tilde{J}$ be any fixed point of Π. Suppose $\|\tilde{J} - J^*\|_\infty \leq \epsilon$. Then ΠT converges to a value function $\bar{J}$ such that:

$$\|\bar{J} - J^*\| \leq 2\epsilon + \frac{2\gamma\epsilon}{1-\gamma}$$

- I.e., if we pick a non-expansion function approximator which can approximate J* well, then we obtain a good value function estimate.
- To apply to discretization: use continuity assumptions to show that J* can be approximated well by chosen discretization scheme

CS 287: Advanced Robotics
Fall 2009

Lecture 5: Control 4: Optimal control / Reinforcement learning--- function approximation in dynamic programming

Pieter Abbeel
UC Berkeley EECS

Today

- Recap + continuation of value iteration with function approximation
- Performance boosts
- Speed-ups
- Intermezzo: Extremely crude outline of (part of) the reinforcement learning field [as it might assist when reading some of the references]

Great references:

Gordon, 1995, "Stable function approximation in dynamic programming"

Tsitsiklis and Van Roy, 1996, "Feature based methods for large scale dynamic programming"

Bertsekas and Tsitsiklis, "Neuro-dynamic programming," Chap. 6

Recall: Discounted infinite horizon

- Markov decision process (MDP) (S, A, P, γ, g)
  - γ: discount factor
- Policy $\pi = (\mu_0, \mu_1, \ldots),\ \mu_k : S \rightarrow A$
- Value of a policy π: $J^\pi(x) = \mathrm{E}[\sum_{t=0}^\infty \gamma^t g(x(t), u(t)) | x_0 = x, \pi]$
- Goal: find $\pi^* \in \arg\min_{\pi \in \Pi} J^\pi$

Recall: Discounted infinite horizon

- **Dynamic programming (DP) aka Value iteration (VI):**

  For i=0,1, …

  For all s $\in$ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

- **Facts:**

  $J^{(i)} \rightarrow J^*$ for $i \rightarrow \infty$

  There is an optimal stationary policy: $\pi^* = (\mu^*, \mu^*, \ldots)$ which satisfies:

$$\mu^*(s) = \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^*(s)$$

- **Issue in practice: Bellman's curse of dimensionality: number of states grows exponentially in the dimensionality of the state space**

DP/VI with function approximation

Pick some $S' \subseteq S$ [typically the idea is that $|S'| << |S|$].
Iterate for $i = 0, 1, 2, \ldots$:

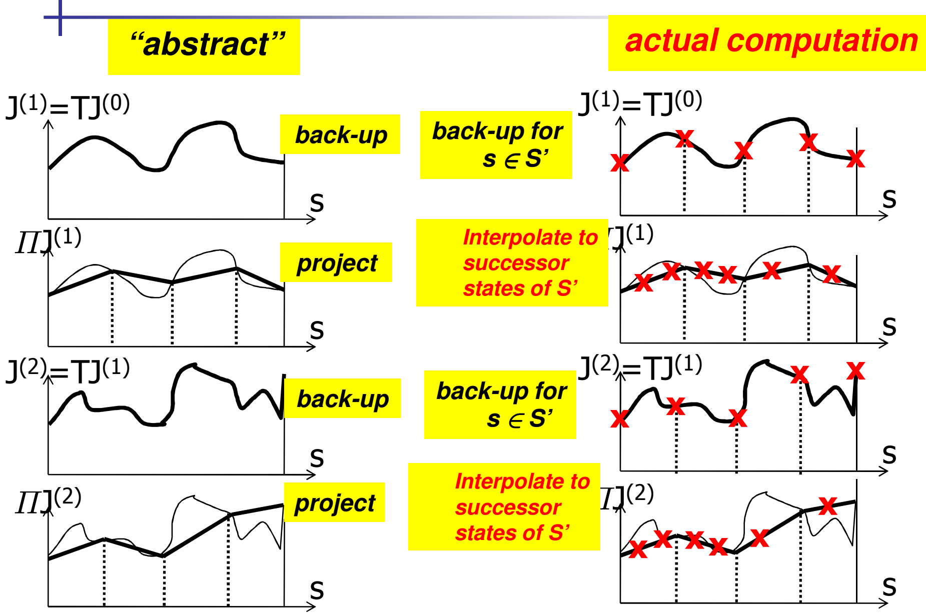
$$\text{back-ups:} \forall s \in S' : \bar{J}^{(i+1)}(s) \leftarrow \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) \hat{J}_{\theta^{(i)}}(s')$$

$$\text{projection: find some } \theta^{(i+1)} \text{ such that } \forall s \in S' \quad \hat{J}_{\theta^{(i+1)}}(s) = (\Pi \bar{J}^{(i+1)})(s) \approx \bar{J}^{(i+1)}(s)$$

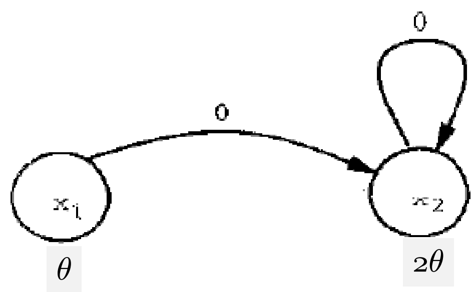
Projection enables generalization to $s \in S \setminus S'$, which in turn enables the Bellman back-ups in the next iteration.

θ parameterizes the class of functions used for approximation of the cost-to-go function

Example --- piecewise linear



Recall: VI with function approximation need not converge!



P(x2|x1,u) = 1; P(x2|x2,u) = 1
g(x1,u) = 0; g(x2,u) = 0;

Function approximator: [1 2] * θ

VI w/ least squares function approximation diverges for γ > 5/6 [see last lecture for details]

Contractions

- **Fact.** *The Bellman operator, T, is a γ-contraction w.r.t. the infinity norm, i.e.,*

$$\forall J_1, J_2 : \|TJ_1 - TJ_2\|_\infty \leq \gamma \|J_1 - J_2\|_\infty$$

- **Theorem.** *The Bellman operator has a unique fixed point J* = TJ* and for all J we have that $T^{(k)}$J converges to J* for k going to infinity.*

- Note:

$$\begin{aligned} \|T^{(k)}J - J^*\|_\infty &= \|T^{(k)}J - T^{(k)}J^*\|_\infty \\ &\leq \gamma\|T^{(k-1)}J - T^{(k-1)}J^*\|_\infty \\ &\leq \gamma^k\|J - J^*\|_\infty \end{aligned}$$

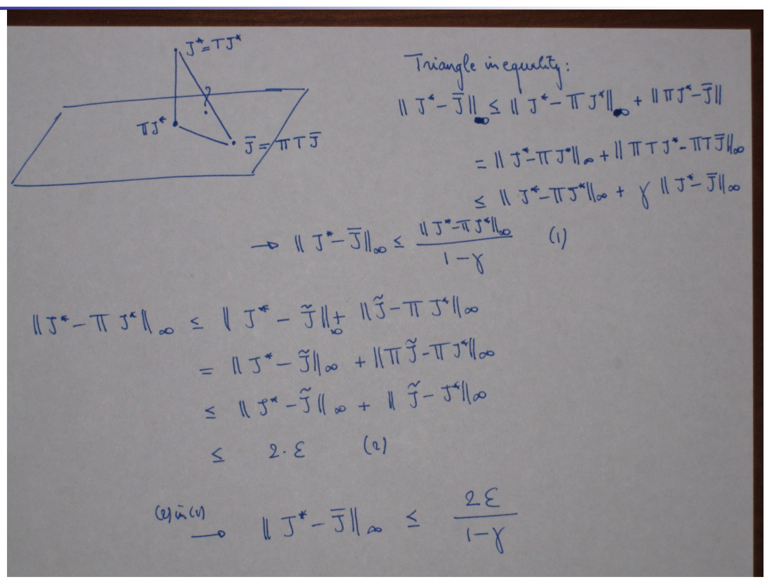
I.e., with every back-up, the infinity norm distance to J^* decreases.

Guarantees for fixed point

Theorem. Let J^* be the optimal value function for a finite MDP with discount factor γ. Let the projection operator Π be a non-expansion w.r.t. the infinity norm and let $\tilde{J}$ be any fixed point of Π. Suppose $\|\tilde{J} - J^*\|_\infty \leq \epsilon$. Then ΠT converges to a value function $\bar{J}$ such that:

$$\|\bar{J} - J^*\| \leq \frac{2\epsilon}{1-\gamma}$$

Proof



[See also Gordon 1995]

Can we generally verify goodness of some estimate J despite not having access to J*

Fact. Assume we have some $\hat{J}$ for which we have that $\|\hat{J} - T\hat{J}\|_\infty \leq \epsilon$. Then we have that $\|\hat{J} - J^*\|_\infty \leq \frac{\epsilon}{1-\gamma}$.

Proof:

$$\begin{aligned}
\|\hat{J} - J^*\|_\infty &= \|\hat{J} - T\hat{J} + T\hat{J} - T^2\hat{J} + T^2\hat{J} - T^3\hat{J} + \ldots - J^*\|_\infty \\
&\leq \|\hat{J} - T\hat{J}\|_\infty + \|T\hat{J} - T^2\hat{J}\|_\infty + \|T^2\hat{J} - T^3\hat{J}\|_\infty + \ldots + \|T^\infty\hat{J} - J^*\|_\infty \\
&\leq \epsilon + \gamma\epsilon + \gamma^2\epsilon + \ldots
\end{aligned}$$

$$= \frac{\epsilon}{1-\gamma}$$

- Of course, in most (perhaps all) large scale settings in which function approximation is desirable, it will be hard to compute the bound on the infinity norm …

What if the projection fails to be a non-expansion

- Assume Π only introduces a little bit of noise, i.e.,

$$\forall \text{ iterations } i: \|T\bar{J}^{(i)} - \Pi T\bar{J}^{(i)}\|_\infty \leq \epsilon$$

Or, more generally, we have a noisy sequence of back-ups:

$$J^{(i+1)} \leftarrow TJ^{(i)} + w^{(i)} \text{ with the noise } w^{(i)} \text{ satisfying: } \|w^{(i)}\|_\infty \leq \epsilon$$

Fact. $\|J^{(i)} - T^iJ\| \leq \epsilon(1+\gamma+\ldots+\gamma^{i-1})$ and as a consequence $\limsup_{i\rightarrow\infty}\|J^{(i)} - J^*\| \leq \frac{\epsilon}{1-\gamma}$.

Proof by induction:
Base case: We have $\|J^{(1)} - TJ^{(0)}\|_\infty \leq \epsilon$.
Induction: We also have for any $i > 1$:

$$\begin{aligned}
\|T^iJ^{(0)} - J^{(i)}\|_\infty &= \|TT^{i-1}J^{(0)} - TJ^{(i-1)} - w^{(i-1)}\|_\infty \\
&\leq \epsilon + \gamma\|T^{i-1}J^{(0)} - J^{(i-1)}\|_\infty \\
&\leq \epsilon + \gamma(\epsilon(1 + \gamma + \gamma^2 + \ldots + \gamma^{(i-2)}))
\end{aligned}$$

Guarantees for greedy policy w.r.t. approximate value function

Definition. μ is the greedy policy w.r.t. J if for all states s:

$$\mu(s) \in \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u)J(s')$$

Fact. Suppose that J satisfies $\|J - J^*\|_\infty \leq \epsilon$. If μ is a greedy policy based on J, then

$$\|J^\mu - J^*\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma}$$

Here $J^\mu = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t g(s_t, \mu(s_t))]$.

[See also Bertsekas and Ttsitsiklis, 6.1.1]

Proof

Recall:

$$(TJ)(s) = \min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u)J(s')$$

Similarly define:

$$(T_\mu J)(s) = g(s,\mu(s)) + \gamma \sum_{s'} P(s'|s,\mu(s))J(s')$$

We have $TJ^* = J^*$ and (same result for MDP with only 1 policy available) $T_\mu J^\mu = J^\mu$.

A very typical proof follows, with the main ingredients adding and subtracting the same terms to make terms pairwise easier to compare/bound:

$$\begin{aligned}
\|J^\mu - J^*\|_\infty &= \|T_\mu J^\mu - J^*\|_\infty \\
&\leq \|T_\mu J^\mu - T_\mu J\|\infty + \|T_\mu J - J^*\|_\infty \\
&\leq \gamma\|J^\mu - J\|_\infty + \|TJ - J*\|_\infty \\
&\leq \gamma\|J^\mu - J^*\|_\infty + \gamma\|J^* - J\|_\infty + \gamma\|J - J^*\|_\infty \\
&= \gamma\|J^\mu - J^*\|_\infty + 2\gamma\epsilon,
\end{aligned}$$

and the result follows.

Recap function approximation

- DP/VI with function approximation:
  - Iterate: $J \leftarrow \Pi T J$
- Need not converge!
- Guarantees when:
  - The projection is an infinity norm non-expansion
  - Bounded error in each projection/function approximation step
- In later lectures we will also study the policy iteration and linear programming approaches

Reinforcement learning---very crude map

- Exact methods w/full model available (e.g. Value iteration/DP, policy iteration, LP)
- Approximate DP w/model available
- Sample states:
  - Use all sampled data in batch → often reducible to "exact methods" on an approximate transition model
  - Use incremental updates → stochastic approximation techniques might prove convergence to desired solution

Improving performance with a given value function

1. Multi-stage lookahead aka Receding/Moving horizon

- Rather than using greedy policy μ w.r.t. approximate value function, with

$$\mu(s_t) = \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u)\hat{J}_\theta(s')$$

- Two-stage lookahead:
  - At time t perform back-ups for all s' which are successor states of s_t
  - Then use these backed up values to perform the back-up for s_t
- N stage lookahead: similarly,perform back-ups to N-stages of successor states of s_t backward in time
- Can't guarantee N-stage lookahead provides better performance [Can guarantee tighter infinity norm bound on attained value function estimates by N-stage lookahead.]
- Example application areas in which it has improved performance chess, backgammon

See also Bertsekas and Tsitsiklis, 6.1.2

Improving performance with a given value function

2. Roll-out policies

- Given a policy π, choose the current action u by evaluating the cost encurred by taking action u followed by executing the policy π from then onwards
- Guaranteed to perform better than the baseline policy on top of which it builds (thanks to general guarantees of policy iteration algorithm)
- Baseline policy could be obtained with any method
- Practicalities
  - Todo --- fill in

See also Bertsekas and Tsitsiklis, 6.1.3

Speed-ups

- Parallelization
  - VI lends itself to parallellization
- Multi-grid, Coarse-to-fine grid, Variable resolution grid
- Prioritized sweeping
- Richardson extrapolation
- Kuhn triangulation

Prioritized sweeping

- **Dynamic programming (DP) / Value iteration (VI):**

  For i=0,1, …

  For all s $\in$ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

- Prioritized sweeping idea: focus updates on states for which the update is expected to be most significant
- Place states into priority queue and perform updates accordingly
  - For every Bellman update: compute the difference J^{(i+1)} – J^{(i)}
  - Then update the priority of the states s' from which one could transition into s based upon the above difference and the transition probability of transitioning into s'
- For details: See Moore and Atkeson, 1993, "Prioritized sweeping: RL with less data and less real time"

Richardson extrapolation

- Generic method to improve the rate of convergence of a sequence
- Assume h is the grid-size parameter in a discretization scheme
- Assume we can approximate $J^{(h)}(x)$ as follows:

$$J^{(h)}(x) = J(x) + J_1(x)h + o(h)$$

- Similarly:

$$J^{(h/2)}(x) = J(x) + J_1(x)h/2 + o(h)$$

- Then we can get rid of the order h error term by using the following approximation which combines both:

$$2J^{(h/2)}(x) - J^{(h)}(x) = J(x) + o(h)$$

Kuhn triangulation

- Allows efficient computation of the vertices participating in a point's barycentric coordinate system and of the convex interpolation weights (aka the barycentric coordinates)

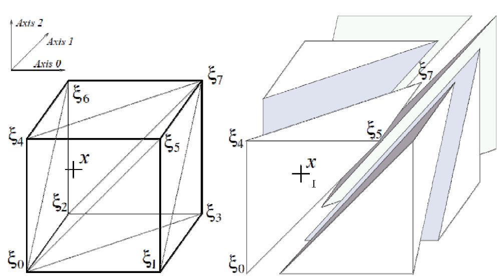


Figure 2. The Kuhn triangulation of a (3d) rectangle. The point x satisfying $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ is in the simplex $(\xi_0, \xi_4, \xi_5, \xi_7)$.

- See Munos and Moore, 2001 for further details.

Kuhn triangulation (from Munos and Moore)

3.1. Computational issues

Although the number of simplexes inside a rectangle is factorial with the dimension d, the computation time for interpolating the value at any point inside a rectangle is only of order $(d \ln d)$, which corresponds to a sorting of the d relative coordinates $(x_0, ..., x_{d-1})$ of the point inside the rectangle.

Assume we want to compute the indexes $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing a point defined by its relative coordinates $(x_0, ..., x_{d-1})$ with respect to the rectangle in which it belongs to. Let $\{\xi_0, ..., \xi_{2^d}\}$ be the corners of this d-rectangle. The indexes of the corners use the binary decomposition in dimension d, as illustrated in Figure 2. Computing these indexes is achieved by sorting the coordinates from the highest to the smallest: there exist indices $j_0, ..., j_{d-1}$, permutation of $\{0, .., d-1\}$, such that $1 \geq x_{j_0} \geq x_{j_1} \geq ... \geq x_{j_{d-1}} \geq 0$. Then the indices $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing the point are: $i_0 = 0$, $i_1 = i_0 + 2^{j_0}$, ..., $i_k = i_{k-1} + 2^{j_{k-1}}$, ..., $i_d = i_{d-1} + 2^{j_{d-1}} = 2^d - 1$. For example, if the coordinates satisfy: $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ (illustrated by the point x in Figure 2) then the vertices are: ξ_0 (every simplex contains this vertex, as well as $\xi_{2^d-1} = \xi_7$), ξ_4 (we added 2^2), ξ_5 (we added 2^0) and ξ_7 (we added 2^1).

Let us define the *barycentric coordinates* $\lambda_0, ..., \lambda_d$ of the point x inside the simplex $\xi_{i_0}, ..., \xi_{i_d}$ as the positive coefficients (uniquely) defined by: $\sum_{k=0}^{d} \lambda_k = 1$ and $\sum_{k=0}^{d} \lambda_k \xi_{i_k} = x$. Usually, these barycentric coordinates are expensive to compute; however, in the case of Kuhn triangulation these coefficients are simply: $\lambda_0 = 1 - x_{j_0}$, $\lambda_1 = x_{j_0} - x_{j_1}$, ..., $\lambda_k = x_{j_{k-1}} - x_{j_k}$, ..., $\lambda_d = x_{j_{d-1}} - 0 = x_{j_{d-1}}$. In the previous example, the barycentric coordinates are: $\lambda_0 = 1 - x_2$, $\lambda_1 = x_2 - x_0$, $\lambda_2 = x_0 - x_1$, $\lambda_3 = x_1$.

CS 287: Advanced Robotics
Fall 2009

Lecture 5: Control 4: Optimal control / Reinforcement learning--- function approximation in dynamic programming

Pieter Abbeel
UC Berkeley EECS

Today

- Recap + continuation of value iteration with function approximation
- Performance boosts
- Speed-ups
- Intermezzo: Extremely crude outline of (part of) the reinforcement learning field [as it might assist when reading some of the references]

Great references:

Gordon, 1995, “Stable function approximation in dynamic programming”

Tsitsiklis and Van Roy, 1996, “Feature based methods for large scale dynamic programming”

Bertsekas and Tsitsiklis, “Neuro-dynamic programming,” Chap. 6

Recall: Discounted infinite horizon

- Markov decision process (MDP) (S, A, P, γ, g)
  - γ: discount factor
- Policy $\pi = (\mu_0, \mu_1, \ldots),\ \mu_k : S \rightarrow A$
- Value of a policy π: $J^{\pi}(x) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t g(x(t), u(t)) | x_0 = x, \pi]$
- Goal: find $\pi^* \in \arg\min_{\pi \in \Pi} J^{\pi}$

Recall: Discounted infinite horizon

- **Dynamic programming (DP) aka Value iteration (VI):**

For i=0,1, …

For all s $\in$ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

- **Facts:**

$J^{(i)} \rightarrow J^*$ for $i \rightarrow \infty$

There is an optimal stationary policy: $\pi^* = (\mu^*, \mu^*, \ldots)$ which satisfies:

$$\mu^*(s) = \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^*(s)$$

- **Issue in practice: Bellman's curse of dimensionality: number of states grows exponentially in the dimensionality of the state space**

DP/VI with function approximation

Pick some $S' \subseteq S$ [typically the idea is that $|S'| << |S|$].
Iterate for $i = 0, 1, 2, \ldots$:

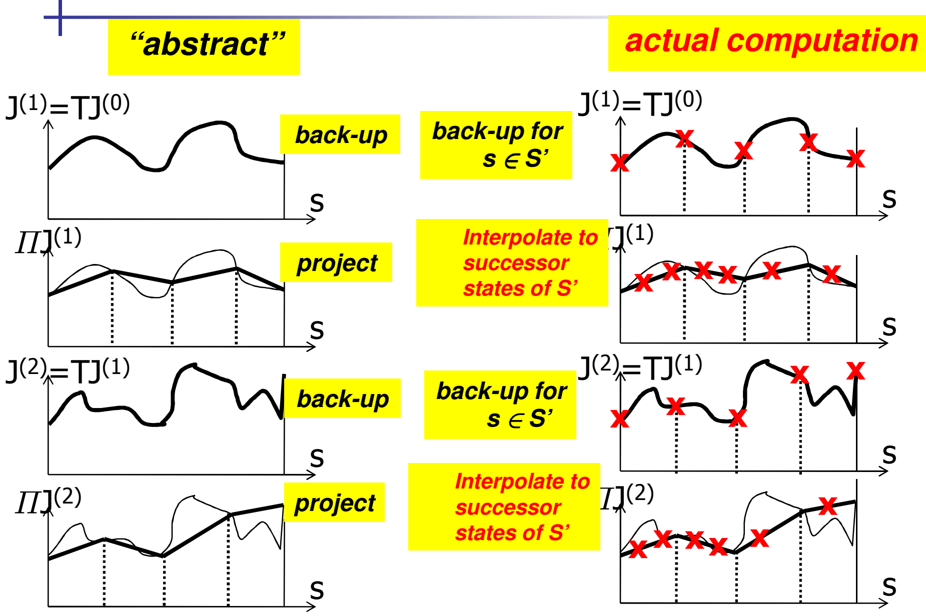
$$\text{back-ups:} \forall s \in S' : \bar{J}^{(i+1)}(s) \leftarrow \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) \hat{J}_{\theta^{(i)}}(s')$$

$$\text{projection: find some } \theta^{(i+1)} \text{ such that } \forall s \in S' \quad \hat{J}_{\theta^{(i+1)}}(s) = (\Pi \bar{J}^{(i+1)})(s) \approx \bar{J}^{(i+1)}(s)$$

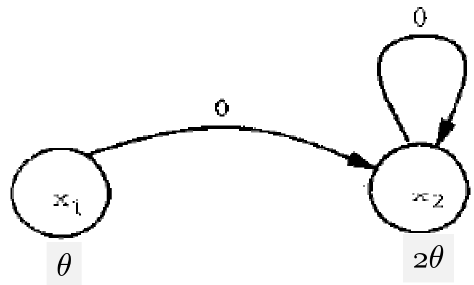
Projection enables generalization to $s \in S \setminus S'$, which in turn enables the Bellman back-ups in the next iteration.

θ parameterizes the class of functions used for approximation of the cost-to-go function

Example --- piecewise linear



Recall: VI with function approximation need not converge!



P(x2|x1,u) = 1; P(x2|x2,u) = 1
g(x1,u) = 0; g(x2,u) = 0;

Function approximator: [1 2] * θ

VI w/ least squares function approximation diverges for γ > 5/6 [see last lecture for details]

Contractions

- **Fact.** *The Bellman operator, T, is a γ-contraction w.r.t. the infinity norm, i.e.,*

$$\forall J_1, J_2 : \|TJ_1 - TJ_2\|_\infty \leq \gamma \|J_1 - J_2\|_\infty$$

- **Theorem.** *The Bellman operator has a unique fixed point J* = TJ* and for all J we have that $T^{(k)}$J converges to J* for k going to infinity.*

- Note:

$$\begin{aligned} \|T^{(k)}J - J^*\|_\infty &= \|T^{(k)}J - T^{(k)}J^*\|_\infty \\ &\leq \gamma \|T^{(k-1)}J - T^{(k-1)}J^*\|_\infty \\ &\leq \gamma^k \|J - J^*\|_\infty \end{aligned}$$

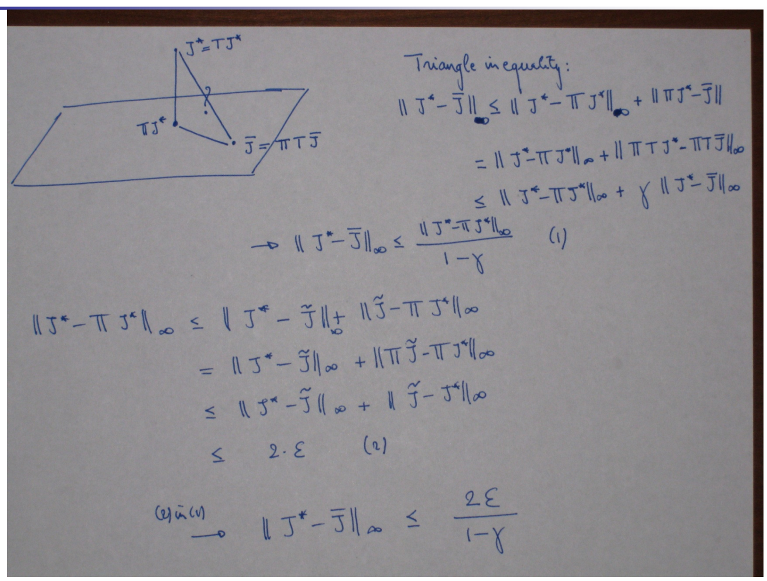
I.e., with every back-up, the infinity norm distance to J^* decreases.

Guarantees for fixed point

Theorem. Let J^* be the optimal value function for a finite MDP with discount factor γ. Let the projection operator Π be a non-expansion w.r.t. the infinity norm and let $\tilde{J}$ be any fixed point of Π. Suppose $\|\tilde{J} - J^*\|_\infty \leq \epsilon$. Then ΠT converges to a value function $\bar{J}$ such that:

$$\|\bar{J} - J^*\| \leq \frac{2\epsilon}{1-\gamma}$$

Proof



[See also Gordon 1995]

Can we generally verify goodness of some estimate J despite not having access to J*

Fact. Assume we have some $\hat{J}$ for which we have that $\|\hat{J} - T\hat{J}\|_\infty \leq \epsilon$. Then we have that $\|\hat{J} - J^*\|_\infty \leq \frac{\epsilon}{1-\gamma}$.
Proof:

$$
\begin{aligned}
\|\hat{J} - J^*\|_\infty &= \|\hat{J} - T\hat{J} + T\hat{J} - T^2\hat{J} + T^2\hat{J} - T^3\hat{J} + \ldots - J^*\|_\infty \\
&\leq \|\hat{J} - T\hat{J}\|_\infty + \|T\hat{J} - T^2\hat{J}\|_\infty + \|T^2\hat{J} - T^3\hat{J}\|_\infty + \ldots + \|T^\infty\hat{J} - J^*\|_\infty \\
&\leq \epsilon + \gamma\epsilon + \gamma^2\epsilon + \ldots
\end{aligned}
$$

$$= \frac{\epsilon}{1-\gamma}$$

- Of course, in most (perhaps all) large scale settings in which function approximation is desirable, it will be hard to compute the bound on the infinity norm …

What if the projection fails to be a non-expansion

- Assume Π only introduces a little bit of noise, i.e.,

$$\forall \text{ iterations } i : \|T\bar{J}^{(i)} - \Pi T\bar{J}^{(i)}\|_\infty \leq \epsilon$$

Or, more generally, we have a noisy sequence of back-ups:

$$J^{(i+1)} \leftarrow TJ^{(i)} + w^{(i)} \text{ with the noise } w^{(i)} \text{ satisfying: } \|w^{(i)}\|_\infty \leq \epsilon$$

Fact. $\|J^{(i)} - T^iJ\| \leq \epsilon(1+\gamma+\ldots+\gamma^{i-1})$ and as a consequence $\limsup_{i\to\infty} \|J^{(i)} - J^*\| \leq \frac{\epsilon}{1-\gamma}$.

Proof by induction:
Base case: We have $\|J^{(1)} - TJ^{(0)}\|_\infty \leq \epsilon$.
Induction: We also have for any $i > 1$:

$$
\begin{aligned}
\|T^iJ^{(0)} - J^{(i)}\|_\infty &= \|TT^{i-1}J^{(0)} - TJ^{(i-1)} - w^{(i-1)}\|_\infty \\
&\leq \epsilon + \gamma\|T^{i-1}J^{(0)} - J^{(i-1)}\|_\infty \\
&\leq \epsilon + \gamma(\epsilon(1 + \gamma + \gamma^2 + \ldots + \gamma^{(i-2)}))
\end{aligned}
$$

Guarantees for greedy policy w.r.t. approximate value function

Definition. μ is the greedy policy w.r.t. J if for all states s:

$$\mu(s) \in \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J(s')$$

Fact. Suppose that J satisfies $\|J - J^*\|_\infty \leq \epsilon$. If μ is a greedy policy based on J, then

$$\|J^\mu - J^*\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma}$$

Here $J^\mu = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t g(s_t, \mu(s_t))]$.

[See also Bertsekas and Ttsitsiklis, 6.1.1]

Proof

Recall:

$$(TJ)(s) = \min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J(s')$$

Similarly define:

$$(T_\mu J)(s) = g(s,\mu(s)) + \gamma \sum_{s'} P(s'|s,\mu(s)) J(s')$$

We have $TJ^* = J^*$ and (same result for MDP with only 1 policy available) $T_\mu J^\mu = J^\mu$.

A very typical proof follows, with the main ingredients adding and subtracting the same terms to make terms pairwise easier to compare/bound:

$$\begin{aligned}
\|J^\mu - J^*\|_\infty &= \|T_\mu J^\mu - J^*\|_\infty \\
&\leq \|T_\mu J^\mu - T_\mu J\|\infty + \|T_\mu J - J^*\|_\infty \\
&\leq \gamma\|J^\mu - J\|_\infty + \|TJ - J*\|_\infty \\
&\leq \gamma\|J^\mu - J^*\|_\infty + \gamma\|J^* - J\|_\infty + \gamma\|J - J^*\|_\infty \\
&= \gamma\|J^\mu - J^*\|_\infty + 2\gamma\epsilon,
\end{aligned}$$

and the result follows.

Recap function approximation

- DP/VI with function approximation:
  - Iterate: $J \leftarrow \Pi T J$
- Need not converge!
- Guarantees when:
  - The projection is an infinity norm non-expansion
  - Bounded error in each projection/function approximation step
- In later lectures we will also study the policy iteration and linear programming approaches

Reinforcement learning---very crude map

- Exact methods w/full model available (e.g. Value iteration/DP, policy iteration, LP)
- Approximate DP w/model available
- Sample states:
  - Use all sampled data in batch → often reducible to "exact methods" on an approximate transition model
  - Use incremental updates → stochastic approximation techniques might prove convergence to desired solution

Improving performance with a given value function

1. **Multi-stage lookahead aka Receding/Moving horizon**
   - Rather than using greedy policy μ w.r.t. approximate value function, with

     $$\mu(s_t) = \arg\min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u)\hat{J}_\theta(s')$$
   - Two-stage lookahead:
     - At time t perform back-ups for all s' which are successor states of s_t
     - Then use these backed up values to perform the back-up for s_t
   - N stage lookahead: similarly,perform back-ups to N-stages of successor states of s_t backward in time
   - Can't guarantee N-stage lookahead provides better performance [Can guarantee tighter infinity norm bound on attained value function estimates by N-stage lookahead.]
   - Example application areas in which it has improved performance chess, backgammon

See also Bertsekas and Tsitsiklis, 6.1.2

Improving performance with a given value function

2. Roll-out policies

- Given a policy π, choose the current action u by evaluating the cost encurred by taking action u followed by executing the policy π from then onwards
- Guaranteed to perform better than the baseline policy on top of which it builds (thanks to general guarantees of policy iteration algorithm)
- Baseline policy could be obtained with any method
- Practicalities
  - Todo --- fill in

See also Bertsekas and Tsitsiklis, 6.1.3

Speed-ups

- Parallelization
  - VI lends itself to parallellization
- Multi-grid, Coarse-to-fine grid, Variable resolution grid
- Prioritized sweeping
- Richardson extrapolation
- Kuhn triangulation

Prioritized sweeping

- **Dynamic programming (DP) / Value iteration (VI):**

  For i=0,1, …

  For all s $\in$ S

$$J^{(i+1)}(s) \quad \leftarrow \quad \min_{u \in A} g(s,u) + \gamma \sum_{s'} P(s'|s,u) J^{(i)}(s')$$

- Prioritized sweeping idea: focus updates on states for which the update is expected to be most significant
- Place states into priority queue and perform updates accordingly
  - For every Bellman update: compute the difference J^{(i+1)} – J^{(i)}
  - Then update the priority of the states s' from which one could transition into s based upon the above difference and the transition probability of transitioning into s'
- For details: See Moore and Atkeson, 1993, "Prioritized sweeping: RL with less data and less real time"

Richardson extrapolation

- Generic method to improve the rate of convergence of a sequence
- Assume h is the grid-size parameter in a discretization scheme
- Assume we can approximate $J^{(h)}(x)$ as follows:

$$J^{(h)}(x) = J(x) + J_1(x)h + o(h)$$

- Similarly:

$$J^{(h/2)}(x) = J(x) + J_1(x)h/2 + o(h)$$

- Then we can get rid of the order h error term by using the following approximation which combines both:

$$2J^{(h/2)}(x) - J^{(h)}(x) = J(x) + o(h)$$

Kuhn triangulation

- Allows efficient computation of the vertices participating in a point's barycentric coordinate system and of the convex interpolation weights (aka the barycentric coordinates)

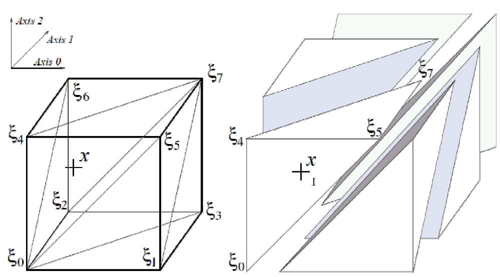


Figure 2. The Kuhn triangulation of a (3d) rectangle. The point x satisfying $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ is in the simplex $(\xi_0, \xi_4, \xi_5, \xi_7)$.

- See Munos and Moore, 2001 for further details.

Kuhn triangulation (from Munos and Moore)

3.1. Computational issues

Although the number of simplexes inside a rectangle is factorial with the dimension d, the computation time for interpolating the value at any point inside a rectangle is only of order $(d \ln d)$, which corresponds to a sorting of the d relative coordinates $(x_0, ..., x_{d-1})$ of the point inside the rectangle.

Assume we want to compute the indexes $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing a point defined by its relative coordinates $(x_0, ..., x_{d-1})$ with respect to the rectangle in which it belongs to. Let $\{\xi_0, ..., \xi_{2^d}\}$ be the corners of this d-rectangle. The indexes of the corners use the binary decomposition in dimension d, as illustrated in Figure 2. Computing these indexes is achieved by sorting the coordinates from the highest to the smallest: there exist indices $j_0, ..., j_{d-1}$, permutation of $\{0, .., d-1\}$, such that $1 \geq x_{j_0} \geq x_{j_1} \geq ... \geq x_{j_{d-1}} \geq 0$. Then the indices $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing the point are: $i_0 = 0$, $i_1 = i_0 + 2^{j_0}$, ..., $i_k = i_{k-1} + 2^{j_{k-1}}$, ..., $i_d = i_{d-1} + 2^{j_{d-1}} = 2^d - 1$. For example, if the coordinates satisfy: $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ (illustrated by the point x in Figure 2) then the vertices are: ξ_0 (every simplex contains this vertex, as well as $\xi_{2^d-1} = \xi_7$), ξ_4 (we added 2^2), ξ_5 (we added 2^0) and ξ_7 (we added 2^1).

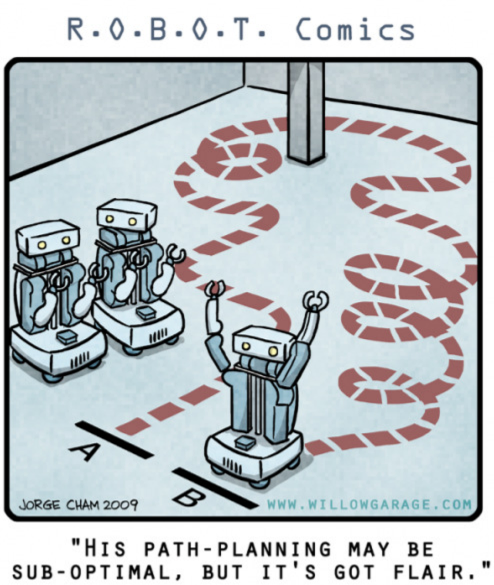
Let us define the *barycentric coordinates* $\lambda_0, ..., \lambda_d$ of the point x inside the simplex $\xi_{i_0}, ..., \xi_{i_d}$ as the positive coefficients (uniquely) defined by: $\sum_{k=0}^{d} \lambda_k = 1$ and $\sum_{k=0}^{d} \lambda_k \xi_{i_k} = x$. Usually, these barycentric coordinates are expensive to compute; however, in the case of Kuhn triangulation these coefficients are simply: $\lambda_0 = 1 - x_{j_0}$, $\lambda_1 = x_{j_0} - x_{j_1}$, ..., $\lambda_k = x_{j_{k-1}} - x_{j_k}$, ..., $\lambda_d = x_{j_{d-1}} - 0 = x_{j_{d-1}}$. In the previous example, the barycentric coordinates are: $\lambda_0 = 1 - x_2$, $\lambda_1 = x_2 - x_0$, $\lambda_2 = x_0 - x_1$, $\lambda_3 = x_1$.

CS 287: Advanced Robotics
Fall 2009

Lecture 6: Control 5: Optimal control --- [Function approximation in dynamic programming---special case: quadratic]

Pieter Abbeel
UC Berkeley EECS

PHD



Announcements

- Will there be lecture this Thursday (Sept 24)?
  - Yes.

- No office hours this Thursday (as I am examining students for prelims).
- Feel free to schedule an appointment by email instead.

Announcements

- Final project contents:
  - Original investigation into an area that relates sufficiently closely to the course.
    - Could be algorithmic/theoretical idea
    - Could be application of existing algorithm(s) to a platform or domain in which these algorithms carry promise but have not been applied
    - Alternatively: Significant improvement for an existing (or new) assignment for this course or for an existing (or new) assignment for 188 which has close ties to this course.
  - Ideally: we are able to identify a topic that relates both to your on-going PhD research and the course.
  - You are very welcome to come up with your own project ideas, yet make sure to pass them by me **before** you submit your abstract.
  - Feel free to stop by office hours or set an appointment (via email) to discuss potential projects.

Announcements

- Final project logistics:
  - Final result: 6-8 page paper.
    - Should be structured like a conference paper, i.e., focus on the problem setting, why it matters, what is interesting/unsolved about it, your approach, results, analysis, and so forth. Cite and briefly survey prior work as appropriate, but don't re-write prior work when not directly relevant to understand your approach.
  - Milestones:
    - Oct. 9th, 23:59: **Approved-by-me** abstracts due: 1 page description of project + goals for milestone. Make sure to sync up with me before then!
    - Nov 9th, 23:59: 1 page milestone report due
    - Dec 3rd, In-class project presentations [tentatively]
    - Dec 11th, 23:59: Final paper due
  - 1 or 2 students/project. If you are two students on 1 final project, I will expect twice as much research effort has gone into it!

Bellman's curse of dimensionality

- n-dimensional state space
- Number of states grows exponentially in n
- In practice
  - Discretization is considered only computationally feasible up to 5 or 6 dimensional state spaces even when using
    - Variable resolution discretization
    - Very fast implementations

Today

- Linear Quadratic (LQ) setting --- special case: can solve continuous optimal control problem exactly

Great reference:

[optional] Anderson and Moore, Linear Quadratic Methods --- standard reference for LQ setting

Linear Quadratic Regulator (LQR)

The LQR setting assumes a linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t,$$

x_t: state at time t
u_t: input at time t
It assumes a quadratic cost function:

$$g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$$

with $Q \succ 0, R \succ 0$.

For a square matrix X we have $X \succ 0$ if and only if for all vectors z we have $z^\top X z > 0$. Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.

While LQ assumptions might seem very restrictive, we will see the method can be made applicable for non-linear systems, e.g., helicopter.

Value iteration

- Back-up step for i+1 steps to go:

$$J_{i+1}(s) \leftarrow \min_u g(s,u) + \gamma \sum_{s'} P(s'|s,u) J_i(s')$$

- LQR:

$$J_{i+1}(x) \leftarrow \min_u x^\top Q x + u^\top R u + \gamma \sum_{x'=Ax+Bu} J_i(x')$$

$$= \min_u \left[x^\top Q x + u^\top R u + \gamma J_i(Ax+Bu) \right]$$

LQR value iteration: J_1

$$J_{i+1}(x) \leftarrow \min_u \left[x^\top Q x + u^\top R u + J_i(Ax + Bu) \right]$$

Initialize $J_0(x) = x^\top P_0 x$.

$$\begin{aligned} J_1(x) &= \min_u \left[x^\top Q x + u^\top R u + J_0(Ax + Bu) \right] \\ &= \min_u \left[x^\top Q x + u^\top R u + (Ax + Bu)^\top P_0 (Ax + Bu) \right] \qquad (1) \end{aligned}$$

To find the minimum over u, we set the gradient w.r.t. u equal to zero:

$$\nabla_u [\ldots] = 2Ru + 2B^\top P_0 (Ax + Bu) = 0,$$

$$\text{hence: } u = -(R + B^\top P_0 B)^{-1} B^\top P_0 A x \qquad (2)$$

$$\begin{aligned} \text{(2) into (1): } J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{aligned}$$

LQR value iteration: J_1 (ctd)

- In summary:

$$\begin{aligned} &J_0(x) = x^\top P_0 x \\ &x_{t+1} = Ax_t + Bu_t \\ &g(x, u) = u^\top R u + x^\top Q x \end{aligned}$$

$$\begin{aligned} J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{aligned}$$

- $J_1(x)$ is quadratic, just like $J_0(x)$.

  →Value iteration update is the same for all times and can be done in closed form!

$$\begin{aligned} J_2(x) &= x^\top P_2 x \\ \text{for: } P_2 &= Q + K_2^\top R K_2 + (A + BK_2)^\top P_1 (A + BK_2) \\ K_2 &= -(R + B^\top P_1 B)^{-1} B^\top P_1 A. \end{aligned}$$

Value iteration solution to LQR

Set $P_0 = 0$.
for $i = 1, 2, 3, \ldots$

$$\begin{aligned} K_i &= -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A \\ P_i &= Q + K_i^\top R K_i + (A + BK_i)^\top P_{i-1} (A + BK_i) \end{aligned}$$

The optimal policy for a i-step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i-step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

LQR assumptions revisited

$$\begin{aligned} x_{t+1} &= A x_t + B u_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

= for keeping a linear system at the all-zeros state.

- Extensions which make it more generally applicable:
  - Affine systems
  - System with stochasticity
  - Regulation around non-zero fixed point for non-linear systems
  - Penalization for change in control inputs
  - Linear time varying (LTV) systems
  - Trajectory following for non-linear systems

LQR Ext0: Affine systems

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + c \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

- Optimal control policy remains linear, optimal cost-to-go function remains quadratic
- Two avenues to do derivation:
  - 1. Work through the DP update as we did for standard setting
  - 2. Redefine the state as: z_t = [x_t; 1], then we have:

$$z_{t+1} = \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_t = A' z_t + B' u_t$$

LQR Ext1: stochastic system

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \\ & \quad w_t, t = 0, 1, \ldots \text{are zero mean and independent} \end{aligned}$$

- Exercise: work through similar derivation as we did for the deterministic case.
- Result:
  - Same optimal control policy
  - Cost-to-go function is almost identical: has one additional term which depends on the variance in the noise (and which cannot be influenced by the choice of control inputs)

LQR Ext2: non-linear systems

Nonlinear system: $x_{t+1} = f(x_t, u_t)$

We can keep the system at the state x^* iff

$$\exists u^* \text{s.t.} \quad x^* = f(x^*, u^*)$$

Linearizing the dynamics around x^* gives:

$$x_{t+1} \approx f(x^*, u^*) + \underbrace{\frac{\partial f}{\partial x}(x^*, u^*)}_{A}(x_t - x^*) + \underbrace{\frac{\partial f}{\partial u}(x^*, u^*)}_{B}(u_t - u^*)$$

Equivalently:

$$x_{t+1} - x^* \approx A(x_t - x^*) + B(u_t - u^*)$$

Let $z_t = x_t - x^*$, let $v_t = u_t - u^*$, then:

$$z_{t+1} = Az_t + Bv_t, \qquad \text{cost} = z_t^\top Q z_t + v_t^\top R v_t$$ [=standard LQR]

$$v_t = K z_t \Rightarrow u_t - u^* = K(x_t - x^*) \Rightarrow u_t = u^* + K(x_t - x^*)$$

LQR Ext3: penalize for change in control inputs

- Standard LQR:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

- When run in this format on real systems: often high frequency control inputs get generated. Typically highly undesirable and results in poor control performance.
- Why?
- Solution: frequency shaping of the cost function. (See, e.g., Anderson and Moore.)
- Simple special case which works well in practice: penalize for change in control inputs. ---- How ??

LQR Ext3: penalize for change in control inputs

- Standard LQR:

$$
\begin{aligned}
x_{t+1} &= Ax_t + Bu_t \\
g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t
\end{aligned}
$$

- How to incorporate the change in controls into the cost/reward function?
  - Soln. method A: explicitly incorporate into the state and the reward function, and re-do the derivation based upon value iteration.
  - Soln. method B: change of variables to fit into the standard LQR setting.

LQR Ext3: penalize for change in control inputs

- Standard LQR:

$$
\begin{aligned}
x_{t+1} &= Ax_t + Bu_t \\
g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t
\end{aligned}
$$

- Introducing change in controls Δu:

$$
\underbrace{\begin{bmatrix} x_{t+1} \\ u_{t+1} \end{bmatrix}}_{x'_{t+1}} \underbrace{=}_{=} \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{A'} \underbrace{\begin{bmatrix} x_t \\ u_{t-1} \end{bmatrix}}_{x'_t} \underbrace{+}_{+} \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{B'} \underbrace{\Delta u_t}_{u'_t}
$$

$$
\text{cost} = -(x'^\top Q' x' + \Delta u^\top R' \Delta u) \qquad Q' = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}
$$

$$
R' = \text{penalty for change in controls}
$$

[If R'=0, then equivalent to standard LQR.]

LQR Ext4: Linear Time Varying (LTV) Systems

$$
\begin{aligned}
x_{t+1} &= A_t x_t + B_t u_t \\
g(x_t, u_t) &= x_t^\top Q_t x_t + u_t^\top R_t u_t
\end{aligned}
$$

LQR Ext4: Linear Time Varying (LTV) Systems

Set $P_0 = 0$.
for $i = 1, 2, 3, \ldots$

$$
\begin{aligned}
K_i &= -(R_{H-i} + B_{H-i}^\top P_{i-1} B_{H-i})^{-1} B_{H-i}^\top P_{i-1} A_{H-i} \\
P_i &= Q_{H-i} + K_i^\top R_{H-i} K_i + (A_{H-i} + B_{H-i} K_i)^\top P_{i-1} (A_{H-i} + B_{H-i} K_i)
\end{aligned}
$$

The optimal policy for a i-step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i-step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

LQR Ext5: Trajectory following for non-linear systems

- A state sequence x_o^*, x_1^*, …, x_H^* is a feasible target trajectory iff

$$\exists u_0^*, u_1^*, \ldots, u_{H-1}^* \ : \ \forall t \in \{0, 1, \ldots, H-1\} \ : \ x_{t+1}^* = f(x_t^*, u_t^*)$$

- Problem statement:

$$\min_{u_0, u_1, \ldots, u_{H-1}} \sum_{t=0}^{H-1} (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*)$$
$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

- Transform into linear time varying case (LTV):

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t}(x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t}(u_t - u_t^*)$$

$$x_{t+1} - x_t^* \approx A_t(x_t - x_t^*) + B_t(u_t - u_t^*)$$

LQR Ext5: Trajectory following for non-linear systems

- Transformed into linear time varying case (LTV):

$$\min_{u_0, u_1, \ldots, u_{H-1}} \sum_{t=0}^{H-1} (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*)$$

$$\text{s.t. } x_{t+1} - x_{t+1}^* \approx A_t(x_t - x_t^*) + B_t(u_t - u_t^*)$$

- Now we can run the standard LQR back-up iterations.
- Resulting policy at i time-steps from the end:

$$u_{H-i} - u_{H-i}^* = K_i(x_{H-i} - x_{H-i}^*)$$

- The target trajectory need not be feasible to apply this technique, however, if it is infeasible then the linearizations are not around the (state,input) pairs that will be visited

Most general cases

- Methods which attempt to solve the generic optimal control problem

$$\begin{array}{ll} \min_u & \sum_{t=0}^{H} g(x_t, u_t) \\ \text{subject to} & x_{t+1} = f(x_t, u_t) \;\; \forall t \end{array}$$

by iteratively approximating it and leveraging the fact that the linear quadratic formulation is easy to solve.

Iteratively apply LQR

Initialize the algorithm by picking either (a) A control policy $\pi^{(0)}$ or (b) A sequence of states $x_0^{(0)}, x_1^{(0)}, \ldots, x_H^{(0)}$ and control inputs $u_0^{(0)}, u_1^{(0)}, \ldots, u_H^{(0)}$. With initialization (a), start in Step (1). With initialization (b), start in Step (2).

Iterate the following:

(1) Execute the current policy $\pi^{(i)}$ and record the resulting state-input trajectory $x_0^{(i)}, u_0^{(i)}, x_1^{(i)}, u_1^{(i)}, \ldots, x_H^{(i)}, u_H^{(i)}$.

(2) Compute the LQ approximation of the optimal control around the obtained state-input trajectory by computing a first-order Taylor expansion of the dynamics model, and a second-order Taylor expansion of the cost function.

(3) Use the LQR back-ups to solve for the optimal control policy $\pi^{(i+1)}$ for the LQ approximation obtained in Step (2).

(4) Set $i = i + 1$ and go to Step (1).

Iterative LQR: in standard LTV format

Standard LTV is of the form $z_{t+1} = A_t z_t + B_t v_t$, $g(z, v) = z^\top Q z + v^\top R v$.

Linearizing around $(x_t^{(i)}, u_t^{(i)})$ in iteration i of the iterative LQR algorithm gives us (up to first order!):

$$x_{t+1} = f(x_t^{(i)}, u_t^{(i)}) + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Subtracting the same term on both sides gives the format we want:

$$x_{t+1} - x_{t+1}^{(i)} = f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^{(i)} + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Hence we get the standard format if using:

$$\begin{aligned}
z_t &= [x_t - x_t^{(i)} \quad 1]^\top \\
v_t &= (u_t - u_t^{(i)}) \\
A_t &= \begin{bmatrix} \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)}) & f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^{(i)} \\ 0 & 1 \end{bmatrix} \\
B_t &= \begin{bmatrix} \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)}) \\ 0 \end{bmatrix}
\end{aligned}$$

Iteratively apply LQR: convergence

- Need not converge as formulated!
  - Reason: the optimal policy for the LQ approximation might end up not staying close to the sequence of points around which the LQ approximation was computed by Taylor expansion.
  - Solution: in each iteration, adjust the cost function so this is the case, i.e., use the cost function

    $$(1-\alpha)g(x_t, u_t) + \alpha(\|x_t - x_t^{(i)}\|_2^2 + \|u_t - u_t^{(i)}\|_2^2)$$

    Assuming g is bounded, for α close enough to one, the 2nd term will dominate and ensure the linearizations are good approximations around the solution trajectory found by LQR.

Iteratively apply LQR: practicalities

- f is non-linear, hence this is a non-convex optimization problem. Can get stuck in local optima! Good initialization matters.
- g could be non-convex: Then the LQ approximation fails to have positive-definite cost matrices.

Iterative LQR: in standard LTV format

Standard LTV is of the form $z_{t+1} = A_t z_t + B_t v_t$, $g(z,v) = z^\top Q z + v^\top R v$.

Linearizing around $(x_t^{(i)}, u_t^{(i)})$ in iteration i of the iterative LQR algorithm gives us (up to first order!):

$$x_{t+1} = f(x_t^{(i)}, u_t^{(i)}) + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Subtracting the same term on both sides gives the format we want:

$$x_{t+1} - x_{t+1}^{(i)} = f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^{(i)} + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Hence we get the standard format if using:

$$\begin{aligned}
z_t &= [x_t - x_t^{(i)} \quad 1]^\top \\
v_t &= (u_t - u_t^{(i)}) \\
A_t &= \begin{bmatrix} \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)}) & f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^{(i)} \\ 0 & 1 \end{bmatrix} \\
B_t &= \begin{bmatrix} \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)}) \\ 0 \end{bmatrix}
\end{aligned}$$

A similar derivation is needed to find Q and R.

Iterative LQR for trajectory following

While there is no need to follow this particular route, this is a (imho) particularly convenient way of turning the linearized and quadraticized approximation in the iLQR iterations into the standard LQR format for the setting of trajectory following with a quadratic penalty for deviation from the trajectory.

Let $x_t^{(i)}, u_t^{(i)}$ be the state and control around which we linearize. Let x_t^*, u_t^* be the target controls then we have:

$$
\begin{aligned}
x_{t+1} &= f(x_t^{(i)}, u_t^{(i)}) + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)}) \\
x_{t+1} - x_{t+1}^* &= f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^* + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)} - x_t^* + x_t^*) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)} - u_t^* + u_t^*) \\
x_{t+1} - x_{t+1}^* &= f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^* + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^*) + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t^* - x_t^{(i)}) \\
&\quad + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^*) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t^* - u_t^{(i)}) \\
[x_{t+1} - x_{t+1}^*; 1] &= A[(x_t - x_t^*); 1] + B(u_t - u_t^*)
\end{aligned}
$$

For

$$
A = \begin{bmatrix} \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)}) & f(x_t^{(i)}, u_t^{(i)}) - x_{t+1}^* + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t^* - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t^* - u_t^{(i)}) \\ 0 & 1 \end{bmatrix}
$$

and

$$
B = \begin{bmatrix} \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)}) \\ 0 \end{bmatrix}
$$

The cost function can be used as is: $(x_t - x_t^*)^\top Q(x_t - x_t^*) + (u_t - u_t^*)^\top R(u_t - u_t^*)$.

Differential Dynamic Programming (DDP)

- Often loosely used to refer to iterative LQR procedure.
- More precisely: Directly perform 2nd order Taylor expansion of the Bellman back-up equation [rather than linearizing the dynamics and 2nd order approximating the cost]
- Turns out this retains a term in the back-up equation which is discarded in the iterative LQR approach
- [It’s a quadratic term in the dynamics model though, so even if cost is convex, resulting LQ problem could be non-convex …]

[Typically cited book: Jacobson and Mayne, “Differential dynamic programming,” 1970]

Differential dynamic programming

$$\begin{aligned} J_{i+1}(x) &= \min_u \\ &\quad \text{2nd order expansion of } g \text{ around } (x^*, u^*) \\ &\quad + J_i(f(x^*, u^*)) \\ &\quad + \frac{dJ}{dx}(f(x,u) - f(x^*, u^*)) \\ &\quad + (f(x,u) - f(x^*, u^*))^\top \frac{d^2 J}{dx^2}(f(x,u) - f(x^*, u^*)) \end{aligned}$$

To keep entire expression 2nd order:
Use Taylor expansions of f and then remove all resulting terms which are higher than 2nd order.
Turns out this keeps 1 additional term compared to iterative LQR

Can we do even better?

- Yes!
- At convergence of iLQR and DDP, we end up with linearizations around the (state,input) trajectory the algorithm converged to
- In practice: the system could not be on this trajectory due to perturbations / initial state being off / dynamics model being off / …
- Solution: at time t when asked to generate control input ut, we could re-solve the control problem using iLQR or DDP over the time steps t through H
- Replanning entire trajectory is often impractical → in practice: replan over horizon h. = ***receding horizon control***
  - This requires providing a cost to go J^{(t+h)} which accounts for all future costs. This could be taken from the offline iLQR or DDP run

Multiplicative noise

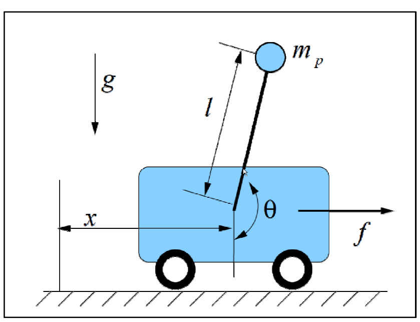
- In many systems of interest, there is noise entering the system which is multiplicative in the control inputs, i.e.:

$$x_{t+1} = Ax_t + (B + B_w w_t)u_t$$

- Exercise: LQR derivation for this setting

[optional related reading:Todorov and Jordan, nips 2003]

Cart-pole

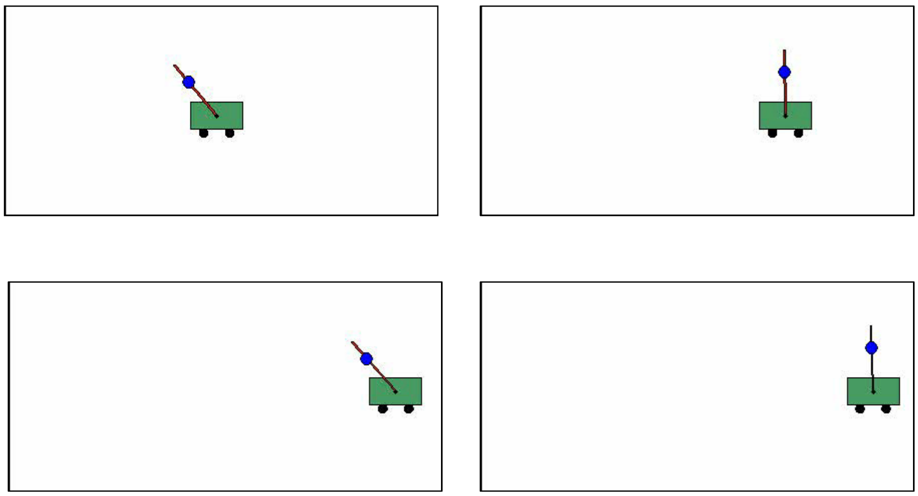


$$\boxed{H(q)\ddot{q} + C(q,\dot{q}) + G(q) = B(q)u}$$

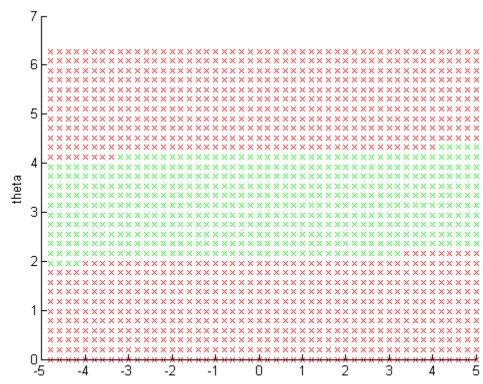
$$\begin{aligned}
H(q) &= \begin{bmatrix} m_c + m_p & m_p l \cos\theta \\ m_p l \cos\theta & m_p l^2 \end{bmatrix} \\
C(q,\dot{q}) &= \begin{bmatrix} 0 & -m_p l \dot{\theta} \sin\theta \\ 0 & 0 \end{bmatrix} \\
G(q) &= \begin{bmatrix} 0 \\ m_p g l \sin\theta \end{bmatrix} \\
B &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\end{aligned}$$

[See also Section 3.3 in Tedrake notes.]

Cart-pole --- LQR

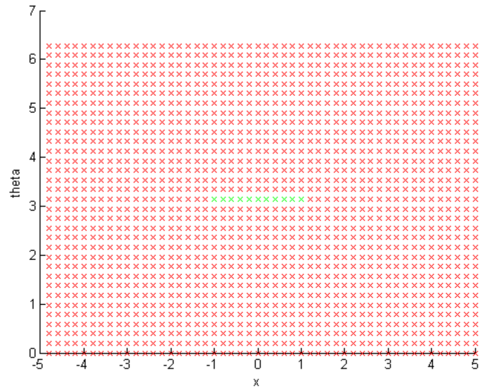


Cart-pole --- LQR



Q = diag([1;1;1;1]); R = 0; [x, theta, xdot, thetadot]

Cart-pole --- LQR



Q = diag([1;1;1;1]); R = 1; [x, theta, xdot, thetadot]

Lyapunov's linearization method

We will not cover any details, but here is the basic result:

Assume x is an equilibrium point for f(x), i.e., x* = f(x*).*

If x is an asymptotically stable equilibrium point for the linearized system, then it is asymptotically stable for the non-linear system.*

If x is unstable for the linear system, it's unstable for the non-linear system.*

If x is marginally stable for the linear system, no conclusion can be drawn.*

This provides additional justification for using linear control design techniques for non-linear systems.

[See, e.g., Slotine and Li, or Boyd lecture notes (pointers available on course website) if you want to find out more.]

CS 287: Advanced Robotics
Fall 2009

Lecture 8: Control 7: MPC --- Feedback Linearization --- Controllability --- Lagrangian Dynamics

Pieter Abbeel
UC Berkeley EECS

Model predictive control (MPC)

- Optimal control problem

  Given a system with (stochastic) dynamics: $x_{t+1} = f(x_t, u_t, w_t)$ Find the optimal policy π which minimizes the expected cost:

$$\min_{\pi} \mathrm{E}[\sum_{t=0}^{H} g(x_t, u_t) | \pi]$$

- MPC:

  For $t = 0, 1, 2, \ldots$

  1. Solve

$$\begin{array}{ll} \min_{u_t, u_{t+1}, \ldots, u_H} & \sum_{k=t}^{H} g(x_k, u_k) \\ \text{s.t.} & x_{k+1} = f(x_k, u_k, 0) \quad \forall k = t, t+1, \ldots, H-1 \end{array}$$

  2. Execute u_t from the solution found in (1).

- In practice, one often ends up having to solve:

$$\begin{array}{ll} \min_{u_t, u_{t+1}, \ldots, u_{t+h-1}} & \sum_{k=t}^{t+h} g(x_k, u_k) + \bar{g}(x_{t+h}, u_{t+h}) \\ \text{s.t.} & x_{k+1} = f(x_k, u_k, 0) \quad \forall k = t, t+1, \ldots, t+h-1 \end{array}$$

Single shooting

- At core of MPC, need to quickly solve problems of the form:

$$\begin{array}{ll} \min_{u_t, u_{t+1}, \ldots, u_H} & \sum_{k=t}^{H} g(x_k, u_k) \\ \text{s.t.} & x_{k+1} = f(x_k, u_k, 0) \;\; \forall k = t, t+1, \ldots, H-1 \end{array}$$

- Single shooting methods directly solve for

  i.e., they solve:

- Underneath, this typically boils down to iterating:
  - For the current simulate and find the state sequence
  - Take the 1st (and 2nd) derivatives w.r.t.
- Note: When taking derivatives, one ends up repeatedly applying the chain rule and the same Jacobians keep re-occurring
- → Beneficial to not waste time re-computing same Jacobians; pretty straightforward, various specifics with their own names. (E.g., back-propagation.)

Single shooting drawback

- Numerical conditioning of the problem:
  - Influence on objective function of earlier actions vs. later actions
- What happens in case of a non-linear, unstable system?

Multiple shooting/Direct collocation

- Keep the state at each time in the optimization problem:

$$\begin{array}{ll} \min_{u_t,u_{t+1},\ldots,u_H,x_t,x_{t+1},\ldots,x_H} & \sum_{k=t}^{H} g(x_k,u_k) \\ \text{s.t.} & x_{k+1} = f(x_k,u_k,0) \;\; \forall k = t, t+1, \ldots, H-1 \\ & h_k(x_k,u_k) \leq 0 \;\; \forall k = t, t+1, \ldots, H-1 \end{array}$$

- Larger optimization problem, yet sparse structure.
- Special case: Linear MPC: f linear, h, g convex → convex opt. problem, "easily" solved

Sequential Quadratic Programming (SQP)

- Goal: solve

$$\begin{array}{ll} \min_{u_t,u_{t+1},\ldots,u_H,x_t,x_{t+1},\ldots,x_H} & \sum_{k=t}^{H} g(x_k,u_k) \\ \text{s.t.} & x_{k+1} = f(x_k,u_k,0) \;\; \forall k = t, t+1, \ldots, H-1 \\ & h_k(x_k,u_k) \leq 0 \;\; \forall k = t, t+1, \ldots, H-1 \end{array}$$

- SQP: Iterates over
  - Linearize f around current point $(\boldsymbol{u}, \boldsymbol{x})$, quadraticize g, h around current point
  - Solve the resulting Quadratic Programming problem to find the updated "current point" (u, x)
- Corresponds to:
  - Write out the first-order necessary conditions for optimality (the Karuhn-Kuhn-Tucker (KKT) conditions)
  - Apply Newton's method to solve the (typically non-linear) KKT equations

Sequential Quadratic Programming (SQP)

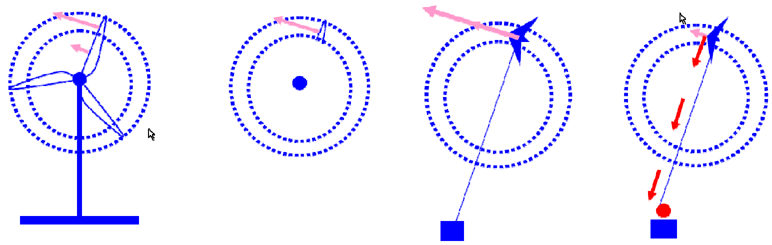
- Not only method, but happens to be quite popular
- Packages available, such as SNOPT, SOCS.
- Many choices underneath:
  - Quasi-Newton methods
- Compared to single shooting:
  - Easier initialization (single shooting relies on control sequence)
  - Easy to incorporate constraints on state / controls
  - More variables, yet good algorithms leverage sparsity to offset this

Further readings

- Tedrake Chapter 9.
- Diehl, Ferreau and Haverbeke, 2008, Nonlinear MPC overview paper
- Francesco Borelli (M.E., UC Berkeley): taught course in Spring 2009 on (linear) MPC
- Packages:
  - SNOPT, ACADO, SOCS, …
- We have ignored:
  - Continuous time aspects
  - Details of optimization methods underneath --- matters in practice b/c the faster the longer horizon
  - Theoretical guarantees

Related intermezzo: Nonlinear control applied to kite-based power generation

[Diehl + al.]



- Many companies pursuing this: Makani, KiteGen, SkySails, AmpyxPower, …
- Number from Diehl et al.: For a 500m2 kite and 10m/s wind speed (in sim) can produce an average power of more than 5MW
- Technically interesting aspect in particular work of Diehl et al.: incorporate open-loop stability into the optimization problem.
  - Only possible for non-linear systems
  - The criterion quantifies how much deviation from the nominal trajectory would amplify/decrease in one cycle

Non-minimum phase example

[Slotine and Li, p. 195, Example II.2]

Consider the linear system

$$\ddot{y} + 2\dot{y} + 2y = -\dot{u} + u$$

The system is non-minimum phase because it has a zero at $p = 1$. Assume that perfect tracking is achieved, *i.e.*, that $y(t) \equiv y_d(t)$, $\forall\, t \geq 0$. Then, the input u satisfies

$$\dot{u} - u = -(\ddot{y}_d + 2\dot{y}_d + 2y_d)$$

Since this represents an unstable dynamics, u diverges exponentially. Note that the above dynamics has a pole which exactly coincides with the unstable zero of the original system, *i.e.*, perfect tracking for non-minimum phase systems can be achieved only by infinite control inputs. By writing u as

$$u = -\frac{p^2 + 2p + 2}{p - 1}\, y_d$$

we see that the perfect-tracking controller is actually inverting the plant dynamics. □

Feedback linearization

Feedback linearization

Feedback linearization

Feedback linearization

Feedback linearization

- Further readings:
  - Slotine and Li, Chapter 6
  - Isidori, Nonlinear control systems, 1989.

Announcements

- Reminder: No office hours today.
  - [Feel free to schedule over email instead]

Controllability [defn., linear systems]

- A system $x_{t+1} = f(x_t, u_t)$ if for all x_0 and all x, there exists a time k and a control sequence $u_0, \ldots, u_{k-1}$ such that $x_t = x$.

Fact. The linear system $x_{t+1} = Ax_t + Bu_t$ with $x_t \in \Re^n$ is controllable iff $[B\ AB\ A^2B\ \ldots\ A^nB]$ is full rank.

Lagrangian dynamics

[From: Tedrake Appendix A]

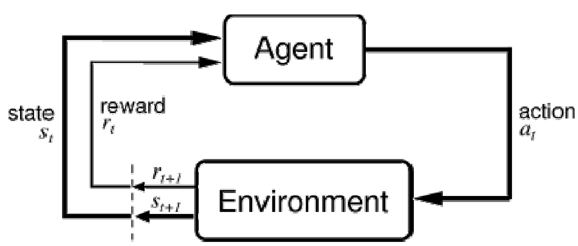
Lagrangian dynamics: example

CS 287: Advanced Robotics
Fall 2009

Lecture 9: Reinforcement Learning 1: Bandits

Pieter Abbeel
UC Berkeley EECS

Reinforcement Learning



- Model: Markov decision process (S, A, T, R, γ)
  - Goal: Find π that maximizes expected sum of rewards
- T and R might be unknown

[Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998]

Exploration vs. exploitation

- = classical dilemma in reinforcement learning
- A conceptual solution: Bayesian approach:
  - State space = { x : x = probability distribution over T, R}
    - For known initial state --- tree of sufficient statistics could suffice
  - Transition model: describes transitions in new state space
  - Reward = standard reward
- Today: one particular setting in which the Bayesian solution is in fact computationally practical

Multi-armed bandits

- Slot machines
- Clinical trials
- Advertising
- Merchandising

Multi-armed bandits

Consider slot machines $H_1, H_2, \ldots, H_n$.

Slot machine i has pay-off $= \begin{cases} 0, & \text{with probability } 1-\theta_i \\ 1, & \text{with probability } \theta_i \end{cases}$

where θ_i is unknown.

Now the objective to maximize is:

$$E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \text{ (where } s_t \text{ is unchanged).}$$

Information state

$$\begin{pmatrix} \#\text{ of successes on } H_1 \\ \#\text{ of failures on } H_1 \\ \#\text{ of successes on } H_2 \\ \#\text{ of failures on } H_2 \\ \vdots \\ \#\text{ of successes on } H_n \\ \#\text{ of failures on } H_n \end{pmatrix}$$

Semi-MDP

- Transition model:

$$P(s_{t_{k+1}} = s', t_{k+1} = t_k + \Delta | s_k = s, a_k = a)$$

- Objective:

$$E[\sum_{k=0}^{\infty} \gamma^{t_k} R(s_{t_{k+1}}, \Delta_k, s_{t_k})].$$

- Bellman update:

$$V(s) = \max_a \sum_{s',\Delta} P(s', \Delta | s, a)[R(s, \Delta, a, s') + \gamma^{\Delta} V(s')]$$

Optimal stopping

- A specialized version of the Semi-MDP is is the "Optimal stopping problem". At each of the times, we have two choices:
  - 1. continue
  - 2. stop and accumulate reward g for current time and for all future times.
- The optimal stopping problem has the following Bellman update:

$$V(s) = \max\{\sum_{s',\Delta} P(s', \Delta | s)[R(s, \Delta, s') + \gamma^{\Delta} V(s')], \frac{g}{1-\gamma})$$

Optimal stopping

- Optimal stopping Bellman update:

$$V(s) = \max\{\sum_{s',\Delta} P(s',\Delta|s)[R(s,\Delta,s') + \gamma^{\Delta} V(s')], \frac{g}{1-\gamma})$$

- Hence, for fixed g, we can find the value of each state in the optimal stopping problem by dynamic programming
- However, we are interested in g*(s) for all s:

$$g^*(s) = \min\{g | \frac{g}{1-\gamma} \geq \max_{\tau} \mathbb{E}_{\tau}[\sum_{k=0}^{\tau-1} \gamma^{t_k} R(s_{t_k}, \Delta_k, s_{tk+1}) + \sum_{t=\tau}^{\infty} \gamma^t g]$$

- Note: τ is a random variable, which denotes the stopping time. It is the policy in this setting.
- Any stopping policy can be represented as a set of states in which we decided to stop. The random variable τ takes on the value = time when we first visit a state in the stopping set.

Optimal stopping

- One approach:
  - Solve the optimal stopping problem for many values of g, and for each state keep track of the smallest value of g which causes stopping

Reward rate

- Reward rate

$$\sum_{t=0}^{\Delta_{t_k}-1} \gamma^t r(s_{t_k}, \Delta_{t_k}, s_{t_{k+1}}) = R(s_{t_k}, \Delta_k, s_{t_{k+1}})$$

- Expected reward rate

$$\bar{r}(s) = \mathbb{E}_{\Delta,s'}[r(s,\Delta,s')] = \sum_{\Delta,s'} P(s',\Delta|s) r(s,\Delta,s')$$

Basic idea to find g*

Now consider

$$s^* = \arg\max_s \bar{r}(s).$$

Of all states, the state s^* would require the highest payoff to be willing to stop. Namely,

$$g^*(s^*) = \bar{r}(s^*)$$

This means that when $g < g^*(s^*)$, the optimal stopping policy will choose to continue at s^*.
Note that for $s \neq s^*$, $g^*(s) < g^*(s^*)$.

To compute $g^*(s)$ for the other states, we consider a new semi-MDP which differs from the existing one only in that we always continue when at state s^*. This is equivalent to letting the new state space $\bar{S} = S \backslash \{s^*\}$.

Finding the optimal stopping costs

$while |\bar{\mathcal{S}}| > 0:$

$\mathcal{S} \leftarrow \mathcal{S} \backslash \{s^*\}$

Adjust the transition model and the reward function accordingly—namely, assuming we always continue when visiting state s^*.

Compute reward rates$r$ by solving: $\sum_{t=0}^{\Delta-1} \gamma^t \bar{r}(s, \Delta, s') = \bar{R}(s, \Delta, s')$

Compute expected reward rates $\bar{r}(s) = \mathbb{E}_{\Delta, s'} r(s, \Delta, s')$.

$s^* \leftarrow \arg\max_s \bar{r}(s)$

$g^*[s^*] \leftarrow \bar{r}(s^*)$

Solving the multi-armed bandit

- 1. Find the optimal stopping cost $g^*(s_t^{(i)})$ for each bandit's current state
- 2. When asked to act, pull an arm i such that

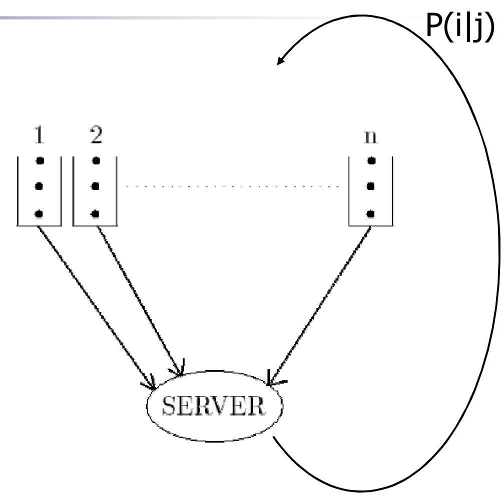
$$i \in \arg\max_i g^{*(i)}(s_t^{(i)})$$

Key requirements

- Reward at time t only depends on state of M_i at time t
- When pulling M_i, only state of M_i changes

- Note: M_i need not "just" be a bandit; we just need to be able to compute its optimal stopping cost

Example: cashier's nightmare



P(i|j): probability of joining queue i after being served in queue j
c_i : cost of a customer being in queue i

Further readings

- Gittins, J.C., D.M. Jones. 1974. A dynamic allocation index for the sequential design of experiments.*["Gittins indices"]*
- Different family of approaches: regret-based
  - Lai and Robbins, 1985
  - Auer +al, UCB algorithm (1998)

  Type of result: after n plays, the regret is bounded by an expression O(log n)

  After n plays the regret is defined by:

  $$n\mu^* - \sum_j \mu_j \mathrm{E}[T_j(n)] \text{where } \mu^* = \max_j \mu_j$$

  Deterministic policy: UCB1.
  Initialization: Play each machine once.
  Loop:

  – Play machine j that maximizes $\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$, where $\bar{x}_j$ is the average reward obtained from machine j, n_j is the number of times machine j has been played so far, and n is the overall number of plays done so far.

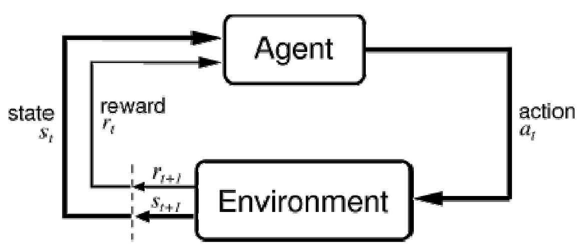
- Loosening and strengthening assumptions, e.g.,
  - Guha, S., K. Munagala. 2007. Approximation algorithms for budgeted learning problems. *STOC '07.*
  - Various Robert Kleinberg publications
  - "contextual bandit" setting

CS 287: Advanced Robotics
Fall 2009

Lecture 11: Reinforcement Learning

Pieter Abbeel
UC Berkeley EECS

Reinforcement Learning



- Model: Markov decision process (S, A, T, R, γ)
  - Goal: Find π that maximizes expected sum of rewards
- T and R might be unknown

[Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998]

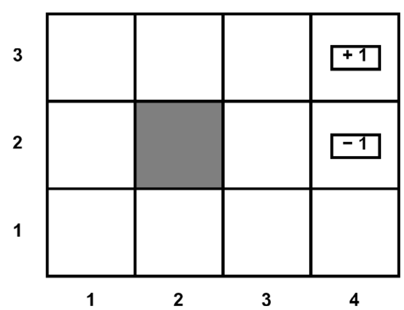
Examples

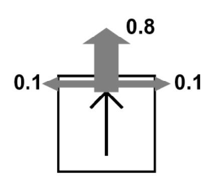
MDP (S, A, T, γ, R), goal: $\max_{\pi} E [\sum_t \gamma^t R(s_t, a_t) \mid \pi]$

- Cleaning robot
- Walking robot
- Pole balancing
- Games: tetris, backgammon
- Server management
- Shortest path problems
- Model for animals, people

Canonical Example: Grid World

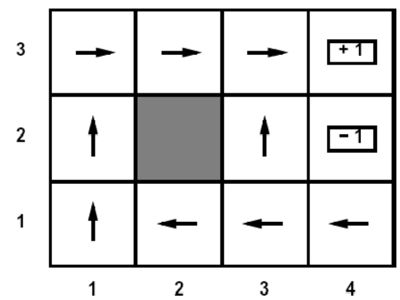
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Big rewards come at the end



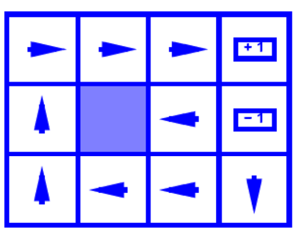


Solving MDPs

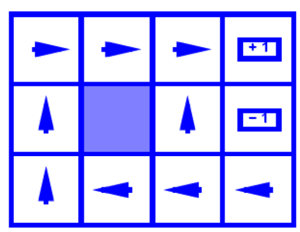
- In deterministic single-agent search problem, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent



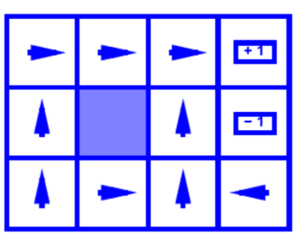
Example Optimal Policies



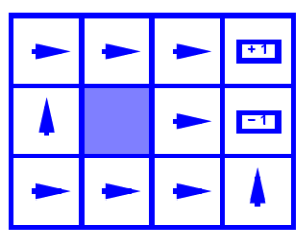
R(s) = -0.02



R(s) = -0.04



R(s) = -0.1



R(s) = -2.0

Outline current and next few lectures

- Recap and extend exact methods
  - Value iteration
  - Policy iteration
  - Generalized policy iteration
  - Linear programming [later]

- Additional challenges we will address by building on top of the above:
  - Unknown transition model and reward function
  - Very large state spaces

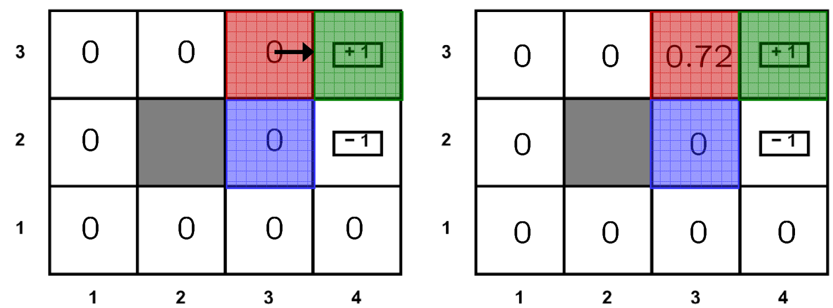
Value Iteration

- Algorithm:
  - Start with $V_0(s) = 0$ for all s.
  - Given V_i, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

  - This is called a value update or Bellman update/back-up
  - Repeat until convergence

Example: Bellman Updates

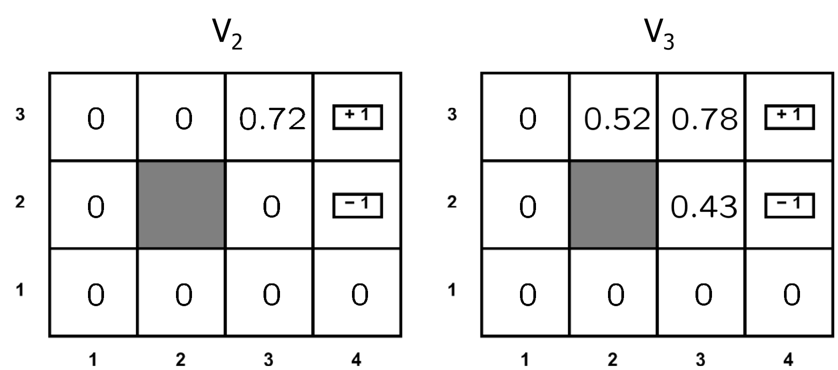


$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma\, V_i(s')\right]$$

$$V_2(\langle 3,3\rangle) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s')\left[R(\langle 3,3\rangle) + 0.9\, V_1(s')\right]$$

$$= 0.9\,[0.8\cdot 1 + 0.1\cdot 0 + 0.1\cdot 0]$$

Example: Value Iteration



- Information propagates outward from terminal states and eventually all states have correct value estimates

Convergence

Infinity norm: $\|V\|_\infty = \max_s |V(s)|$

Fact. Value iteration converges to the optimal value function V^* which satisfies the Bellman equation:

$$\forall s \in S: \quad V^*(s) = \max_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V^*(s'))$$

Or in operator notation: $V^* = TV^*$ where T denotes the Bellman operator.

Fact. If an estimate V satisfies $\|V - TV\|_\infty \leq \epsilon$ then we have that

$$\|V - V^*\|_\infty \leq \frac{\epsilon}{1-\gamma}$$

Practice: Computing Actions

- Which action should we chose from state s:
  - Given optimal values V*?

$$\pi(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

  - = greedy action with respect to V*
  - = action choice with one step lookahead w.r.t. V*

12

Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate value function for a fixed policy (not optimal!) until convergence
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) value function
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

13

Policy Iteration

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

14

Comparison

- Value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- Policy iteration:
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies
- Generalized policy iteration:
  - General idea of two interacting processes revolving around an approximate policy and an approximate value
- Asynchronous versions:
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

15

CS 287: Advanced Robotics
Fall 2009

Lecture 12: Reinforcement Learning

Pieter Abbeel
UC Berkeley EECS

Outline

- LP approach for finding the optimal value function of MDPs
- Model-free approaches

Solving an MDP with linear programming

Solving an MDP with linear programming

Solving an MDP with linear programming

The dual LP

The dual LP: interpretation

$$\max_{\lambda \geq 0} \sum_{s,a,s'} T(s,a,s')\lambda(s,a)R(s,a,s')$$

$$\text{s.t.} \quad \forall s \quad \sum_a \lambda(s,a) = c(s) + \sum_{s',a} \lambda(s',a)T(s',a,s)$$

- Meaning λ(s,a) ?
- Meaning c(s) ?

LP approach recap

The optimal value function satisfies:

$$\forall s : V(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right].$$

We can relax these non-linear equality constraints to inequality constraints:

$$\forall s : V(s) \geq \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right].$$

Equivalently, ($x \geq \max_i y_i$ is equivalent to $\forall i \;\; x \geq y_i$), we have:

$$\forall s, \forall a : V(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]. \quad (1)$$

The relaxation still has the optimal value function as one of its solutions, but we might have introduced new solutions. So we look for an objective function that will favor the optimal value function over other solutions of (1). To this extent, we observed the following monotonicity property of the Bellman operator T:

$$\forall s \;\; V_1(s) \geq V_2(s) \;\; \text{implies}: \;\; \forall s \;\; (TV_1)(s) \geq (TV_2)(s)$$

Any solution to (1) satisfies $V \geq TV$, hence also: $TV \geq T^2V$, hence also: $T^2V \geq T^3V$ ... "$T^{\infty-1}$"$V \geq T^{\infty}V = V^*$. Stringing these together, we get for any solution V of (1) that the following holds:

$$V \geq V^*$$

Hence to find V^* as the solution to (1), it suffices to add an objective function which favors the smallest solution:

$$\min_V c^\top V \quad \text{s.t.} \forall s, \forall a : V(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]. \quad (2)$$

If $c(s) > 0$ for all s, the unique solution to (2) is V^*.

Taking the Lagrange dual of (2), we obtain another interesting LP:

$$\max_{\lambda \geq 0} \sum_{s,a,s'} T(s,a,s')\lambda(s,a)R(s,a,s')$$

$$\text{s.t.} \quad \forall s \quad \sum_a \lambda(s,a) = c(s) + \gamma \sum_{s',a} \lambda(s',a)T(s',a,s)$$

Announcements

- PS 1: posted on class website, due Monday October 26.

- Final project abstracts due tomorrow.

- **Value iteration:**
  - Start with $V_0(s) = 0$ for all s. Iterate until convergence:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

- **Policy iteration:**
  - Policy evaluation: Iterate until values converge

$$V^{\pi_k}_{i+1}(s) \leftarrow \sum_{s'} T(s,\pi_k(s),s') \left[R(s,\pi_k(s),s') + \gamma V^{\pi_k}_i(s')\right]$$

  - Policy improvement:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_k}(s')\right]$$

- **Generalized policy iteration:**
  - Any interleaving of policy evaluation and policy improvement
  - Note: for particular choice of interleaving → value iteration
- **Linear programming:**

$$\min c^\top V \quad \text{s.t. } \forall s,a: \; V(s) \geq \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V(s')\right)$$

What if T and R unknown

- Model-based reinforcement learning
  - Estimate model from experience
  - Solve the MDP as if the model were correct
- Model-free reinforcement learning
  - Adaptations of the exact algorithms which only require (s, a, r, s') traces [some of them use (s, a, r, s', a')]
  - No model is built in the process

Sample Avg to Replace Expectation?

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

- Who needs T and R? Approximate the expectation with samples (drawn from T!)

$$sample_1 = R(s, a, s_1') + \gamma V_i^{\pi}(s_1')$$
$$sample_2 = R(s, a, s_2') + \gamma V_i^{\pi}(s_2')$$
$$\ldots$$
$$sample_k = R(s, a, s_k') + \gamma V_i^{\pi}(s_k')$$

Problem: We need to estimate these too!

Sample Avg to Replace Expectation?

- We could estimate $V^{\pi}(s)$ for all states simultaneously:

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

- Old updates will use very poor estimates of $V^{\pi}(s')$
  - This will surely affect our estimates of $V^{\pi}(s)$ initially, but will this also affect our final estimate?

Sample Avg to Replace Expectation?

- Big idea: why bother learning T?
  - Update V(s) each time we experience (s,a,s')
  - Likely s' will contribute updates more often
- Temporal difference learning (TD or TD(0))
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Exponential Moving Average

- Weighted averages emphasize certain samples

$$\frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (which contains mistakes in TD)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

TD(0) for estimating V^π

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        $a \leftarrow$ action given by π for s
        Take action a; observe reward, r, and next state, s'
        $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
        $s \leftarrow s'$
    until s is terminal

Note: this is really V^π

Convergence guarantees for TD(0)

- Convergence with probability 1 for the states which are visited infinitely often if the step-size parameter decreases according to the "usual" stochastic approximation conditions

$$\sum_{k=0}^{\infty} \alpha_k = \infty$$
$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

- Examples:
  - 1/k
  - C/(C+k)

Experience replay

- If limited number of trials available: could repeatedly go through the data and perform the TD updates again

- Under this procedure, the values will converge to the values under the empirical transition and reward model.

CS 287: Advanced Robotics
Fall 2009

Lecture 13: Reinforcement Learning

Pieter Abbeel
UC Berkeley EECS

Outline

- Model-free approaches
  - Recap TD(0)
  - Sarsa
  - Q learning
  - TD(λ), sarsa(λ), Q(λ)
  - Function approximation and TD
  - TD Gammon

TD(0) for estimating V^π

Stochastic version of the policy evaluation update:

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^\pi(s')\right]$$

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
    $s \leftarrow s'$
  until s is terminal

Note: this is really V^π

Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy---as required for a policy update step---we're sunk:

$$\pi_{k+1}(s) = \arg\max_a Q^{\pi_k}(s, a)$$

$$Q^{\pi_k}(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Update Q values directly

- When experiencing s_t, a_t, s_{t+1}, r_{t+1}, a_{t+1} perform the following "sarsa" update:

$$
\begin{aligned}
Q^\pi(s_t,a_t) &\leftarrow (1-\alpha)Q^\pi(s_t,a_t) + \alpha\left[r(s_t,a_t,s_{t+1}) + \gamma Q^\pi(s_{t+1},a_{t+1})\right] \\
&= Q^\pi(s_t,a_t) + \alpha\left[r(s_t,a_t,s_{t+1}) + \gamma Q^\pi(s_{t+1},a_{t+1}) - Q^\pi(s_t,a_t)\right]
\end{aligned}
$$

- Will find the Q values for the current policy π.
- How about Q(s,a) for action a inconsistent with the policy π at state s?
- Converges (w.p. 1) to Q function for current policy π for all states and actions *if* all states and actions are visited infinitely often (assuming proper step-sizing)

Exploration aspect

- To ensure convergence for all Q(s,a) we need to visit every (s,a) infinitely often

  → The policy π needs to include some randomness

  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to some current policy
  - → This results in a new policy π'
  - *We end up finding the Q values for this new policy π'*

Does policy iteration still work when we execute epsilon greedy policies?

- Policy iteration iterates:
  - Evaluate value of current policy V^{π}
  - Improve policy by choosing the greedy policy w.r.t. V^{π}

- Answer: Using the epsilon greedy policies can be interpreted as running policy iteration w.r.t. a related MDP which differs slighty in its transition model: with probability ϵ the transition is according to a random action in the new MDP

Need not wait till convergence with the policy improvement step

- Recall: Generalized policy iteration methods: interleave policy improvement and policy evaluation and guaranteed to converge to the optimal policy as long as value for every state updated infinitely often

- → Sarsa: continuously update the policy by choosing actions ϵ greedy w.r.t. the current Q function

Sarsa: updates Q values directly

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Sarsa converges w.p. 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, e.g., by having ϵ greedy policies with ϵ = 1 / t).

Q learning

- Directly approximate the optimal Q function Q*:

$$Q(s_t, a_t) \quad \leftarrow \quad (1-\alpha)Q(s_t, a_t) + \alpha\left[r(s_t, a_t, s_{t+1}) + \max_a \gamma Q^{\pi}(s_{t+1}, a)\right]$$

- Compare to sarsa:

$$Q^{\pi}(s_t, a_t) \quad \leftarrow \quad (1-\alpha)Q^{\pi}(s_t, a_t) + \alpha\left[r(s_t, a_t, s_{t+1}) + \gamma Q^{\pi}(s_{t+1}, a_{t+1})\right]$$

Q learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$;
  until s is terminal

Q-Learning Properties

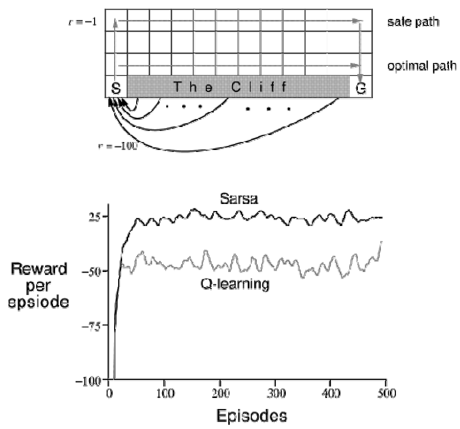
- Will converge to optimal Q function if
  - Every (s,a) visited infinitely often
  - α is chosen to decay according to standard stochastic approximation requirements
- Neat property: learns optimal Q-values regardless of policy used to collect the experience

→ "Off policy" method

- Strictly better than TD, sarsa? Some caveats.

Behaviour of Q-learning vs. sarsa

- Reward = 0 at goal; -100 in cliff region; -1 everywhere else
- $\epsilon = 0.1$



Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability $1-\varepsilon$, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - Takes a long time to explore certain spaces
    - One solution: lower ε over time
    - Another solution: exploration functions

Exploration Functions

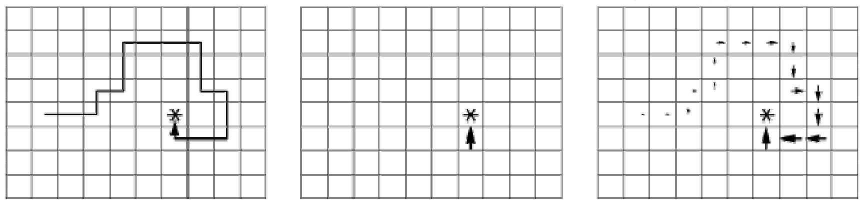
- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established
- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$ (exact form not important---for optimality guarantees: it should guarantee that every (s,a) is visited infinitely often _or_ that Q(s,a) is always optimistic)

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a'))$$

TD(λ) --- motivation (grid world)

TD(λ) --- motivation



TD(λ) "backward view"

- t: $V(s_t) \leftarrow V(s_t) + \alpha[R(s_t) + \gamma V(s_{t+1}) - V(s_t)]$
- t+1: $V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha[R(s_{t+1}) + \gamma V(s_{t+2}) - V(s_{t+1})]$

+also perform: $V(s_t) \leftarrow V(s_t) + \alpha\gamma\lambda\delta_{t+1}$

- t+2: $V(s_{t+2}) \leftarrow V(s_{t+2}) + \alpha[R(s_{t+2}) + \gamma V(s_{t+3}) - V(s_{t+2})]$

+also:
$$V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha\gamma\lambda\delta_{t+2}$$
$$V(s_t) \leftarrow V(s_t) + \alpha\gamma^2\lambda^2\delta_{t+2}$$

TD(λ) --- backward view wordy

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{[R(s_t) + \gamma V(s_{t+1}) - V(s_t)]}_{\delta_t}$$

Similarly, the update at the next time step is

$$V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha \underbrace{(R(s_{t+1}) + \gamma V(s_{t+2}) - V(s_{t+1})}_{\delta_{t+1}}$$

Note that at the next time step we update $V(s_{t+1})$. This (crudely speaking) results in having a better estimate of the value function for state s_{t+1}. TD(λ) takes advantage of the availability of this better estimate to improve the update we performed for $V(s_t)$ in the previous step of the algorithm. Concretely, TD(λ) performs another update on $V(s_t)$ to account for our improved estimate of $V(s_{t+1})$ as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha\gamma\lambda\delta_{t+1}$$

where λ is a fudge factor that determines how heavily we weight changes in the value function for s_{t+1}.

Similarly, at time $t+2$ we perform the following set of updates:

$$V(s_{t+2}) \leftarrow V(s_{t+2}) + \alpha \underbrace{[R(s_{t+2}) + \gamma V(s_{t+2}) - V(s_{t+3})]}_{\delta_{t+2}} \ldots$$

$$V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha\gamma\lambda\delta_{t+2}$$

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\gamma^2\lambda^2}_{e(s_t)} \delta_{t+2}$$

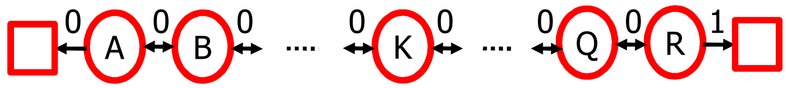
The term $e(s_t)$ is called the *eligibility vector*.

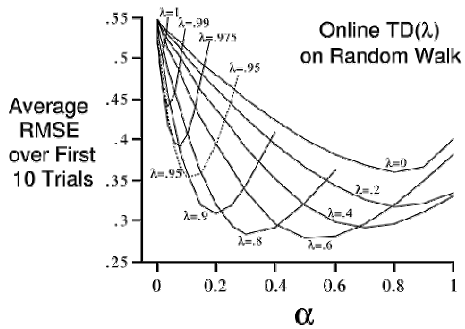
TD(λ)

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        $a \leftarrow$ action given by π for s
        Take action a, observe reward, r, and next state, s'
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For all s:
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
        $s \leftarrow s'$
    until s is terminal

TD(λ) --- example

Random walk over 19 states. Left and rightmost states are sinks. Rewards always zero, except when entering right sink.





TD(λ) --- "forward view"

- TD: $V(s_t) \leftarrow (1-\alpha)V(s_t) + \alpha \ \text{sample}$
- Sample = $R(s_t) + \gamma V(s_{t+1})$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 V(s_{t+2})$$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \gamma^3 V(s_{t+3})$$

$$\ldots$$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \ldots + \gamma^T R(s_T)$$

- $\lambda \in$ [0,1]
- Forward view equivalent to backward view

Sarsa(λ)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

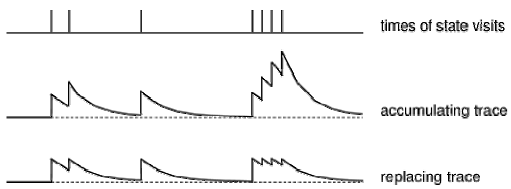
Watkins Q(λ)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $a^* \leftarrow \arg\max_b Q(s', b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
            else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

Replacing traces

- What if a state is visited at two different times t_1 and t_2 ?
- Recall TD(λ)

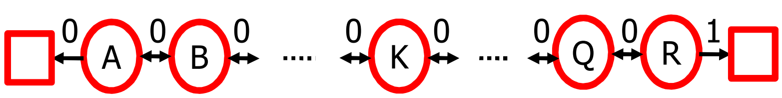
Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

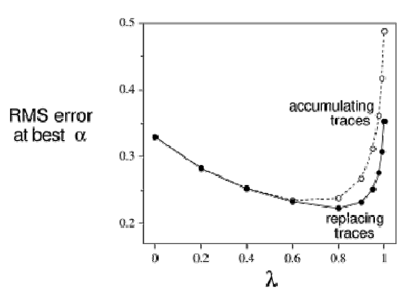


$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ 1 & \text{if } s = s_t. \end{cases}$$

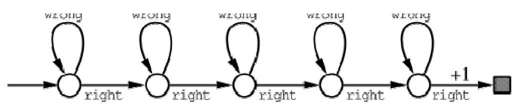
Replacing traces: example 1

Random walk over 19 states. Left and rightmost states are sinks. Rewards always zero, except when entering right sink.





Replacing traces: example 2



Recap RL so far

- When model is available:
  - VI, PI, GPI, LP
- When model is not available:
  - Model-based RL: collect data, estimate model, run one of the above methods for estimated model
  - Model-free RL: learn V, Q directly from experience:
    - TD(λ), sarsa(λ), Q(λ)

- What about large MDPs for which we cannot represent all states in memory or cannot collect experience from all states?

  → Function Approximation

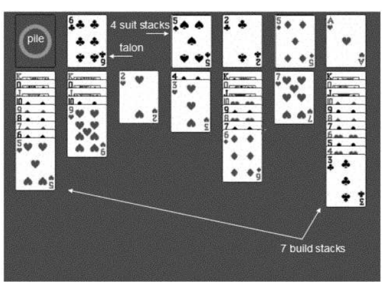
CS 287: Advanced Robotics
Fall 2009

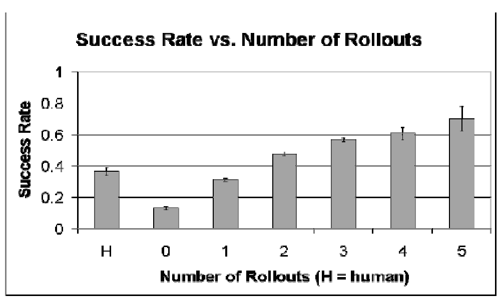
Lecture 14: Reinforcement Learning with Function Approximation and TD Gammon case study

Pieter Abbeel
UC Berkeley EECS

Assignment #1

- **Roll-out**: nice example paper: X. Yan, P. Diaconis, P. Rusmevichientong, and B. Van Roy, ``Solitaire: Man Versus Machine,'' *Advances in Neural Information Processing Systems 17*, MIT Press, 2005.





Recap RL so far

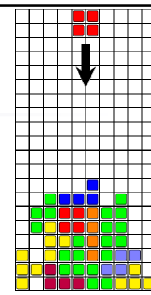
- When model is available:
  - VI, PI, GPI, LP
- When model is not available:
  - Model-based RL: collect data, estimate model, run one of the above methods for estimated model
  - Model-free RL: learn V, Q directly from experience:
    - TD(λ), sarsa(λ): on policy updates
    - Q: off policy updates
- What about large MDPs for which we cannot represent all states in memory or cannot collect experience from all states?

  → Function Approximation

Generalization and function approximation

- Represent the value function using a parameterized function $V_\theta(s)$, e.g.:
  - Neural network: θ is a vector with the weights on the connections in the network
  - Linear function approximator: $V_\theta(s) = \theta^\top \phi(s)$
    - Radial basis functions $\phi_i(s) = \exp\left(\frac{1}{2}(s - s_i)^\top \Sigma^{-1}(s - s_i)\right)$
    - Tilings: (often multiple) partitions of the state space
    - Polynomials: $\phi_i(s) = x_1^{j_1^i} x_2^{j_2^i} \ldots x_n^{j_n^i}$
    - Fourier basis $\{1, \sin(2\pi \frac{x_1}{L_1}), \cos(2\pi \frac{x_1}{L_1}), \sin(2\pi \frac{x_2}{L_2}), \cos(2\pi \frac{x_2}{L_2}), \ldots\}$
    - [Note: most algorithms kernelizable]
    - Often also specifically designed for the problem at hand

Example: tetris



- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece

$V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s)$

- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0*, . . . , 9, mapping the state to the height h[k] of each of the ten* columns.
  - Nine basis functions, 10*, . . . , 18, each mapping the state to the absolute difference* between heights of successive columns: *|h[k+1] – h[k]|, k = 1, . . . , 9.*
  - One basis function, 19, that maps state to the maximum column height: $\max_k$ *h[k]*
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD);Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Objective

- A standard way to find θ in supervised learning, optimize MSE:

$$\min_\theta \sum_s P(s)\left(V(s) - V_\theta(s)\right)^2$$

- Is this the correct objective?

Evaluating the objective

- When performing policy evaluation, we can obtain samples by simply executing the policy and recording the empirical (discounted) sum of rewards

$$\min_\theta \sum_{\text{encountered states } s} \left(\hat{V}^\pi(s) - V_\theta^\pi(s)\right)^2$$

- TD methods: use $v_t = r_t + \gamma V_\theta^\pi(s_{t+1})$ as a substitute for $V^\pi(s)$

Stochastic gradient descent

- Stochastic gradient descent to optimize MSE objective:
  - Iterate
    - Draw a state s according to P
    - Update:

$$\theta \leftarrow \theta - \frac{1}{2}\alpha \nabla_\theta \left(V^\pi(s) - V_\theta^\pi(s)\right)^2 = \theta + \alpha(V^\pi(s) - V_\theta^\pi(s))\nabla_\theta V_\theta^\pi(s)$$

- TD(0): use $v_t = r_t + \gamma V_\theta^\pi(s_{t+1})$ as a substitute for $V^\pi(s)$

$$\theta_{t+1} \leftarrow \theta_t + \alpha \left(R(s_t, a_t, s_{t+1}) + \gamma V_{\theta_t}^\pi(s_{t+1}) - V_{\theta_t}^\pi(s_t)\right) \nabla_\theta V_{\theta_t}^\pi(s_t)$$

TD(λ) with function approximation

- time t:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \underbrace{\left(R(s_t, a_t, s_{t+1}) + \gamma V^{\pi}_{\theta_t}(s_{t+1}) - V^{\pi}_{\theta_t}(s_t)\right)}_{\delta_t} \underbrace{\nabla_\theta V^{\pi}_{\theta_t}(s_t)}_{e_t}$$

- time t+1:

$$\theta_{t+2} \leftarrow \theta_{t+1} + \alpha \underbrace{\left(R(s_{t+1}, a_{t+1}, s_{t+2}) + \gamma V^{\pi}_{\theta_{t+1}}(s_{t+2}) - V^{\pi}_{\theta_{t+1}}(s_{t+1})\right)}_{\delta_{t+1}} \nabla_\theta V^{\pi}_{\theta_{t+1}}(s_{t+1})$$

$\theta_{t+2} \leftarrow \theta_{t+2} + \alpha \delta_{t+1} \gamma \lambda e_t$ ["improving previous update"]

Combined:

$$\begin{aligned} \delta_{t+1} &= R(s_{t+1}, a_{t+1}, s_{t+2}) + \gamma V^{\pi}_{\theta}(s_{t+2}) - V^{\pi}_{\theta_{t+1}}(s_{t+1}) \\ e_{t+1} &= \gamma \lambda e_t + \nabla_{\theta_{t+1}} V^{\pi}_{\theta_{t+1}}(s_{t+1}) \\ \theta_{t+2} &= \theta_{t+1} + \alpha \delta_{t+1} e_{t+1} \end{aligned}$$

TD(λ) with function approximation

```
Initialize θ⃗ arbitrarily
Repeat (for each episode):
   e⃗ = 0
   s ← initial state of episode
   Repeat (for each step of episode):
      a ← action given by π for s
      Take action a, observe reward, r, and next state, s'
      δ ← r + γV(s') − V(s)
      e⃗ ← γλe⃗ + ∇_θ⃗ V(s)
      θ⃗ ← θ⃗ + αδe⃗
      s ← s'
   until s is terminal
```

Can similarly adapt sarsa(λ) and Q(λ) eligibility vectors for function approximation

Guarantees

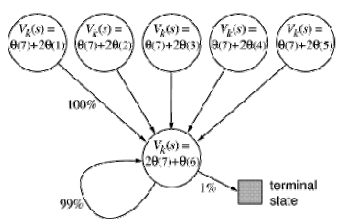
- Monte Carlo based evaluation:
  - Provides unbiased estimates under current policy
  - Will converge to true value of current policy
- Temporal difference based evaluations:
  - **TD(λ) w/linear function approximation: [Tsitsiklis and Van Roy, 1997]** ***If*** samples are generated from traces of execution of the policy π, and for appropriate choice of step-sizes α, TD(λ) converges and at convergence: [D = expected discounted state visitation frequencies under policy π]

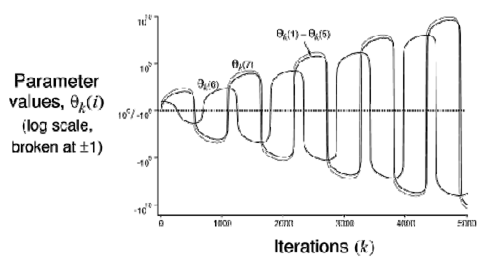
$$\|V_\theta - V^*\|_D \leq \frac{1-\lambda\gamma}{1-\gamma}\|\Pi_D V^* - V^*\|_D$$

  - **Sarsa(λ) w/linear function approximation:** same as TD
  - **Q w/linear function approximation: [Melo and Ribeiro, 2007]** Convergence to "reasonable" Q value under certain assumptions, including: $\forall s, a \|\phi(s,a)\|_1 \leq 1$
  - [Could also use infinity norm contraction function approximators to attain convergence --- see earlier lectures. However, this class of function approximators tends to be more restrictive.]

Off-policy counterexamples

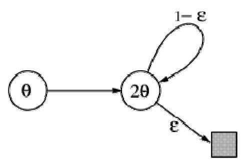
- Baird's counterexample for off policy updates:





Off-policy counterexamples

- Tsitsiklis and Van Roy counterexample: complete back-up with “off-policy” linear regression [i.e., uniform least squares, rather than waited by state visitation rates]



Intuition behind TD(0) with linear function approximation guarantees

- Stochastic approximation of the following operations:
  - Back-up: $(T^\pi V)(s) = \sum_{s'} T(s,\pi(s),s')\,[R(s,\pi(s),s') + \gamma V(s')]$
  - Weighted linear regression: $\min_\theta \sum_s D(s)((T^\pi V)(s) - \theta^\top \phi(s))^2$

    with solution: $\Phi\theta = \underbrace{\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D}_{\Pi_D}(T^\pi V)$
- Key observations:

$$\forall V_1, V_2 \ : \|T^\pi V_1 - T^\pi V_2\|_D \leq \gamma \|V_1 - V_2\|_D, \quad \text{here} : \|x\|_D = \sqrt{\sum_i D(i)x(i)^2}$$

$$\forall V_1, V_2 \ : \|\Pi_D V_1 - \Pi_D V_2\|_D \leq \|V_1 - V_2\|_D$$

Intuition behind TD(λ) guarantees

- Bellman operator:

$$(T^\pi J)(s) = \sum_{s'} P(s'|s,\pi(s)) \left[g(s) + \gamma J(s')\right] = \mathrm{E}\left[g(s) + \gamma J(s')\right]$$

- T$^\lambda$ operator:

$$T^\lambda J(s) = (1-\lambda) \sum_{m=0}^{\infty} \lambda^m \mathrm{E}\left[\sum_{k=0}^{m} \gamma^t g(s_k) + \gamma^{m+1} J(s_{m+1})\right]$$

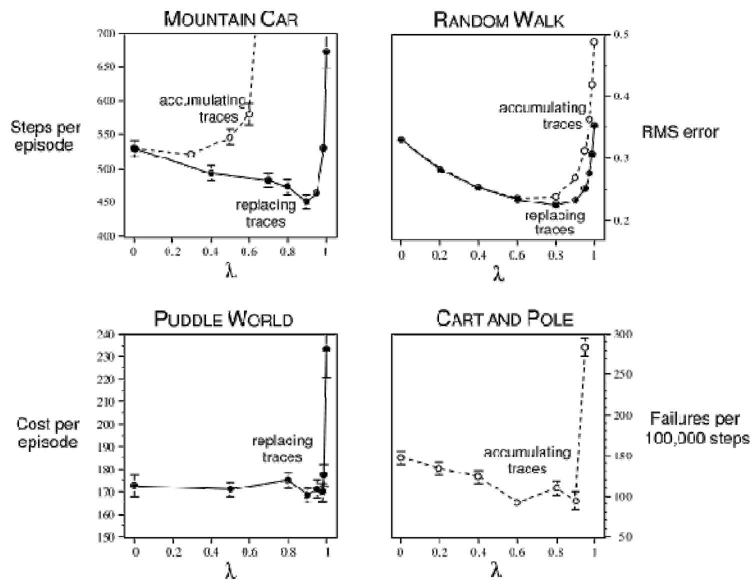
- T$^\lambda$ operator is contraction w.r.t. \| \|_D   for all $\lambda \in$ [0,1]

Should we use TD than well Monte Carlo?

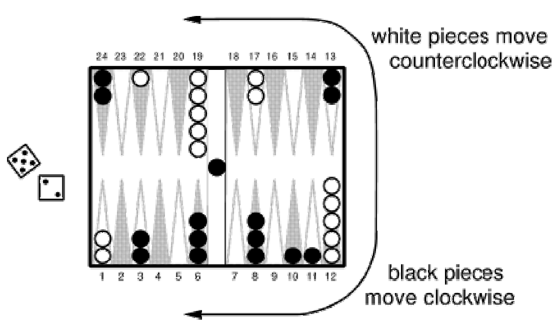
- At convergence: $\|V_\theta - V^*\|_D \leq \frac{1-\lambda\gamma}{1-\gamma} \|\Pi_D V^* - V^*\|_D$

Empirical comparison

(See, Sutton&Barto p.221 for details)

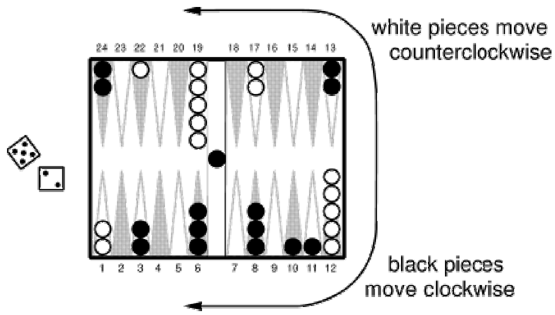


Backgammon



- 15 pieces, try go reach "other side"
- Move according to roll of dice
- If hitting an opponent piece: it gets reset to the middle row
- Cannot hit points with two or more opponent pieces

Backgammon

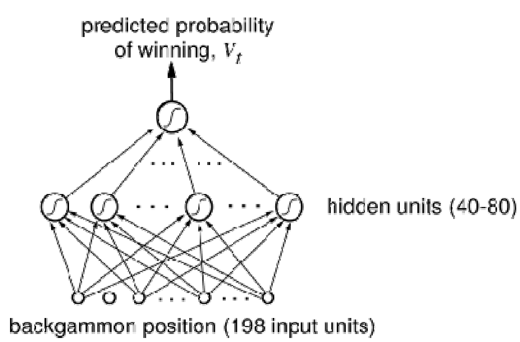


- 30 pieces, 24+2 possible locations
- For typical state and dice roll: often 20 moves

TD Gammon [Tesauro 92,94,95]

- Reward = 1 for winning the game

  = 0 other states
- Function approximator: 2 layer neural network



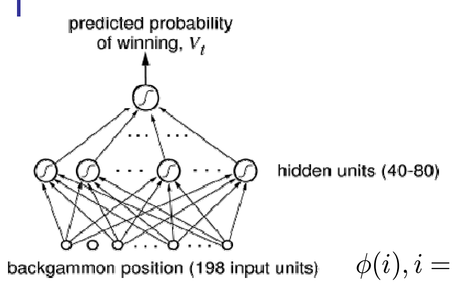
Input features

- For each point on the backgammon board, 4 input units indicate the number of white pieces as follows:
  - 1 piece → unit1=1;
  - 2 pieces → unit1=1, unit2=1;
  - 3 pieces → unit1=1, unit2=1, unit3=1;
  - n>3 pieces → unit1=1, unit2=1, unit3=1, unit4 = (n-3)/2
- Similarly for black

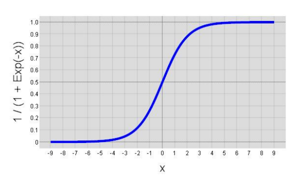
[This already makes for 2*4*24 = 192 input units.]

- Last six: number of pieces on the bar (w/b), number of pieces that completed the game (w/b), white's move, black's move

Neural net



$\phi(i), i = 1, \ldots, 198$



- Each hidden unit computes:

$$h(j) = \sigma(\textstyle\sum_i w_{ij}\phi(i)) = \frac{1}{1+\exp\left(-\sum_i w_{ij}\phi(i)\right)}$$

- Output unit computes:

$$o = \sigma(\textstyle\sum_j w_j h(j)) = \frac{1}{1+\exp\left(-\sum_j w_j h(j)\right)}$$

- Overall: $o = f(\phi(1), \ldots, \phi(198); w)$

Neural nets

- Popular at that time for function approximation / learning in general
- Derivatives/Gradients are easily derived analytically
  - Turns out they can be computed through backward error propagation --- name “error backpropagation”
- Susceptible to local optima!

Learning

- Initialize weights randomly
- TD(λ) [λ = 0.7, α = 0.1]
- Source of games: self-play, greedy w.r.t. current value function [results suggest game has enough stochasticity built in for exploration purposes]

Results

- After 300,000 games as good as best previous computer programs
  - Neurogammon: highly tuned neural network trained on large corpus of exemplary moves
- TD Gammon 1.0: add Neurogammon features
  - Substantially better than all previous computer players; human expert level
- TD Gammon 2.0, 2.1: selective 2-ply search
- TD Gammon 3.0: selective 3-ply search, 160 hidden units

CS 287: Advanced Robotics
Fall 2009

Lecture 15: LSTD, LSPI, RLSTD, imitation learning

Pieter Abbeel
UC Berkeley EECS

TD(0) with linear function approximation guarantees

- Stochastic approximation of the following operations:
  - Back-up: $(T^\pi V)(s) = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s')\right]$
  - Weighted linear regression: $\min_\theta \sum_s D(s)((T^\pi V)(s) - \theta^\top \phi(s))^2$
- Batch version (for large state spaces):
  - Let {(s,a,s')} have been sampled according to D
  - Iterate:
    - Back-up for sampled (s,a,s'): $V(s) \leftarrow [R(s,a,s') + \gamma V(s')] = [R(s,a,s') + \gamma \theta^\top \phi(s')]$
    - Perform regression: $\min_\theta \sum_{(s,a,s')} (V(s) - \theta^\top \phi(s))^2$

$$\min_\theta \sum_{(s,a,s')} (R(s,a,s') + \gamma {\theta^{(\text{old})}}^\top \phi(s') - \theta^\top \phi(s))^2$$

TD(0) with linear function approximation guarantees

- Iterate:

$$\theta^{(\text{new})} = \arg\min_\theta \sum_{(s,a,s')} (R(s,a,s') + \gamma {\theta^{(\text{old})}}^\top \phi(s') - \theta^\top \phi(s))^2$$

- Can we find the fixed point directly?
  - Rewrite the least squares problem in matrix notation:

$$\theta^{(\text{new})} = \arg\min_\theta \|R + \gamma \Phi' \theta^{(\text{old})} - \Phi\theta\|_2^2$$

  - Solution: $\theta^{(\text{new})} = (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta^{(\text{old})})$

TD(0) with linear function approximation guarantees

- Solution: $\theta^{(\text{new})} = (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta^{(\text{old})})$
- Fixed point?

$$\begin{aligned}
\theta &= (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta) \\
(\Phi^\top \Phi)\theta &= \Phi^\top (R + \gamma \Phi' \theta) \\
\left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)\theta &= \Phi^\top R \\
\theta &= \left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)^{-1} \Phi^\top R
\end{aligned}$$

LSTD(0)

- Collect state-action-state triples (s_i,a_i,s'_i) according to a policy π
- Build the matrices:

$$\Phi = \begin{bmatrix} \phi(s_1)^\top \\ \phi(s_2)^\top \\ \dots \\ \phi(s_m)^\top \end{bmatrix}, \Phi' = \begin{bmatrix} \phi(s'_1)^\top \\ \phi(s'_2)^\top \\ \dots \\ \phi(s'_m)^\top \end{bmatrix}, R = \begin{bmatrix} R(s_1, a_1, s'_1) \\ R(s_2, a_2, s'_2) \\ \dots \\ R(s_m, a_m, s'_m) \end{bmatrix}$$

- Find an approximation of the value function

$$V^\pi(s) \approx \theta^\top \phi(s)$$

$$\text{for} \quad \theta = \left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)^{-1} \Phi^\top R$$

LSTD(0) in policy iteration

- Iterate
  - Collect state-action-state triples (s_i,a_i,s'_i) according to current policy π
  - Use LSTD(0) to compute V^π
- Tweaks:
  - Can re-use triples (s_i,a_i,s'_i) from previous policies as long as they are consistent with the current policy
  - Can redo the derivation with Q functions rather than V
  - In case of stochastic policies, can weight contribution of a triple according to Prob(a_i|s_i) under the current policy

  → Doing all three results in "***Least squares policy iteration***," (Lagoudakis and Parr, 2003).

LSTD(0) --- batch vs. incremental updates

- Collect state-action-state triples (s_i,a_i,s'_i) according to a policy π
- Build the matrices:

$$\Phi_m = \begin{bmatrix} \phi(s_1)^\top \\ \phi(s_2)^\top \\ \dots \\ \phi(s_m)^\top \end{bmatrix}, \Phi'_m = \begin{bmatrix} \phi(s'_1)^\top \\ \phi(s'_2)^\top \\ \dots \\ \phi(s'_m)^\top \end{bmatrix}, R_m = \begin{bmatrix} R(s_1, a_1, s'_1) \\ R(s_2, a_2, s'_2) \\ \dots \\ R(s_m, a_m, s'_m) \end{bmatrix}$$

- Find an approximation of the value function

$$V^\pi(s) \approx \theta_m^\top \phi(s)$$

$$\text{for} \quad \theta_m = \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m)\right)^{-1} \Phi_m^\top R_m$$

- One more datapoint → "m+1" :

$$\theta_{m+1} = \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m) + \phi_{m+1}(\phi_m - \gamma\phi'_m)^\top\right)^{-1} \left(\Phi_m^\top R_m + \phi_{m+1} r_{m+1}\right)$$

Sherman-Morrison formula: $(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}.$

RLSTD

- Recursively compute approximation of the value function by leveraging the Sherman-Morrison formula
-
$$\begin{aligned} A_m^{-1} &= \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m)\right)^{-1} \\ b_m &= \Phi_m R_m \\ \theta_m &= A_m^{-1} b_m \end{aligned}$$

- One more datapoint → "m+1" :

$$\begin{aligned} A_{m+1}^{-1} &= A_m^{-1} - \frac{A_m^{-1}\phi_{m+1}(\phi_{m+1} - \gamma\phi'_{m+1})^\top A_m^{-1}}{1 + (\phi_{m+1} - \gamma\phi'_{m+1})^\top A_m^{-1}\phi_{m+1}} \\ b_{m+1} &= b_m + \phi_{m+1} r_{m+1} \end{aligned}$$

- Note: there exist orthogonal matrix techniques to do the same thing but in a numerically more stable fashion (essentially: keep track of the QR decomposition of A_m)

RLSTD: for non-linear function approximators?

- RLSTD with linear function approximation with a Gaussian prior on \theta

  → Kalman filter

- Can be applied to non-linear setting too: simply linearize the non-linear function approximator around the current estimate of \theta; not globally optimal, but likely still better than "naïve" gradient descent

  (+prior → Extended Kalman filter)

Recursive Least Squares (1)

Growing sets of measurements

least-squares problem in 'row' form:

$$\text{minimize} \quad \|Ax - y\|^2 = \sum_{i=1}^{m} (a_i^T x - y_i)^2$$

where a_i^T are the rows of A ($a_i \in \mathbf{R}^n$)

- $x \in \mathbf{R}^n$ is some vector to be estimated
- each pair a_i, y_i corresponds to one measurement
- solution is

$$x_{\text{ls}} = \left(\sum_{i=1}^{m} a_i a_i^T \right)^{-1} \sum_{i=1}^{m} y_i a_i$$

- suppose that a_i and y_i become available sequentially, *i.e.*, m increases with time

[From: Boyd, ee263]

Recursive Least Squares (2)

Recursive least-squares

we can compute $x_{\mathrm{ls}}(m) = \left(\sum_{i=1}^{m} a_i a_i^T\right)^{-1} \sum_{i=1}^{m} y_i a_i$ recursively

- initialize $P(0) = 0 \in \mathbf{R}^{n\times n}$, $q(0) = 0 \in \mathbf{R}^n$
- for $m = 0, 1, \ldots,$

$$P(m+1) = P(m) + a_{m+1}a_{m+1}^T \qquad q(m+1) = q(m) + y_{m+1}a_{m+1}$$

- if $P(m)$ is invertible, we have $x_{\mathrm{ls}}(m) = P(m)^{-1}q(m)$
- $P(m)$ is invertible $\Longleftrightarrow$ $a_1, \ldots, a_m$ span $\mathbf{R}^n$
  (so, once $P(m)$ becomes invertible, it stays invertible)

[From: Boyd, ee263]

Recursive Least Squares (3)

Fast update for recursive least-squares

we can calculate

$$P(m+1)^{-1} = \left(P(m) + a_{m+1}a_{m+1}^T\right)^{-1}$$

efficiently from $P(m)^{-1}$ using the *rank one update formula*

$$\left(P + aa^T\right)^{-1} = P^{-1} - \frac{1}{1 + a^T P^{-1} a}(P^{-1}a)(P^{-1}a)^T$$

valid when $P = P^T$, and P and $P + aa^T$ are both invertible

- gives an $O(n^2)$ method for computing $P(m+1)^{-1}$ from $P(m)^{-1}$
- standard methods for computing $P(m+1)^{-1}$ from $P(m+1)$ are $O(n^3)$

[From: Boyd, ee263]

TD methods recap

- Model-free RL: learn V, Q directly from experience:
  - TD(λ), sarsa(λ): on policy updates
  - Q: off policy updates
- Large MDPs: include function Approximation
  - Some guarantees for linear function approximation
- Batch version
  - No need to tweak various constants
  - Same solution can be obtained incrementally by using recursive updates! This is generally true for least squares type systems.

Applications of TD methods

- Backgammon
- Standard RL testbeds (all in simulation)
  - Cartpole balancing
  - Acrobot swing-up
  - Gridworld --- Assignment #2
  - Bicycle riding
  - Tetris --- Assignment #2
- As part of actor-critic methods (=policy gradient + TD)
  - Fine-tuning / Learning some robotics tasks
- *Many financial institutions use some linear TD for pricing of options*

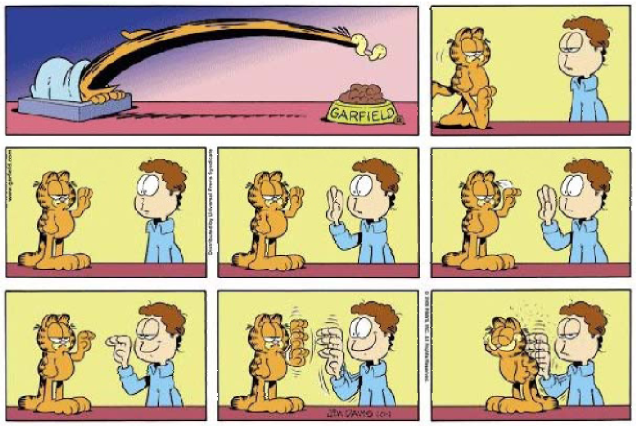
RL: our learning status

- Small MDPs: VI, PI, GPI, LP
- Large MDPs:
  - Value iteration + function approximation
    - Iterate: Bellman back-up, project, …
  - TD methods:
    - TD, sarsa, Q with function approximation
      - Simplicity, limited storage can be a convenience
    - LSTD, LSPI, RLSTD
    - Built upon in and compared to in many current RL papers
    - Main current direction: feature selection
- You should be able to read/understand many RL papers
- Which important ideas are we missing (and will I try to cover between today and the next 3-5 lectures) ?

- Imitation learning
  - Learn from observing an expert
- Linear programming w/function approximation and constraint sampling
  - Guarantees, Generally applicable idea of constraint sampling
- Policy gradient, Actor-Critic (=TD+policy gradient in one)
  - Fine tuning policies through running trials on a real system, Robotic success stories
- Partial observability
  - POMDPS
- Hierarchical methods
  - Incorporate your knowledge to enable scaling to larger systems
- Reward shaping
  - Can we choose reward functions such as to enable faster learning?
- Exploration vs. exploitation
  - How/When should we explore?
- Stochastic approximation
  - Basic intuition behind how/when sampled versions work?

Imitation learning

to program others



Imitation learning: what to learn?

- If expert available, could use expert trace s_1, a_1, s_2, a_2, s_3, a_3, … to learn "something" from the expert
  - Behavioral cloning: use supervised learning to directly learn a policy S→A.
    - No model of the system dynamics required
    - No MDP / optimal control solution algorithm required
  - Inverse reinforcement learning:
    - Learn the reward function
    - Often most compact and transferrable task description
  - Trajectory primitives:
    - Use expert trajectories as motion primitives / components for motion planning
    - Use expert trajectories as starting points for trajectory optimization

Behavioral cloning

- If expert available, could use expert trace $s_1, a_1, s_2, a_2, s_3, a_3, \ldots$ to learn the expert policy $\pi : S \rightarrow A$
- Class of policies to learn:
  - Neural net, decision tree, linear regression, logistic regression, svm, deep belief net, …
- Advantages:
  - No model of the system dynamics required
  - No MDP / optimal control solution algorithm required
- Minuses:
  - Only works if we can come up with a good policy class
    - Typically more applicable to "reactive" tasks, less so to tasks that involve planning
  - No leveraging of dynamics model if available.

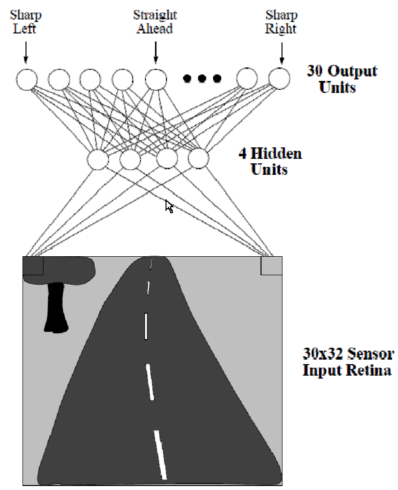
Alvinn

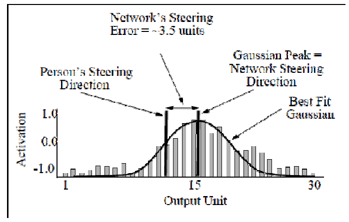
- Task: steer a vehicle

CMU Navlab Autonomous Navigation Testbed

- Input: 30x32 image.

Alvinn

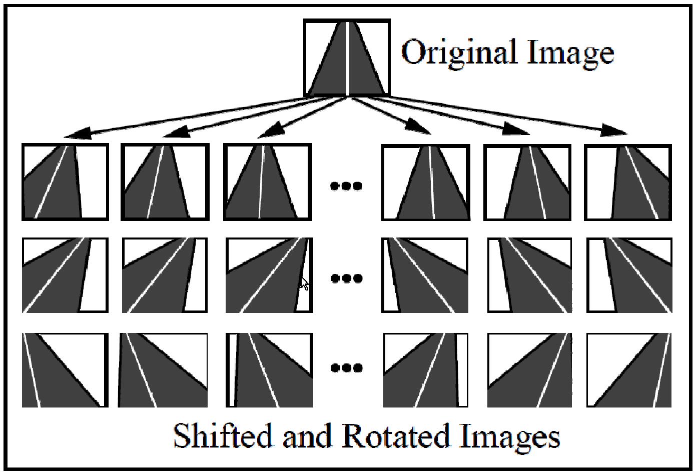




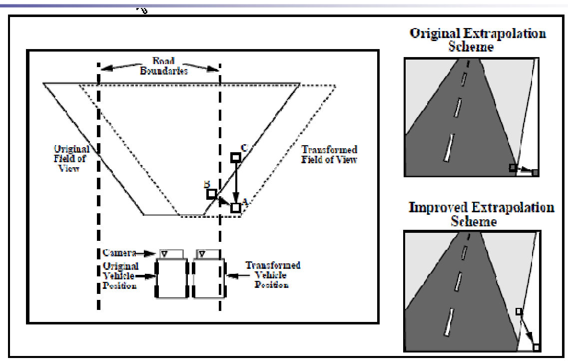
Richness of training data?

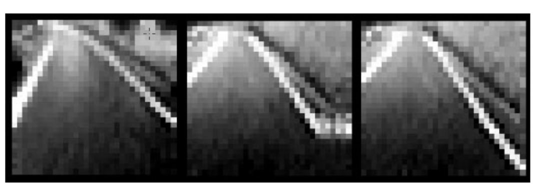
- Training data from good driver does not well represent situations from which it should be able to recover
- Might over-train on the "simple" data
- Solution? Intentionally swerve off-center?
  - Issues:
    - Inconvenience to switch on/off the learning
    - Might require a lot of swerving (which could be especially undesirable in traffic)

Transformed images



Transformed images



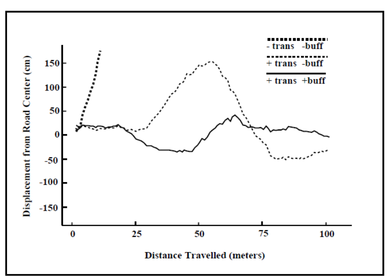


original extrap1 extrap2

Few other details

- Steering direction for transformed images:
  - “pure pursuit model” : constant steering arc will bring it back in the center at distance T
- Image buffering:
  - Keeps 200 images in buffer
  - One backpropagation pass over all images in each round of training
  - Replacement to favor neutral steering
- Road types:

Results



- Achieved 98.2% autonomous driving on a 5000 km (3000-mile) "No hands across America" trip.
- Throttle and brakes were human-controlled.

- Note: other autonomous driving projects:
  - Ernst Dickmanns
  - Darpa Grand and Urban Challenge

Sammut+al, Learning to fly (ICML1992)

- Task (in Silicon Graphics Flight Sim)
  - (crudely) Take off, fly through some waypoints, land
- Training data: 30 flights (/pilot)
- Recorded features: *on_ground, g_limit exceeded, wing_stall, twist* , *elevation*, *azimuth, roll_speed, elevation_speed, azimuth_speed, airspeed, climbspeed, E/W distance* from centre of runway, *altitude, N/S distance* from northern end of runway, *fuel, rollers, elevator, rudder, thrust*, *flaps*
- Data from each flight segmented into seven stages
- In each stage: Four separate decision trees (C4.5), one for each of the elevator, rollers, thrust and flaps.
- Succeeded in synthesizing control rules for a complete flight, including a safe landing. The rules **fly** the Cessna in a manner very similar to that of the pilot whose data were used to construct the rules.
- Pilots who are frugal in their use of the controls give few examples of what to do when things go wrong.

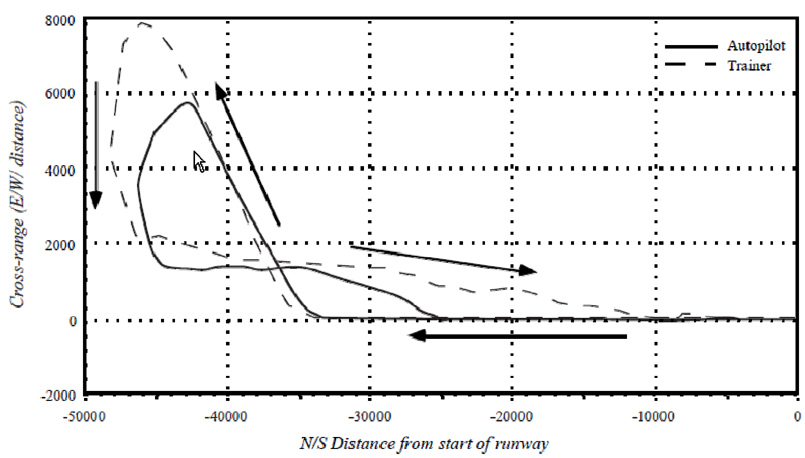
7 stages

- 1. Take off and fly to an altitude of 2,000 feet.
- 2. Level out and fly to a distance of 32,000 feet from the starting point.
- 3. Turn right to a compass heading of approximately 330°. The subjects were actually told to head toward a particular point in the scenery that corresponds to that heading.
- 4. At a North/South distance of 42,000 feet, turn left to head back towards the runway. The scenery contains grid marks on the ground. The starting point for the turn is when the last grid line was reached. This corresponds to about 42,000 feet. The turn is considered complete when the azimuth is between 140° and 180°.
- 5. Line up on the runway. The aircraft was considered to be lined up when the aircraft's azimuth is less than 5° off the heading of the runway and the twist is less that ±10° from horizontal.
- 6. Descend to the runway, keeping in line. The subjects were given the hint that they should have an ‘aiming point’ near the beginning of the runway.
- 7. Land on the runway.

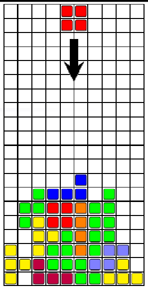
Sammut + al

- Example decision tree:
- Stage 3: Turn right to a compass heading of approximately 330°
  - twist <= -23 : left_roll_3
  - twist > -23 :
    - | azimuth <= -25 : no_roll
    - | azimuth > -25 : right_roll_2

Sammut+al



Tetris



- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece

$V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s)$

- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0*, . . . , 9, mapping the state to the height h[k] of each of the ten* columns.
  - Nine basis functions, 10*, . . . , 18, each mapping the state to the absolute difference* between heights of successive columns: *|h[k+1] – h[k]|, k = 1, . . . , 9.*
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD);Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

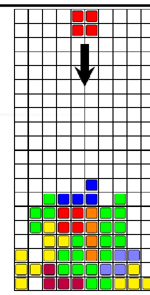
Behavioral cloning in tetris

CS 287: Advanced Robotics
Fall 2009

Lecture 16: imitation learning

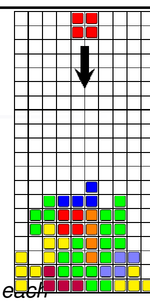
Pieter Abbeel
UC Berkeley EECS

Behavioral cloning example



- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece

Behavioral cloning example



$V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s)$

- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, *. . . , 9, mapping the state to the height h[k] of each of the ten* columns.
  - Nine basis functions, 10, *. . . , 18, each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 9$.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD);Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Behavioral cloning example

Behavioral cloning example

Behavioral cloning example

Training data: Example choices of next states chosen by the demonstrator: $s_+^{(i)}$

Alternative choices of next states that were available: $s_{j-}^{(i)}$

Max-margin formulation

$$\min_{\theta,\xi\geq 0} \quad \theta^\top\theta + C\sum_{i,j}\xi_{i,j}$$

$$\text{subject to} \quad \forall i, \forall j: \theta^\top\phi(s_+^{(i)}) \geq \theta^\top\phi(s_{j-}^{(i)}) + 1 - \xi_{i,j}$$

Probabilistic/Logistic formulation

Assumes experts choose for result $s^{(i)}$ with probability $\frac{\exp(\theta^\top\phi(s_+^{(i)}))}{\exp(\theta^\top\phi(s_+^{(i)}))+\sum_{j-}\exp(\theta^\top\phi(s_{j-}^{(i)}))}$.

Hence the maximum likelihood estimate is given by:

$$\max_\theta \sum_i \log\left(\frac{\exp(\theta^\top\phi(s_+^{(i)}))}{\exp(\theta^\top\phi(s_+^{(i)}))+\sum_{j-}\exp(\theta^\top\phi(s_{j-}^{(i)}))}\right) - C\|\theta\|$$

Motivation for inverse RL

- Scientific inquiry
  - Model animal and human behavior
    - E.g., bee foraging, songbird vocalization. [See intro of Ng and Russell, 2000 for a brief overview.]
- Apprenticeship learning/Imitation learning through inverse RL
  - Presupposition: reward function provides the most succinct and transferable definition of the task
  - Has enabled advancing the state of the art in various robotic domains
- Modeling of other agents, both adversarial and cooperative

Problem setup

- Input:
  - State space, action space
  - Transition model $P_{sa}(s_{t+1} \mid s_t, a_t)$
  - *No* reward function
  - Teacher's demonstration: $s_0, a_0, s_1, a_1, s_2, a_2, \ldots$
    (= trace of the teacher's policy π^*)
- Inverse RL:
  - Can we recover R ?
- Apprenticeship learning via inverse RL
  - Can we then use this R to find a good policy ?
- Vs. Behavioral cloning (which directly learns the teacher's policy using supervised learning)
  - Inverse RL: leverages compactness of the reward function
  - Behavioral cloning: leverages compactness of the policy class considered, does not require a dynamics model

Lecture outline

- Inverse RL intro
- *Mathematical formulations for inverse RL*
- Case studies

Three broad categories of formalizations

- Max margin
- Feature expectation matching
- Interpret reward function as parameterization of a policy class

Basic principle

- Find a reward function R^* which explains the expert behaviour.
- Find R^* such that $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi^*] \geq \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi] \quad \forall \pi$
- Equivalently, find R* such that

$$\sum_{s \in S} R(s) \left(\sum_{t=0}^{\infty} \gamma^t \mathbf{1}\{s_t = s|\pi^*\} \right) \geq \sum_{s \in S} R(s) \left(\sum_{t=0}^{\infty} \gamma^t \mathbf{1}\{s_t = s|\pi^*\} \right) \quad \forall \pi$$

- A convex feasibility problem in R*, but many challenges:
  - R=0 is a solution, more generally: reward function ambiguity
  - We typically only observe expert traces rather than the entire expert policy π^* --- how to compute LHS?
  - Assumes the expert is indeed optimal --- otherwise infeasible
  - Computationally: assumes we can enumerate all policies

X

Feature based reward function

- Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \rightarrow \Re^n$.

$$\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] =$$

Feature based reward function

- Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \rightarrow \Re^n$.

$$\begin{aligned} \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] &= \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t)|\pi] \\ &= w^\top \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \\ &= w^\top \underbrace{\mu(\pi)} \end{aligned}$$

Expected cumulative discounted sum of feature values or "feature expectations"

- Subbing into $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi^*] \geq \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi] \quad \forall \pi$

  gives us:

  Find w^* such that ${w^*}^\top \mu(\pi^*) \geq {w^*}^\top \mu(\pi) \quad \forall \pi$

Feature based reward function

$$\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi^*] \geq \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi] \quad \forall \pi$$

$$\Downarrow$$

Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \rightarrow \Re^n$.
Find w^* such that ${w^*}^\top \mu(\pi^*) \geq {w^*}^\top \mu(\pi) \quad \forall \pi$

- Feature expectations can be readily estimated from sample trajectories.
- The number of expert demonstrations required scales with the number of features in the reward function.
- The number of expert demonstration required does *not* depend on
  - Complexity of the expert's optimal policy π^*
  - Size of the state space

Recap of challenges

Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \rightarrow \Re^n$.
Find w^* such that ${w^*}^\top \mu(\pi^*) \geq {w^*}^\top \mu(\pi) \quad \forall \pi$

- Challenges:
  - Assumes we know the entire expert policy π^* → assumes we can estimate expert feature expectations
  - R=0 is a solution (now: w=0), more generally: reward function ambiguity
  - Assumes the expert is indeed optimal---became even more of an issue with the more limited reward function expressiveness!
  - Computationally: assumes we can enumerate all policies

X

Ambiguity

- We currently have: $\text{Find } w^* \text{ such that } {w^*}^\top \mu(\pi^*) \geq {w^*}^\top \mu(\pi) \quad \forall \pi$
- Standard max margin:
- "Structured prediction" max margin:

Ambiguity

- Standard max margin:

$$\min_w \|w\|_2^2$$
$$\text{s.t.} \quad w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + 1 \quad \forall \pi$$

- "Structured prediction" max margin:

$$\min_w \|w\|_2^2$$
$$\text{s.t.} \quad w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + m(\pi^*, \pi) \quad \forall \pi$$

- Justification: margin should be larger for policies that are very different from π*.
- Example: m(π, π*) = number of states in which π* was observed and in which π and π* disagree

X

Expert suboptimality

- Structured prediction max margin:

$$\min_w \|w\|_2^2$$
$$\text{s.t.} \quad w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + m(\pi^*, \pi) \qquad \forall \pi$$

Expert suboptimality

- Structured prediction max margin with slack variables:

$$\min_{w,\xi} \|w\|_2^2 + C\xi$$
$$\text{s.t.} \quad w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + m(\pi^*, \pi) - \xi \qquad \forall \pi$$

- Can be generalized to multiple MDPs (could also be same MDP with different initial state)

$$\min_{w,\xi^{(i)}} \|w\|_2^2 + C \sum_i \xi^{(i)}$$
$$\text{s.t.} \quad w^\top \mu(\pi^{(i)*}) \geq w^\top \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \qquad \forall i, \pi^{(i)}$$

Complete max-margin formulation

$$\min_w \|w\|_2^2 + C \sum_i \xi^{(i)}$$
$$\text{s.t.} \quad w^\top \mu(\pi^{(i)*}) \geq w^\top \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \qquad \forall i, \pi^{(i)}$$

[Ratliff, Zinkevich and Bagnell, 2006]

- Resolved: access to π*, ambiguity, expert suboptimality
- *One challenge remains: very large number of constraints*
  - *Ratliff+al use subgradient methods.*
  - *In this lecture: constraint generation*

Constraint generation

Initialize $\Pi^{(i)}$ = {} for all i and then iterate

- Solve

$$\min_w \|w\|_2^2 + C \sum_i \xi^{(i)}$$
$$\text{s.t.} \quad w^\top \mu(\pi^{(i)*}) \geq w^\top \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \qquad \forall i, \forall \pi^{(i)} \in \Pi^{(i)}$$

- For current value of w, find the most violated constraint for all i by solving:

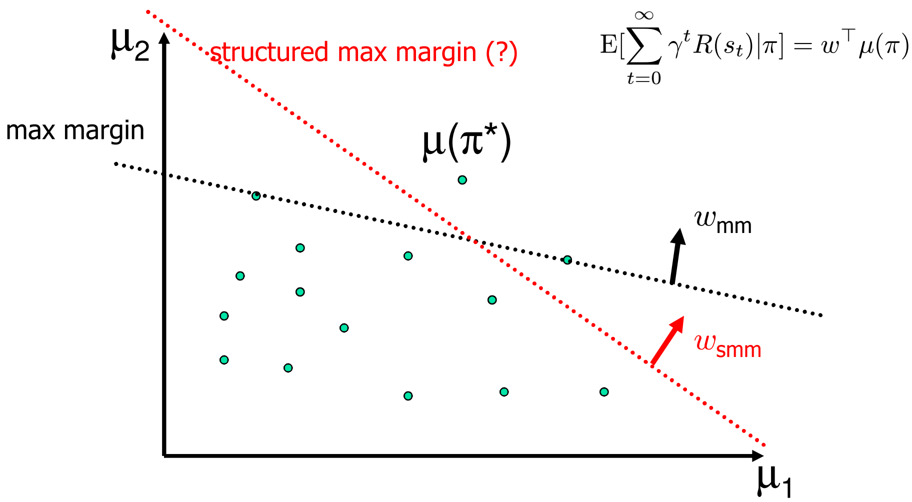
$$\max_{\pi^{(i)}} w^\top \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)})$$

  = find the optimal policy for the current estimate of the reward function (+ loss augmentation m)

- For all i add $\pi^{(i)}$ to $\Pi^{(i)}$
- If no constraint violations were found, we are done.

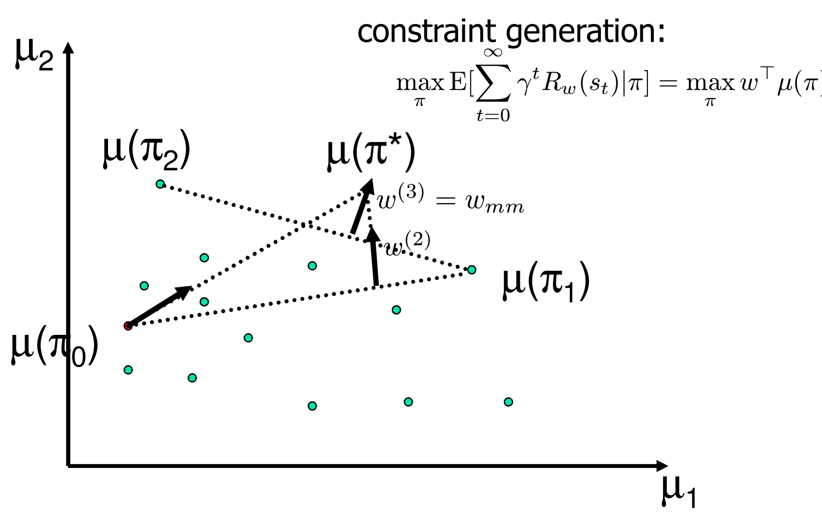
Visualization in feature expectation space

- Every policy π has a corresponding feature expectation vector $\mu(\pi)$, which for visualization purposes we assume to be 2D



Constraint generation

- Every policy π has a corresponding feature expectation vector $\mu(\pi)$, which for visualization purposes we assume to be 2D



Three broad categories of formalizations

- Max margin (Ratliff+al, 2006)
  - Feature boosting [Ratliff+al, 2007]
  - Hierarchical formulation [Kolter+al, 2008]

- *Feature expectation matching (Abbeel+Ng, 2004)*
  - *Two player game formulation of feature matching (Syed+Schapire, 2008)*
  - *Max entropy formulation of feature matching (Ziebart+al,2008)*

- Interpret reward function as parameterization of a policy class. (Neu+Szepesvari, 2007; Ramachandran+Amir, 2007)

Feature expectation matching

- Inverse RL starting point: find a reward function such that the expert outperforms other policies

Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \to \Re^n$.

Find w^* such that ${w^*}^\top \mu(\pi^*) \geq {w^*}^\top \mu(\pi) \quad \forall \pi$

- Observation in Abbeel and Ng, 2004: for a policy π to be guaranteed to perform as well as the expert policy π*, it suffices that the feature expectations match:

$$\|\mu(\pi) - \mu(\pi^*)\| \text{ small implies } \|{w^*}^\top \mu(\pi^*) - {w^*}^\top \mu(\pi)\| \text{ small}$$

→ How to find such a policy π ?

Feature expectation matching

- If expert suboptimal:
  - *Abbeel and Ng, 2004*: resulting policy is a mixture of policies which have expert in their convex hull---In practice: pick the best one of this set and pick the corresponding reward function.
  - *Syed and Schapire, 2008* recast the same problem in game theoretic form which, at cost of adding in some prior knowledge, results in having a unique solution for policy and reward function.
  - *Ziebart+al, 2008* assume the expert stochastically chooses between paths where each path's log probability is given by its expected sum of rewards.

Lecture outline

- Inverse RL intro
- Mathematical formulations for inverse RL
  - Max-margin
  - Feature matching
  - *Reward function parameterizing the policy class*
- Case studies

Reward function parameterizing the policy class

- Recall:

$$V^*(s;R) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s;R)$$
$$Q^*(s,a;R) = R(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s;R)$$

- Let's assume our expert acts according to:

$$\pi(a|s;R,\alpha) = \frac{1}{Z(s;R,\alpha)} \exp(\alpha Q^*(s,a;R))$$

- Then for any R and α, we can evaluate the likelihood of seeing a set of state-action pairs as follows:

$$P((s_1,a_1))\ldots P((s_m,a_m)) = \frac{1}{Z(s_1;R,\alpha)} \exp(\alpha Q^*(s_1,a_1;R)) \ldots \frac{1}{Z(s_m;R,\alpha)} \exp(\alpha Q^*(s_m,a_m;R))$$

- Note: non-convex formulation --- due to non-linear equality constraint for V!
- Ramachandran and Amir, AAAI2007: MCMC method to sample from this distribution
- Neu and Szepesvari, UAI2007: gradient method to find local optimum of the likelihood

Lecture outline

- Inverse RL intro
- Mathematical formulations for inverse RL
- *Case studies:*
  - *Highway driving,*
  - *Parking lot navigation,*
  - *Route inference,*
  - *Quadruped locomotion*

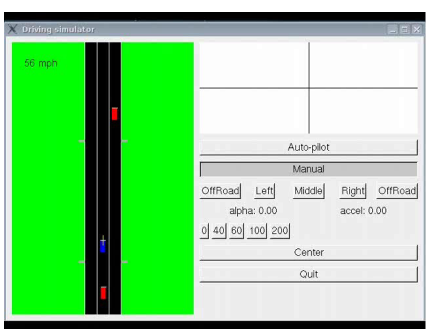
Simulated highway driving

Abbeel and Ng, ICML 2004; Syed and Schapire, NIPS 2007

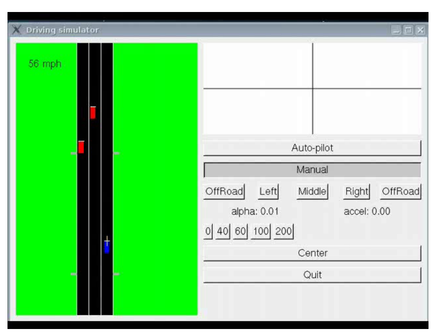
Highway driving

[Abbeel and Ng 2004]

Teacher in Training World



Learned Policy in Testing World

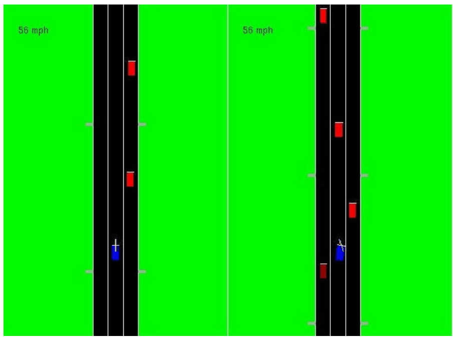


- Input:
  - Dynamics model / Simulator $P_{sa}(s_{t+1} \mid s_t, a_t)$
  - Teacher's demonstration: 1 minute in "training world"
  - Note: R* is unknown.
  - Reward features: 5 features corresponding to lanes/shoulders; 10 features corresponding to presence of other car in current lane at different distances

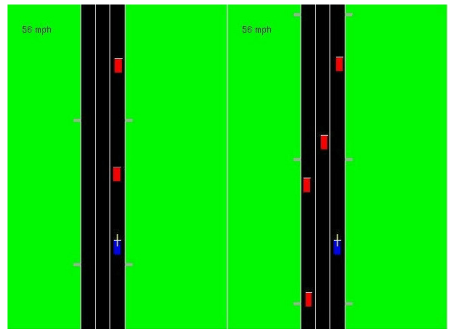
[Abbeel and Ng 2004]

More driving examples

Driving demonstration | Learned behavior



Driving demonstration | Learned behavior



In each video, the left sub-panel shows a demonstration of a different driving "style", and the right sub-panel shows the behavior learned from watching the demonstration.

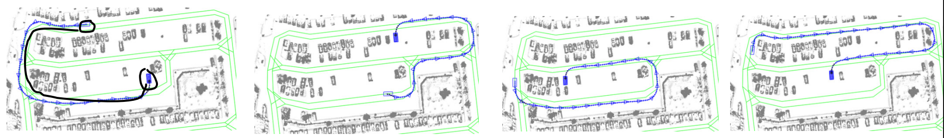
Parking lot navigation

- Reward function trades off:
  - Staying "on-road,"
  - Forward vs. reverse driving,
  - Amount of switching between forward and reverse,
  - Lane keeping,
  - On-road vs. off-road,
  - Curvature of paths.

[Abbeel et al., IROS 08]

Experimental setup

- Demonstrate parking lot navigation on “train parking lots.”



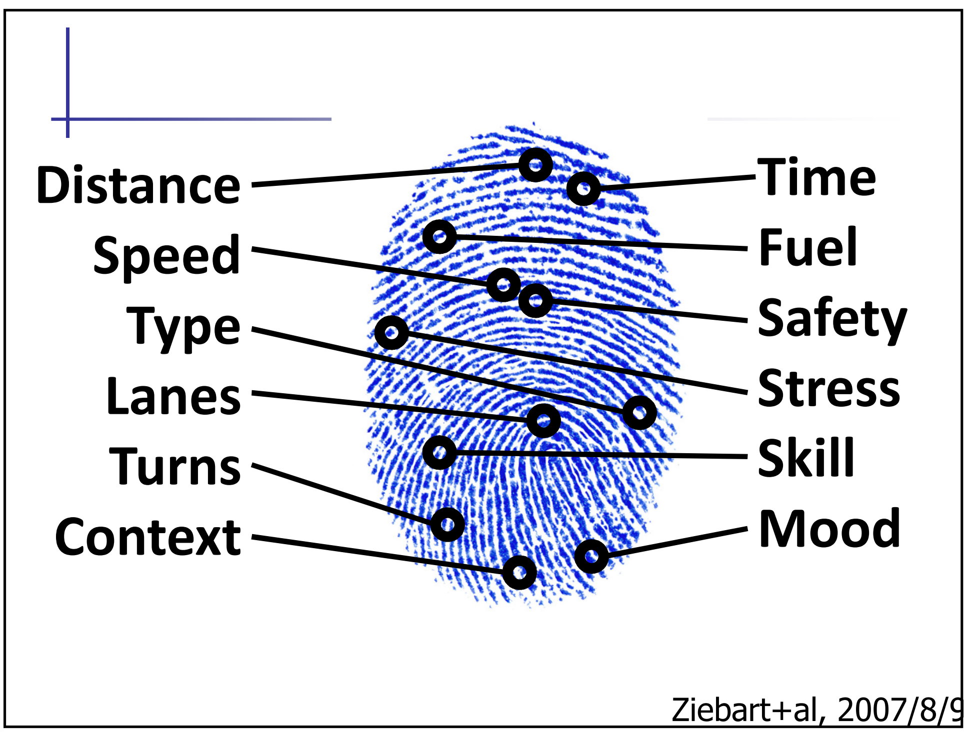
- Run our apprenticeship learning algorithm to find the reward function.
- Receive “test parking lot” map + starting point and destination.
- Find the trajectory that maximizes the *learned reward function* for navigating the test parking lot.

Nice driving style

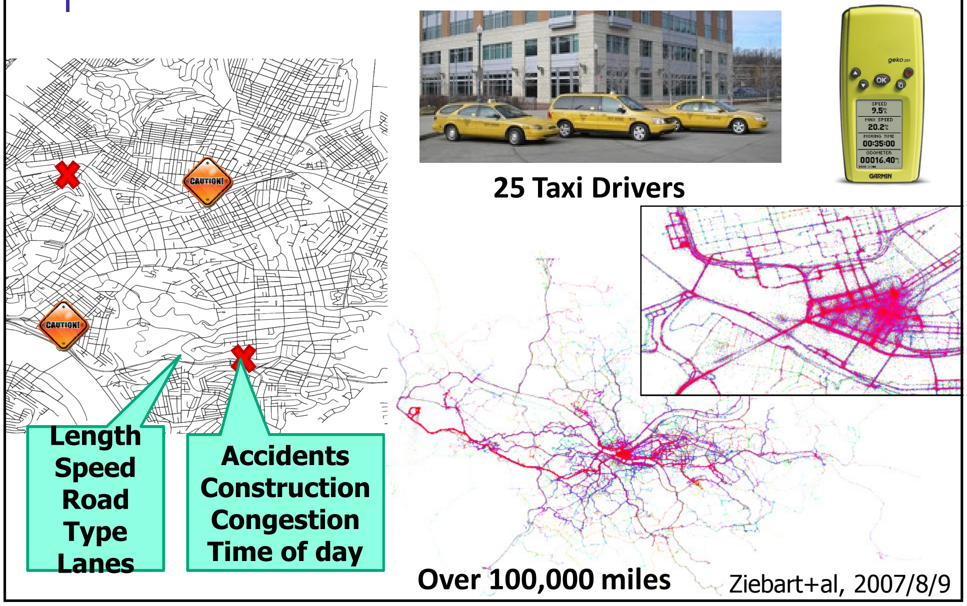
Sloppy driving-style

Only 35% of routes are "fastest" (Letchner, Krumm, & Horvitz 2006)

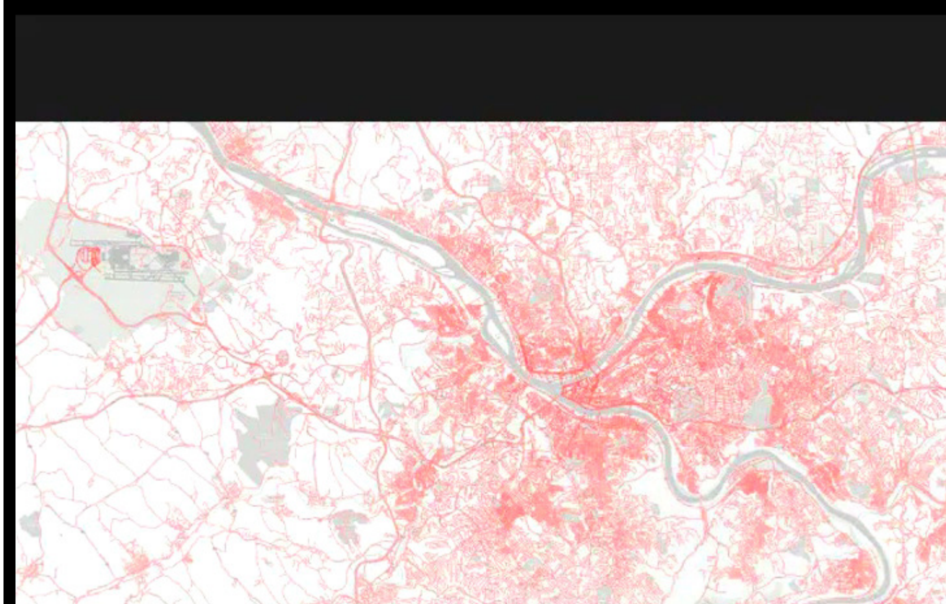

Time
Money
Stress
Skill
Ziebart+al, 2007/8/9


Distance
Speed
Type
Lanes
Turns
Context
Time
Fuel
Safety
Stress
Skill
Mood
Ziebart+al, 2007/8/9

Data Collection



Destination Prediction



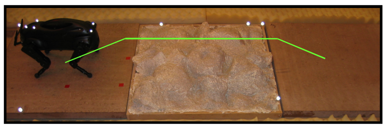
Quadruped

- Reward function trades off 25 features.

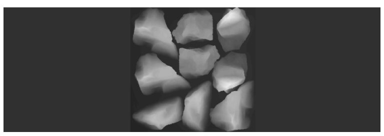
Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

Experimental setup

- Demonstrate path across the "training terrain"



- Run our apprenticeship learning algorithm to find the reward function
- Receive "testing terrain"---height map.



- Find the optimal policy with respect to the *learned reward function* for crossing the testing terrain.

Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

Without learning

With learned reward function

Inverse RL history

- 1964, Kalman posed the inverse optimal control problem and solved it in the 1D input case
- 1994, Boyd+al.: a linear matrix inequality (LMI) characterization for the general linear quadratic setting
- 2000, Ng and Russell: first MDP formulation, reward function ambiguity pointed out and a few solutions suggested
- 2004, Abbeel and Ng: inverse RL for apprenticeship learning---reward feature matching
- 2006, Ratliff+al: max margin formulation

Inverse RL history

- 2007, Ratliff+al: max margin with boosting---enables large vocabulary of reward features
- 2007, Ramachandran and Amir, and Neu and Szepesvari: reward function as characterization of policy class
- 2008, Kolter, Abbeel and Ng: hierarchical max-margin
- 2008, Syed and Schapire: feature matching + game theoretic formulation
- 2008, Ziebart+al: feature matching + max entropy
- 2008, Abbeel+al: feature matching -- application to learning parking lot navigation style
- Active inverse RL? Inverse RL w.r.t. minmax control, partial observability, learning stage (rather than observing optimal policy), … ?

Consider the following scenario:

There are two envelopes, each of which has an unknown amount of money in it. You get to choose one of the envelopes. Given this is all you get to know, how should you choose?

Consider the changed scenario:

Same as above, but before you get to choose, you can ask me to disclose the amount in one of the envelopes. Without any distributional assumptions on the amounts of money, is there a strategy that could improve your expected pay-off over simply picking an envelope at random?

CS 287: Advanced Robotics
Fall 2009

Lecture 17: Policy search

Pieter Abbeel
UC Berkeley EECS

Problem setup

- Input:
  - State space, action space
  - Transition model $P_{sa}(s_{t+1} \mid s_t, a_t)$
  - *No* reward function
  - Teacher's demonstration: $s_0, a_0, s_1, a_1, s_2, a_2, \ldots$
    (= trace of the teacher's policy π^*)
- Inverse RL:
  - Can we recover R ?
- Apprenticeship learning via inverse RL
  - Can we then use this R to find a good policy ?
- Vs. Behavioral cloning (which directly learns the teacher's policy using supervised learning)
  - Inverse RL: leverages compactness of the reward function
  - Behavioral cloning: leverages compactness of the policy class considered, does not require a dynamics model

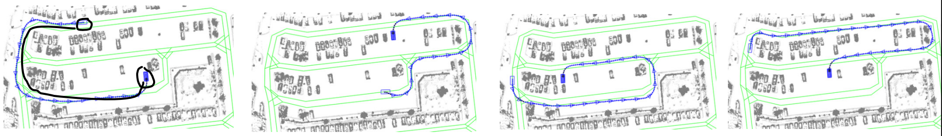
Parking lot navigation

- Reward function trades off:
  - Staying "on-road,"
  - Forward vs. reverse driving,
  - Amount of switching between forward and reverse,
  - Lane keeping,
  - On-road vs. off-road,
  - Curvature of paths.

[Abbeel et al., IROS 08]

Experimental setup

- Demonstrate parking lot navigation on "train parking lots."



- Run our apprenticeship learning algorithm to find the reward function.
- Receive "test parking lot" map + starting point and destination.
- Find the trajectory that maximizes the *learned reward function* for navigating the test parking lot.

Nice driving style

Sloppy driving-style

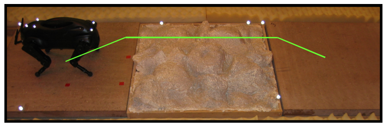
Quadruped

- Reward function trades off 25 features.

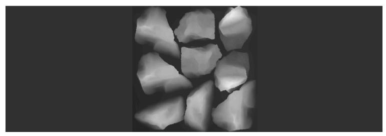
Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

Experimental setup

- Demonstrate path across the “training terrain”



- Run our apprenticeship learning algorithm to find the reward function
- Receive “testing terrain”---height map.



- Find the optimal policy with respect to the *learned reward function* for crossing the testing terrain.

Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

Without learning

With learned reward function

Announcements

- Assignment 1 was due yesterday at 23:59pm
  - For late day policy details, see class webpage.

- Reminder: Project milestone (1 page progress report) due Friday November 6.

- Grading:
  - 3 assignments: 25%, 25%, 5%
  - Final project: 45%

- Final project presentations:
  - Dec 1 & Dec 3
- November 24: Zico Kolter guest lecture

As part of the course requirements, I expect you to attend these 3 lectures

Lecture outline

- Inverse RL case studies wrap-up
- *Policy search*
  - *Assume some policy class parameterized by a vector θ, search for the optimal setting of θ*
  - *Perhaps the largest number of success stories of RL*

Policy class for helicopter hover

x, y, z: x points forward along the helicopter, y sideways to the right, z downward.

n_x, n_y, n_z: rotation vector that brings helicopter back to "level" position (expressed in the helicopter frame).

$$u_{collective} = \theta_1 \cdot f_1(z^* - z) + \theta_2 \cdot \dot{z}$$

$$u_{elevator} = \theta_3 \cdot f_2(x^* - x) + \theta_4 f_4(\dot{x}) + \theta_5 \cdot q + \theta_6 \cdot n_y$$

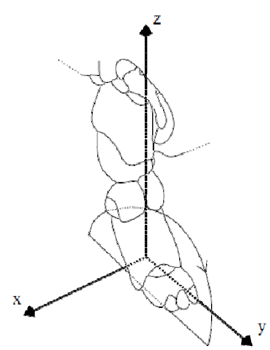
$$u_{aileron} = \theta_7 \cdot f_3(y^* - y) + \theta_8 f_5(\dot{y}) + \theta_9 \cdot p + \theta_{10} \cdot n_x$$

$$u_{rudder} = \theta_{11} \cdot r + \theta_{12} \cdot n_z$$

[Policy search was done in simulation] Ng + al, ISER 2004

Policy class for quadruped locomotion on flat terrain

- 12 parameters define the Aibo's gait:
  - 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



Kohl and Stone, ICRA2004

Kohl and Stone, ICRA 2004

Before learning (hand-tuned)

After learning

[Policy search was done through trials on the actual robot.]

Policy class for tetris

Let $S = s_1, s_2, \ldots, s_n$ be the set of board situations available depending on the choice of placement of the current block. Then the probability of choosing the action that leads to board situation s_i is taken is given by:

$$\frac{\exp\left(\theta^\top \phi(s_i)\right)}{\sum_{j=1}^{n} \exp\left(\theta^\top \phi(s_j)\right)}$$

Deterministic, known dynamics

$$\max_\theta U(\theta) = \max_\theta \mathrm{E}\left[\sum_{t=0}^{H} R(s_t, a_t, s_{t+1}) | \pi_\theta\right]$$

Numerical optimization: find the gradient w.r.t. θ and take a step in the gradient direction.

$$\frac{\partial U}{\partial \theta_i} = \sum_{t=0}^{H} \frac{\partial R}{\partial s}(s_t, u_t, s_{t+1}) \frac{\partial s_t}{\partial \theta_i} + \frac{\partial R}{\partial u}(s_t, u_t, s_{t+1}) \frac{\partial u_t}{\partial \theta_i} + \frac{\partial R}{\partial s'}(s_t, u_t, s_{t+1}) \frac{\partial s_{t+1}}{\partial \theta_i}$$

$$\frac{\partial s_t}{\partial \theta_i} = \frac{\partial f}{\partial s}(s_{t-1}, u_{t-1}) \frac{\partial s_{t-1}}{\partial \theta_i} + \frac{\partial f}{\partial s}(s_{t-1}, u_{t-1}) \frac{\partial u_{t-1}}{\partial \theta_i}$$

$$\frac{\partial u_t}{\partial \theta_i} = \frac{\partial \pi_\theta}{\partial \theta_i}(s_t, \theta) + \frac{\partial \pi_\theta}{\partial s}(s_t, \theta) \frac{\partial s_t}{\partial \theta_i}$$

Computing these recursively, starting at time 0 is a discrete time instantiation of Real Time Recurrent Learning (RTRL)

Stochastic, known dynamics

$$\max_\theta U(\theta) = \max_\theta \mathrm{E}\left[\sum_{t=0}^{H} R(s_t, a_t, s_{t+1}) | \pi_\theta\right] = \max_\theta \sum_{t=0}^{H} \sum_{s,u} P_t(s_t = s, u_t = u; \theta) R(s, u)$$

Numerical optimization: find the gradient w.r.t. θ and take a step in the gradient direction.

$$\frac{\partial U}{\partial \theta_i} = \sum_{t=0}^{H} \sum_{s,u} \frac{\partial P_t}{\partial \theta_i}(s_t = s, u_t = u; \theta) R(s, u)$$

Using the chain rule we obtain the following recursive procedure:

$$P_t(s_t = s, u_t = u; \theta) = \sum_{s_-, u_-} P_{t-1}(s_{t-1} = s_-, u_{t-1} = u_-) T(s_-, u_-, s) \pi(u | s; \theta)$$

$$\frac{\partial P_t}{\partial \theta_i}(s_t = s, u_t = u; \theta) = \sum_{s_-, u_-} \frac{\partial P_{t-1}}{\partial \theta_i}(s_{t-1} = s_-, u_{t-1} = u_-; \theta) T(s_-, u_-, s) \pi(u | s; \theta)$$
$$+ P_{t-1}(s_{t-1} = s_-, u_{t-1} = u_-; \theta) T(s_-, u_-, s) \frac{\partial \pi}{\partial \theta_i}(u | s; \theta)$$

Computationally impractical for most large state/action spaces!

Policy gradient

	Deterministic		Stochastic	
	Known Dynamics	Unknown Dynamics	Known Dynamics	Unknown Dynamics
Analytical	OK Taking derivatives---potentially time consuming and error-prone	N/A	OK Often computationally impractical	N/A

Finite differences

We can compute the gradient g using standard finite difference methods, as follows:

$$\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}$$

Where:

$$e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{array}{l} \\ \\ \\ \\ \leftarrow \; j\text{'th entry} \\ \\ \\ \\ \end{array}$$

Finite differences --- generic points

Locally around our current estimate θ, we can approximate the utility $U(\theta^{(i)})$ at an alternative point $\theta^{(i)}$ with a linear approximation:

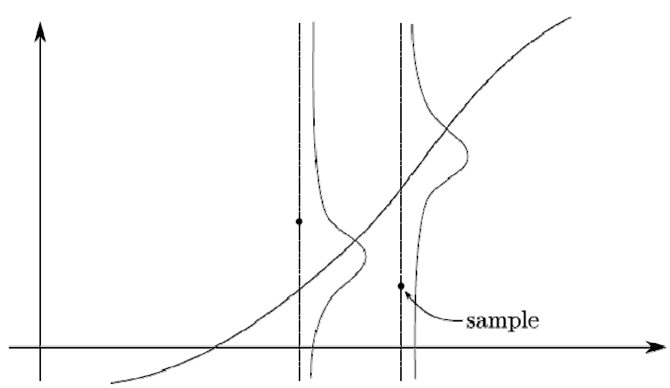
$$U(\theta) \approx U(\theta) + g^T(\theta^{(i)} - \theta)$$

Let's say we have a set of small perturbations $\theta^{(0)}, \theta^{(1)}, \ldots, \theta^{(m)}$ for which we are willing to evaluate the utility $U(\theta^{(i)})$. We can get an estimate of the gradient through solving the following set of equations for the gradient g through least squares:

$$\begin{aligned} U(\theta^{(0)}) &= U(\theta) + (\theta^{(0)} - \theta)^\top g \\ U(\theta^{(1)}) &= U(\theta) + (\theta^{(1)} - \theta)^\top g \\ &\ldots \\ U(\theta^{(m)}) &= U(\theta) + (\theta^{(m)} - \theta)^\top g \end{aligned}$$

Issue in stochastic setting

- Noise could dominate the gradient evaluation



"Fixing" the randomness

- Intuition by example: *wind influence on a helicopter is stochastic, but if we assume the same wind pattern across trials, this will make the different choices of θ more readily comparable.*
- General instantiation:
  - Fix the random seed
  - Result: deterministic system
- **If** the stochasticity can be captured in an appropriate way, this will result in a significantly more efficient gradient computation
  - Details of "appropriate": Ng & Jordan, 2000

Policy gradient

	Deterministic		Stochastic	
	Known Dynamics	Unknown Dynamics	Known Dynamics	Unknown Dynamics
Analytical	OK Taking derivatives---potentially time consuming and error-prone	N/A	OK Often computationally impractical	N/A
Finite differences	OK Sometimes computationally more expensive than analytical	OK	OK N = #roll-outs: Naive: $O(N^{-1/4})$, or $O(N^{-2/5})$ Fix random seed: $O(N^{-1/2})$ [1]	Same as known dynamics, but no fixing of random seed.

[1] P. Glynn, "Likelihood ratio gradient estimation: an overview," in *Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987,* pp. 366–375.

Likelihood ratio method

- Assumption:
  - Stochastic policy $\pi_\theta(u_t \mid s_t)$
  - Stochasticity:
    - Required for the methodology
    - + Helpful to ensure exploration
    - - Optimal policy is often not stochastic (though it can be!!)

Likelihood ratio method

Likelihood ratio method

Likelihood ratio method

Likelihood ratio method

Likelihood ratio method

We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ:

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Likelihood ratio method

$$U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Likelihood ratio method

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)};\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}$$

Likelihood ratio method

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Here:

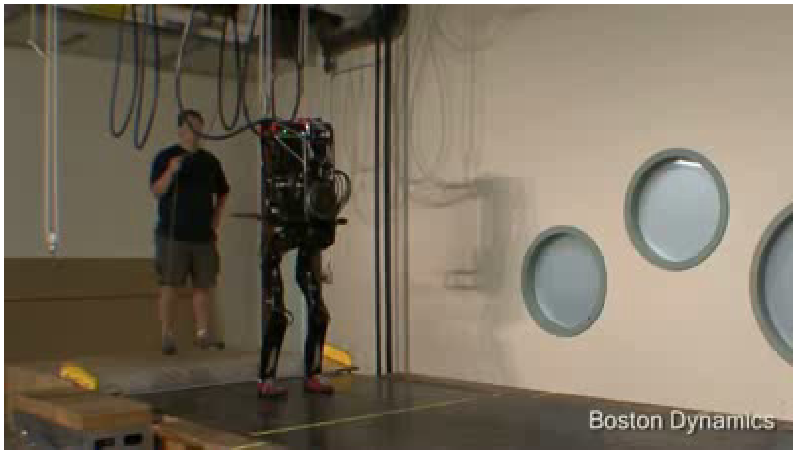
$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$$

We can obtain a gradient estimate from a single trial run! While the math we did is sound (and constitutes the commonly used derivation), this could seem a bit surprising at first. Let's perform another derivation which might give us some more insight.

Boston Dynamics PetMan

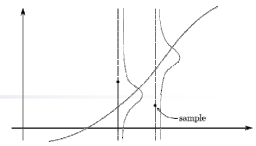


CS 287: Advanced Robotics
Fall 2009

Lecture 18: Policy search

Pieter Abbeel
UC Berkeley EECS

Policy gradient



	Deterministic		Stochastic	
	Known Dynamics	Unknown Dynamics	Known Dynamics	Unknown Dynamics
Analytical	OK Taking derivatives---potentially time consuming and error-prone	N/A	OK Often computationally impractical	N/A
Finite differences	OK Sometimes computationally more expensive than analytical	OK	OK N = #roll-outs: Naive: $O(N^{-1/4})$, or $O(N^{-2/5})$ Fix random seed: $O(N^{-1/2})$ [1]	Same as known dynamics, but no fixing of random seed.
Likelihood ratio method	OK	OK	OK $O(N^{-1/2})$ [1]	OK $O(N^{-1/2})$ [1]

[1] P. Glynn, "Likelihood ratio gradient estimation: an overview," in *Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987,* pp. 366–375.

Likelihood ratio method

- Assumption:
  - Stochastic policy $\pi_\theta(u_t \mid s_t)$
  - Stochasticity:
    - Required for the methodology
    - \+ Helpful to ensure exploration
    - \- Optimal policy within the policy class is not always stochastic (though it can be!!)

Likelihood ratio method

We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

Our goal is to find θ:

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau; \theta) R(\tau)$$

Likelihood ratio method derivation

$$U(\theta) = \sum_\tau P(\tau; \theta) R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_\tau P(\tau; \theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau; \theta) R(\tau) \\
&= \sum_\tau \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P(\tau; \theta) R(\tau) \\
&= \sum_\tau P(\tau; \theta) \frac{\nabla_\theta P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\
&= \sum_\tau P(\tau; \theta) \nabla_\theta \log P(\tau; \theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}$$

Likelihood ratio method: result recap

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Here:

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$$

Likelihood ratio method in practice

- As formulated thus far: yes, unbiased estimator, but very noisy, hence would take very long
- Set of critical fixes that have led to real-world practicality:
  - Add a free parameter to the estimator called “baseline” and set it such that the variance of the estimator is minimized
  - Exploit temporal structure + incorporate value function estimates (= actor-critic learning)
  - Don’t step in the direction of the gradient, follow the “natural” gradient direction instead

Consider the following scenario:

There are two envelopes, each of which has an unknown amount of money in it. You get to choose one of the envelopes. Given this is all you get to know, how should you choose?

Consider the changed scenario:

Same as above, but before you get to choose, you can ask me to disclose the amount in one of the envelopes. Without any distributional assumptions on the amounts of money, is there a strategy that could improve your expected pay-off over simply picking an envelope at random?

Envelopes riddle

Envelopes riddle

Envelopes riddle

- MDP:
  - horizon of 1, always start in state 0
  - Transition to state 1 or 2 according to choice made
  - Observe the reward in the visited state
- Policy

$$\pi_\theta(1|0) = \frac{\exp(\theta)}{1+\exp(\theta)}$$
$$\pi_\theta(2|0) = \frac{1}{1+\exp(\theta)}$$

- Choose to see an envelope's contents according to π_θ
- Perform a gradient update:

$$\nabla_\theta \log P(\tau = 1; \theta) R(\tau = 1) = \frac{1}{1+\exp(\theta)} R(1)$$

$$\nabla_\theta \log P(\tau = 2; \theta) R(\tau = 2) = -\frac{\exp(\theta)}{1+\exp(\theta)} R(2)$$

Envelopes riddle

$$\pi_\theta(1|0) = \frac{\exp(\theta)}{1+\exp(\theta)}$$
$$\pi_\theta(2|0) = \frac{1}{1+\exp(\theta)}$$

- Perform a gradient update:

$$\nabla_\theta \log P(\tau = 1; \theta) R(\tau = 1) = \frac{1}{1+\exp(\theta)} R(1)$$

$$\nabla_\theta \log P(\tau = 2; \theta) R(\tau = 2) = -\frac{\exp(\theta)}{1+\exp(\theta)} R(2)$$

- This gradient update is simply making the recently observed path more likely; and how much more likely depends on the observed R for the observed path
- → rather than let it depend simply on R, if we had a "baseline" b which is an estimate of the expected reward under the current policy, then could update scaled by (R-b) instead,

  i.e. the baseline enables updating such that better than average paths become more likely, less than average paths become less likely

Likelihood ratio method with baseline

- Gradient estimate with baseline:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta)(R(\tau^{(i)}) - b)$$

- This will (crudely speaking) increase the log-likelihood of paths with higher than baseline reward, and decrease the log-likelihood of observed paths with lower than baseline reward.
- Is this still an unbiased gradient estimate?

  Unbiased means: $\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$

Even with baseline, we obtain an unbiased estimate of the gradient

$$\sum_{\tau} P(\tau;\theta) = 1$$
$$\Rightarrow \frac{\partial}{\partial\theta_i}\sum_{\tau} P(\tau;\theta) = 0$$
$$\Leftrightarrow \sum_{\tau} \frac{\partial}{\partial\theta_j} P(\tau;\theta) = 0$$
$$\Leftrightarrow \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \frac{\partial}{\partial\theta_j} P(\tau;\theta) = 0$$
$$\Rightarrow \sum_{\tau} P(\tau;\theta) \frac{\partial}{\partial\theta_j} \log P(\tau;\theta) = 0$$
$$\Rightarrow \mathrm{E}_{\tau}\left[\frac{\partial}{\partial\theta_j} \log P(\tau;\theta)\right] = 0$$
$$\Rightarrow \mathrm{E}_{\tau}\left[\frac{\partial}{\partial\theta_j} \log P(\tau;\theta) b_j\right] = 0$$

$$\begin{aligned}
\frac{\partial}{\partial\theta_j} U(\theta) &= \mathrm{E}_{\tau}\left[\frac{\partial}{\partial\theta_j} \log P(\tau;\theta) R(\tau)\right] \\
&= \mathrm{E}_{\tau}\left[\frac{\partial}{\partial\theta_j} \log P(\tau;\theta)(R(\tau) - b_j)\right] \\
&\approx \frac{1}{m}\sum_{i=1}^{m}\left[\frac{\partial}{\partial\theta_j} \log P(\tau^{(i)};\theta)(R(\tau) - b_j^{(i)})\right]
\end{aligned}$$

Natural choices for b:

- Estimate of utility $U(\theta)$
- Choose b_j to minimize the variance of the gradient estimates.

Our gradient estimate:

$$\hat{g}_j = \frac{\partial}{\partial\theta_j} \log P(\tau^{(i)};\theta) \cdot (R(\tau) - b_j),$$

It is unbiased, i.e.:

$$\mathrm{E}\hat{g}_j = \frac{\partial U(\theta)}{\partial\theta_j}$$

Its variance is given by:

$$\mathrm{E}\left[(\hat{g}_j - \mathrm{E}[\hat{g}_j])^2\right]$$

which we would like to minimize over b_j:

$$\begin{aligned}
\min_{b_j} \mathrm{E}\left[(\hat{g}_j - \mathrm{E}[\hat{g}_j])^2\right] &= \mathrm{E}\hat{g}_j^2 + \mathrm{E}\left[(\mathrm{E}\hat{g}_j)^2\right] - 2\mathrm{E}[\hat{g}_j - \mathrm{E}[\hat{g}_j]] \\
&= \mathrm{E}\hat{g}_j^2 + (\mathrm{E}\hat{g}_j)^2 - 2\mathrm{E}[\hat{g}_j]\,\mathrm{E}[\hat{g}_j] \\
&= \mathrm{E}\hat{g}_j^2 - \underbrace{(\mathrm{E}\hat{g}_j)^2}_{=\frac{\partial U(\theta)}{\partial\theta_j} - \text{independent of } b_j}
\end{aligned}$$

$$
\begin{aligned}
\min_{b_j} \mathrm{E}\hat{g}_j^2 &= \min_{b_j} \mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \cdot (R(\tau) - b_j) \right)^2 \right] \\
&= \min_{b_j} \mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} log(\tau;\theta) \right)^2 \cdot \left(R(\tau)^2 + b_j^2 - 2 b_j R(\tau) \right) \right] \\
&= \underbrace{\min_{b_j} \mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \right)^2 \cdot R(\tau)^2 \right]}_{\text{independent of } b_j} + \mathrm{E} \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \right)^2 \cdot b_j^2 \right] \\
&- 2\mathrm{E} \left[\left(\frac{\partial}{\partial \theta_i} \log P(\tau;\theta)^2 \right) \cdot b_j R(\tau) \right] \\
&= \min_{b_j} b_j^2 \cdot \mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \right)^2 \right] - 2 b_j \mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta)^2 \right)^2 R(\tau) \right]
\end{aligned}
$$

$$
\frac{\partial}{\partial b_j} = 0 \Rightarrow 2 b_j \mathrm{E}_\tau [...] - 2\mathrm{E}_\tau [...] = 0
$$

$$
\boxed{\Rightarrow b_j = \frac{\mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \right)^2 \cdot R(\tau) \right]}{\mathrm{E}_\tau \left[\left(\frac{\partial}{\partial \theta_j} \log P(\tau;\theta) \right)^2 \right]}}
$$

→ Could estimate optimal baseline from samples.

Exploiting temporal structure

Our gradient estimate:

$$
\begin{aligned}
\hat{g}_j &= \frac{1}{m} \sum_{i=1}^{m} \left(\frac{\partial}{\partial \theta_j} \log P(\tau^{(i)};\theta) \right) \left(R(\tau^{(i)}) - b_j \right), \\
&= \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \frac{\partial}{\partial \theta_j} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right) \left(\sum_{t=0}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - b_j \right)
\end{aligned}
$$

Future actions do not depend on past rewards (assuming a fixed policy). This can be formalized as

$$
\mathrm{E} \left[\frac{\partial}{\partial \theta_j} \log \pi_\theta(u_t | s_t) R(s_k, u_k) \right] = 0 \quad \forall k < t
$$

Removing these terms with zero expected value from our gradient estimate we obtain:

$$
\hat{g}_j = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H-1} \frac{\partial}{\partial \theta_j} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b_j \right)
$$

Actor-Critic

Our gradient estimate:

$$\hat{g}_j = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\frac{\partial}{\partial\theta_j}\log\pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1}R(s_k^{(i)},u_k^{(i)})-b_j\right)$$

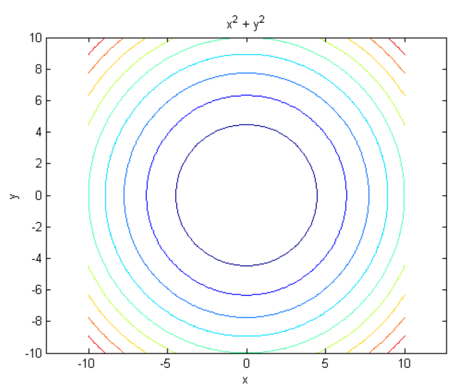
The term $\sum_{k=t}^{H-1}R(s_k^{(i)},u_k^{(i)})$ is a sample based estimate of $Q^{\pi_\theta}(s_t^{(i)},u_k^{(i)})$. If we simultaneously run a temporal difference (TD) learning method to estimate Q^{π_θ}, then we could substitute its estimate for Q instead of the sample based estimate!

Our gradient estimate becomes:

$$\hat{g}_j = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\frac{\partial}{\partial\theta_j}\log\pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\hat{Q}^{\pi_\theta}(s_t^{(i)},u_t^{(i)})-b_j\right)$$

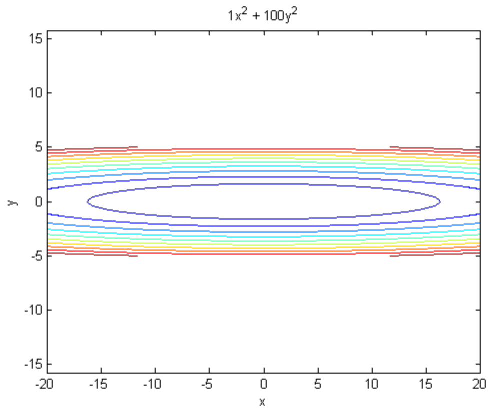
Natural gradient

- Is the gradient the correct direction?



Natural gradient

- Is the gradient the correct direction?



"MODULARITY, POLYRHYTHMS, AND WHAT ROBOTICS AND CONTROL MAY YET LEARN FROM THE BRAIN"

Jean-Jacques Slotine, Nonlinear Systems Laboratory, MIT

Thursday, Nov 5th, 4:00 p.m., 3110 Etcheverry Hall

ABSTRACT

Although neurons as computational elements are 7 orders of magnitude slower than their artificial counterparts, the primate brain grossly outperforms robotic algorithms in all but the most structured tasks. Parallelism alone is a poor explanation, and much recent functional modelling of the central nervous system focuses on its modular, heavily feedback-based computational architecture, the result of accumulation of subsystems throughout evolution. We discuss this architecture from a global functionality point of view, and show why evolution is likely to favor certain types of aggregate stability. We then study synchronization as a model of computations at different scales in the brain, such as pattern matching, restoration, priming, temporal binding of sensory data, and mirror neuron response. We derive a simple condition for a general dynamical system to globally converge to a regime where diverse groups of fully synchronized elements coexist, and show accordingly how patterns can be transiently selected and controlled by a very small number of inputs or connections. We also quantify how synchronization mechanisms can protect general nonlinear systems from noise. Applications to some classical questions in robotics, control, and systems neuroscience are discussed.

The development makes extensive use of nonlinear contraction theory, a comparatively recent analysis tool whose main features will be briefly reviewed.

CS 287: Advanced Robotics
Fall 2009

Lecture 19:
Actor-Critic/Policy gradient for learning to walk in 20 minutes
Natural gradient

Pieter Abbeel
UC Berkeley EECS

Case study: learning bipedal walking

- **Dynamic gait:**
  - A bipedal walking gait is considered dynamic if the ground projection of the center of mass leaves the convex hull of the ground contact points during some portion of the walking cycle.
- **Why hard?**
  - Achieving stable dynamic walking on a bipedal robot is a difficult control problem because bipeds can only control the trajectory of their center of mass through the unilateral, intermittent, uncertain force contacts with the ground.
- ←→ “fully actuated walking”

Passive dynamic walkers

Passive dynamic walkers

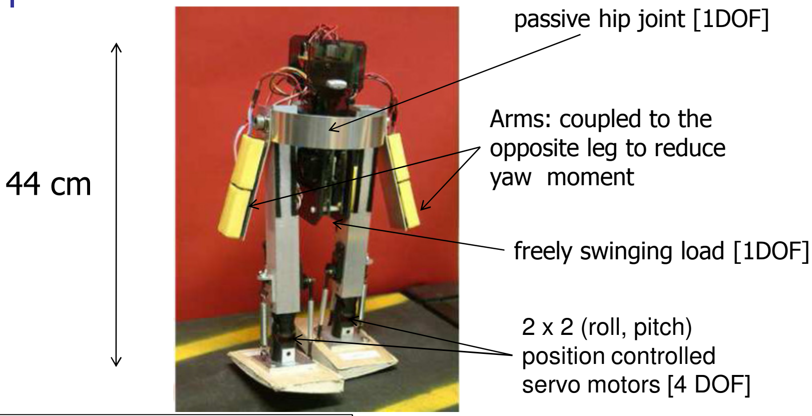
- The energy lost due to friction and collisions when the swing leg returns to the ground are balanced by the gradual conversion of potential energy into kinetic energy as the walker moves down the slope.

- Can we actuate them to have them walk on flat terrains?

- John E. Wilson. Walking toy. Technical report, United States Patent Office, October 15 1936.
- Tad McGeer. Passive dynamic walking. *International Journal of Robotics Research, 9(2):62.82, April 1990.*

Learning to walk in 20 minutes --- Tedrake, Zhang, Seung 2005

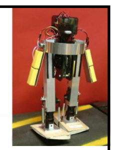
Learning to walk in 20 minutes --- Tedrake, Zhang, Seung 2005



9DOFs:
* 6 internal DOFs
* 3 DOFs for the robot's orientation (always assumed in contact with ground at a single point, absolute (x,y) ignored)

Natural gait down 0.03 radians ramp: 0.8Hz, 6.5cm steps

Dynamics



$$\ddot{q} = f(q, \dot{q}, u, d(t))$$

- q: vector of joint angles
- u: control vector (4D)
- d(t): time-varying vector of random disturbances

- Discrete footstep-to-footstep dynamics: consider state at touchdown of robot's left leg

$$F_\pi(x', x) = P(\hat{x}_{n+1} = x' | \hat{x}_n = x; \pi)$$

- Stochasticity due to
  - Sensor noise
  - Disturbances d(t)

Reinforcement learning formulation

- **Goal:** stabilize the limit cycle trajectory that the passive robot follows when walking down the ramp, making it invariant to slope.
- **Reward function:**

$$R(x(n)) = -\frac{1}{2}\|x(n) - x^*\|_2^2$$

  - x* is taken from the gait of the walker down a slope of 0.03 radians
- **Action space:**
  - At the beginning of each step cycle (=when a foot touches down) we choose an action in the discrete time RL formulation
  - Our action choice is a feedback control policy to be deployed during the step, in this particular example it is a column vector w
  - Choosing this action means that throughout the following step cycle, the following continous-time feedback controls will be exerted:

$$u(t) = \sum_i w_i \phi_i(\hat{x}(t)) = w^\top \phi(\hat{x}(t))$$

→ Goal: find the (constant) action choice w which maximizes expected sum of rewards

Policy class

- To apply the likelihood gradient ratio method, we need to define a stochastic policy class. A natural choice is to choose our action vector w to be sampled from a Gaussian:

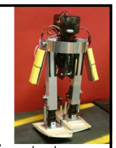
$$w \sim \mathcal{N}(\theta, \sigma^2 I)$$

  Which gives us:

$$\pi_\theta(w|x) = \frac{1}{(2\pi)^d \sigma^d} \exp\left(\frac{-1}{2\sigma^2}(w-\theta)^\top (w-\theta)\right)$$

  [Note: it does not depend on x, this is the case b/c the actions we consider are feedback policies themselves!]
- The policy optimization becomes optimizing the mean of this Gaussian. [In other papers people have also included the optimization of the variance parameter.]

Policy update



Likelihood ratio based gradient estimate from a single trace of H footsteps:

$$\hat{g} = \sum_{n=0}^{H-1} \nabla_\theta \log \pi_\theta(w(n)|\hat{x}(n)) \left(\sum_{k=n}^{H-1} R(\hat{x}(k)) - b \right)$$

We have:

$$\nabla_\theta \log \pi_\theta(w|\hat{x}) = \frac{1}{2\sigma^2}(w - \theta)$$

Rather than waiting till horizon H is reached, we can perform the updates online as follows: (here η_θ is a step-size parameter, $b(n)$ is the amount of baseline we allocate to time n—see next slide)

$$\begin{aligned} e(n) &= e(n-1) + \frac{1}{2\sigma^2}(w(n) - \theta(n)) \\ \theta(n+1) &= \theta(n) + \eta_\theta e(n)(R(\hat{x}(n)) - b(n)) \end{aligned}$$

To reduce variance, can discount the eligibilities:

$$e(n) = \gamma e(n-1) + \frac{1}{2\sigma^2}(w(n) - \theta(n))$$

Choosing the baseline b(n)

A good choice for the baseline is such that it corresponds to an estimate of the expected reward we should have obtained under the current policy.

Assuming we have estimates of the value function $\hat{V}$ under the current policy, we can estimate such a baseline as follows:

$$b(n) = \hat{V}(\hat{x}(n)) - \gamma \hat{V}(\hat{x}(n+1))$$

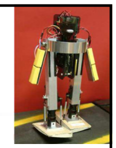
To estimate $\hat{V}$ we can use TD(0) with function approximation. Using linear value function approximation, we have:

$$\hat{V}(\hat{x}) = \sum_i v_i \psi_i(\hat{x}).$$

This gives us the following update equations to learn $\hat{V}$ with TD(0):

$$\begin{aligned} \delta(n) &= R(\hat{x}(n)) + \gamma \hat{V}(\hat{x}(n+1)) - \hat{V}(\hat{x}(n)) \\ v(n+1) &= v(n) + \eta_v \delta(n) \psi(\hat{x}(n)) \end{aligned}$$

The complete actor critic learning algorithm



Before each foot step, sample the feedback control policy parameters $w(n)$ from $\mathcal{N}(\theta(n), \sigma^2 I)$.

During the foot step, execute the following controls in continuous time: $u(t) = w(n)^\top \phi(\hat{x}(t))$.

After the foot step is completed, compute the reward function $R(\hat{x}(n))$ and perform the following updates:

Policy updates:

$$\begin{aligned} e(n) &= \gamma e(n-1) + \frac{1}{2\sigma^2}(w - \theta(n)) \\ \theta(n+1) &= \theta(n) + \eta_\theta e(n)(R(\hat{x}(n)) - b(n)) \\ b(n) &= \hat{V}(\hat{x}(n)) - \gamma \hat{V}(\hat{x}(n+1)) \end{aligned}$$

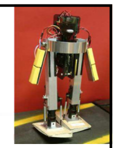
TD(0) updates:

$$\begin{aligned} \delta(n) &= R(\hat{x}(n)) + \gamma \hat{V}(\hat{x}(n+1)) - \hat{V}(\hat{x}(n)) \\ v(n+1) &= v(n) + \eta_v \delta(n) \psi(\hat{x}(n)) \end{aligned}$$

Manual dimensionality reduction

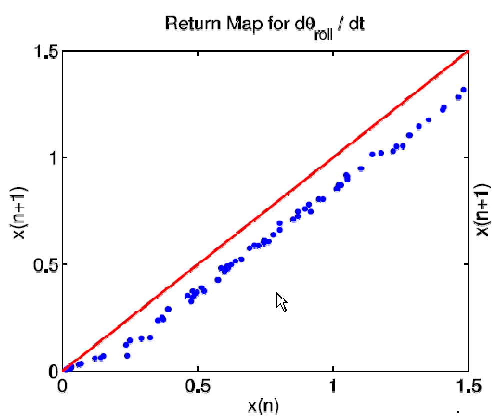
- Decompose the control problem in the frontal and sagittal planes
- Due to simplicity of sagittal plane control---hand set.
- Left with control of the ankle roll actuators to control in the frontal plane
  - Let roll control input only depend on θ_{roll} and $d\theta_{roll}/dt$
  - Basis functions: non-overlapping tile encoding
    - Policy: 35 tiles (5 in θ_{roll} x 7 in $d\theta_{roll}/dt$)
    - Value: 11 tiles (a function in $d\theta_{roll}/dt$ only because the value is evaluated at the discrete time when θ_{roll} hits a particular value)

Experimental setup and results

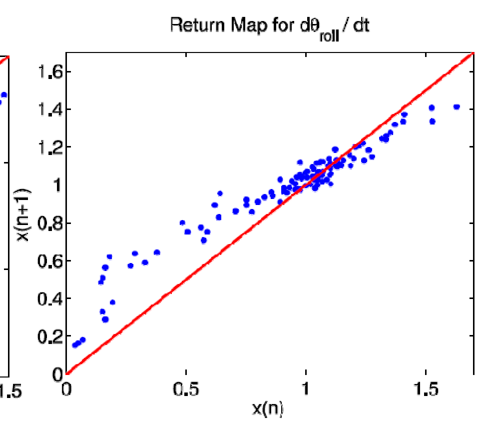


- When the learning begins, the policy parameters, w, are set to 0 and the baseline parameters, v, are initialized so that \hat{V}(x) $\approx$ R(x) / (1-γ)
- Train the robot on flat terrain.
- Reset with simple hand-designed controller that gets it into a random initial state every 10s.
- Results:
  - After 1 minute: foot clearance on every step
  - After 20 minutes: converged to a robust gait (=960 steps at 0.8Hz)

Return maps



before learning



after learning

[Note: this is a projection from 2x9-1 dim to 1dim]

Toddler movie

- On tread-mill: passive walking. On ground: learning.

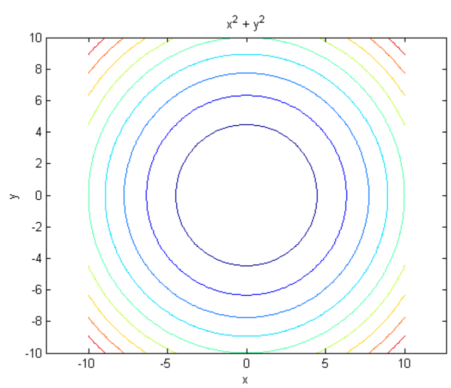
CS 287: Advanced Robotics
Fall 2009

Lecture 20:
Natural gradient
Reward shaping
Approximate LP with function approximation
POMDP
Hierarchical RL

Pieter Abbeel
UC Berkeley EECS

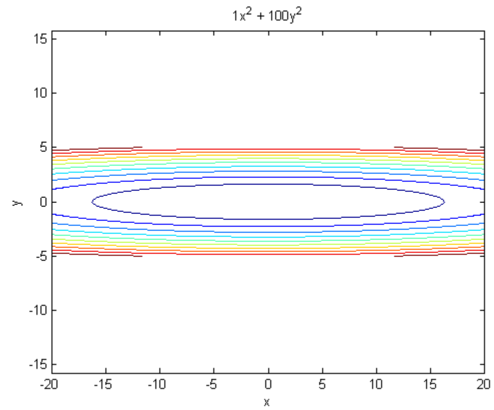
Natural gradient

- Is the gradient the correct direction?



Natural gradient

- Is the gradient the correct direction?



Gradient and linear transformations

Consider the optimization of the function $f(\theta)$ through gradient descent. In iteration k we would perform an update of the following form:

$$\theta^{(k+1)} = \theta^{(k)} + \alpha \nabla_\theta f(\theta^{(k)}). \tag{1}$$

Consider a new coordinate system $x = A^{-1}\theta$. We could work in the new coordinate system instead, and optimize $f(Ax)$ over the variable x. A gradient descent step is given by:

$$x^{(k+1)} = x^{(k)} + \alpha \nabla_x f(Ax^{(k)}) = x^{(k)} + \alpha A^\top \nabla_\theta f(Ax^{(k)}) \tag{2}$$

If $x^{(k)} = A^{-1}\theta^{(k)}$, do we have $x^{(k+1)} = \theta^{(k+1)}$?
No!
The value in θ coordinates that corresponds to x_{k+1} is given by

$$Ax^{(k+1)} = Ax^{(k)} + \alpha AA^\top \nabla_\theta f(Ax^{(k)}) = \theta^{(k)} + \alpha AA^\top \nabla_\theta f(\theta^{(k)}) \neq \theta^{(k+1)}$$

Newton's direction

Newton's method approximates the function $f(\theta)$ by a quadratic function through a Taylor expansion around the current point θ_k:

$$f(\theta) \approx f(\theta_k) + \nabla_\theta f(\theta^{(k)})^\top (\theta - \theta^{(k)}) + \frac{1}{2}(\theta - \theta^{(k)})^\top H(\theta^{(k)})(\theta - \theta^{(k)})$$

Here $H_{ij}(\theta^{(k)}) = \frac{\partial^2 f}{\partial \theta_i \theta_j}(\theta^{(k)})$ is a matrix with the 2nd derivatives of f evaluated at $\theta^{(k)}$.

The local optimum of the 2nd order approximation is found by setting its gradient equal to zero, which gives:

$$\text{Newton step direction} = (\theta - \theta^{(k)}) = -H^{-1}(\theta^{(k)})\nabla_\theta f(\theta^{(k)})$$

The Newton step direction is affine invariant

Newton's step direction for $f(\theta)$ is given by:

$$H^{-1}(\theta^{(k)})\nabla_\theta f(\theta^{(k)}). \tag{1}$$

For $f(Ax)$, with $x = A^{-1}\theta$, we have

$$\begin{aligned} \text{Hessian} &= A^\top H A \\ \text{gradient} &= A^\top \nabla_\theta f \end{aligned}$$

Hence we have for the Newton step direction in the x coordinates:

$$\left(A^\top H A\right)^{-1} A^\top \nabla_\theta f(Ax^{(k)}) = A^{-1}\nabla_\theta f(Ax^{(k)}) \tag{2}$$

Translating this into θ coordinates gives us $AA^{-1}\nabla_\theta f(Ax^{(k)}) = \nabla_\theta f(Ax^{(k)})$, which is identical to the step direction directly computed in θ coordinates.

Natural gradient

- Gradient depends on choice of coordinate system.
- Newton's method is invariant to affine coordinate transformations, but not to general coordinate transformations.
- Can we achieve more invariance than simply affine invariance?

Natural gradient

- Let's first re-interpret the gradient:
  - The gradient is the direction of steepest ascent:

Natural gradient

- Let's first re-interpret the gradient:
  - The gradient is the direction of steepest ascent:

For small ϵ we have:

$$\begin{aligned}
\arg \max_{\delta\theta : \|\delta\theta\|_2 \leq \epsilon} f(\theta + \delta\theta) &\approx \arg \max_{\delta\theta : \|\delta\theta\|_2 \leq \epsilon} f(\theta) + \nabla_\theta f(\theta)^\top \delta\theta \\
&= \arg \max_{\delta\theta : \|\delta\theta\|_2 \leq \epsilon} \nabla_\theta f(\theta)^\top \delta\theta \\
&= \frac{\nabla_\theta f(\theta)}{\|\nabla_\theta f(\theta)\|_2} \epsilon
\end{aligned}$$

  - When expressing our problem in a different coordinate system f remains the same, but the || . || <= 1 constraint means something different in different coordinate systems.
- → Can we find a norm constraint that is independent of the coordinate system????

A distance which is independent of the parameterization of the policy class

- **Kullback-Leibler divergence** between distributions over paths induced by the policies:

$$KL(P(\tau;\theta_1)||P(\tau;\theta_2)) = \sum_\tau P(\tau;\theta_1) \log \frac{P(\tau;\theta_1)}{P(\tau;\theta_2)}$$

- E.g., 2 Bernoulli distributions:
  - Prob(heads)=p and Prob(heads)=q

$$\mathrm{KL}(P(\cdot;p)||P(\cdot;q)) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$$

  - Alternative parameterization: $\text{Prob(heads)} = \frac{\exp(\theta)}{1+\exp(\theta)}$ and $\text{Prob(heads)} = \frac{\exp(\psi)}{1+\exp(\psi)}$

$$\begin{aligned}
\mathrm{KL}(P(\cdot;\theta)\|P(\cdot;\psi)) &= \frac{\exp(\theta)}{1+\exp(\theta)} \log \frac{\frac{\exp(\theta)}{1+\exp(\theta)}}{\frac{\exp(\psi)}{1+\exp(\psi)}} + \frac{1}{1+\exp(\theta)} \log \frac{\frac{1}{1+\exp(\theta)}}{\frac{1}{1+\exp(\psi)}} \\
&= \mathrm{KL}(P(\cdot;p)\|P(\cdot;q)) \quad \text{if } \; p = \frac{\exp(\theta)}{1+\exp(\theta)}, q = \frac{\exp(\psi)}{1+\exp(\psi)}
\end{aligned}$$

A distance which is independent of the parameterization of the policy class

- **Kullback-Leibler divergence** between distributions over paths induced by the policies:

$$KL(P(\tau;\theta_1)||P(\tau;\theta_2)) = \sum_\tau P(\tau;\theta_1)\log\frac{P(\tau;\theta_1)}{P(\tau;\theta_2)}$$

- Second-order Taylor approximation

$$\begin{aligned} KL(P(\tau;\theta)||P(\tau;\theta+\delta\theta)) &\approx \sum_\tau P(\tau;\theta)\delta\theta^\top \nabla_\theta \log P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)^\top \delta\theta \\ &= \delta\theta^\top \left(\sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)^\top\right)\delta\theta \\ &= \delta\theta^\top G(\theta)\delta\theta \end{aligned}$$

- G(θ) = ***Fisher information matrix***, independent of the choice of parameterization of the class of distributions.

2nd order Taylor expansion of KL divergence

$$\begin{aligned}
& \mathrm{KL}(P(X;\theta)\|P(X;\theta+\delta\theta) \\
= & \sum_x P(x;\theta)\log\frac{P(x;\theta)}{P(x;\theta+\delta\theta)} \\
\approx & \sum_x P(x;\theta)\left(\log\frac{P(x;\theta)}{P(x;\theta)} - \frac{d}{d\theta}\log P(x;\theta)^\top\delta\theta - \frac{1}{2}\delta\theta^\top\frac{d^2}{d\theta^2}\log P(x;\theta)\delta\theta\right) \\
= & -\sum_x P(x;\theta)\frac{d}{d\theta}\log P(x;\theta)^\top\delta\theta - \frac{1}{2}\delta\theta^\top\frac{d^2}{d\theta^2}\log P(x;\theta)\delta\theta \\
= & -\sum_x P(x;\theta)\left(\frac{\frac{d}{d\theta}P(x;\theta)}{P(x;\theta)}\right)^\top\delta\theta - \frac{1}{2}\delta\theta\sum_x P(x;\theta)\frac{P(x;\theta)\frac{d^2}{d\theta^2}P(x;\theta) - \left(\frac{dP(x;\theta)}{d\theta}\right)\left(\frac{dP(x;\theta)}{d\theta}\right)^\top}{P(x;\theta)^2}\delta\theta \\
= & -\sum_x \frac{d}{d\theta}P(x;\theta)^\top\delta\theta - \frac{1}{2}\delta\theta\sum_x\frac{d^2}{d\theta^2}P(x;\theta)\delta\theta + \frac{1}{2}\delta\theta\sum_x P(x;\theta)\left(\frac{\frac{dP(x;\theta)}{d\theta}}{P(x;\theta)}\right)\left(\frac{\frac{dP(x;\theta)}{d\theta}}{P(x;\theta)}\right)^\top\delta\theta \\
= & -\left(\frac{d}{d\theta}\sum_x P(x;\theta)\right)^\top\delta\theta - \frac{1}{2}\delta\theta^\top\left(\frac{d^2}{d\theta^2}\sum_x P(x;\theta)\right)\delta\theta \\
& +\frac{1}{2}\delta\theta^\top\left(\sum_x P(x;\theta)\left(\frac{d}{d\theta}\log P(x;\theta)\right)\left(\frac{d}{d\theta}\log P(x;\theta)\right)^\top\right)\delta\theta \\
= & -\left(\frac{d}{d\theta}1\right)^\top\delta\theta - \frac{1}{2}\delta\theta^\top\left(\frac{d^2}{d\theta^2}1\right)\delta\theta \\
& +\frac{1}{2}\delta\theta^\top\left(\sum_x P(x;\theta)\left(\frac{d}{d\theta}\log P(x;\theta)\right)\left(\frac{d}{d\theta}\log P(x;\theta)\right)^\top\right)\delta\theta \\
= & -0-0+\frac{1}{2}\delta\theta^\top\left(\sum_x P(x;\theta)\left(\frac{d}{d\theta}\log P(x;\theta)\right)\left(\frac{d}{d\theta}\log P(x;\theta)\right)^\top\right)\delta\theta \\
= & \frac{1}{2}\delta\theta^\top G(\theta)\delta\theta
\end{aligned}$$

Natural gradient g_N

Natural gradient: general setting

Natural gradient in policy search

Natural gradient g_N

- = the direction with highest increase in the objective per change in KL divergence

$$\begin{aligned} g_N &= \arg \max_{\delta\theta : KL(P(\tau;\theta)||P(\tau;\theta+\delta\theta) \leq \epsilon} f(\theta + \delta\theta) \\ &\approx \arg \max_{\delta\theta : \frac{1}{2}\delta\theta^\top G(\theta)\delta\theta \leq \epsilon} f(\theta) + \nabla_\theta f(\theta)^\top \delta\theta \\ &= \arg \max_{\delta\theta : \frac{1}{2}\delta\theta^\top G(\theta)\delta\theta \leq \epsilon} \nabla_\theta f(\theta)^\top \delta\theta \\ &= G(\theta)^{-1} \nabla_\theta f(\theta) \end{aligned}$$

Natural gradient: general setting

Problem setting: optimize an objective which depends on a probability distribution P_θ

$$\max_\theta f(P_\theta)$$

Rather than following the gradient, which depends on the choice of parameterization for the set of probability distributions that we are searching over, follow the natural gradient g_N:

$$g_N = G(\theta)^{-1} \nabla_\theta f(P_\theta)$$

Here $G(\theta)$ is the Fisher information matrix, and can be computed as follows:

$$G(\theta) = \sum_{x \in X} P_\theta(x) \nabla_\theta \log P_\theta(x) \nabla_\theta \log P_\theta(x)^\top$$

Natural gradient in policy search

Objective:

$$U(\theta) = \sum_\tau P(\tau; \theta) R(\tau)$$

Natural gradient g_N:

$$g_N = G(\theta)^{-1} \nabla_\theta U(\theta)$$

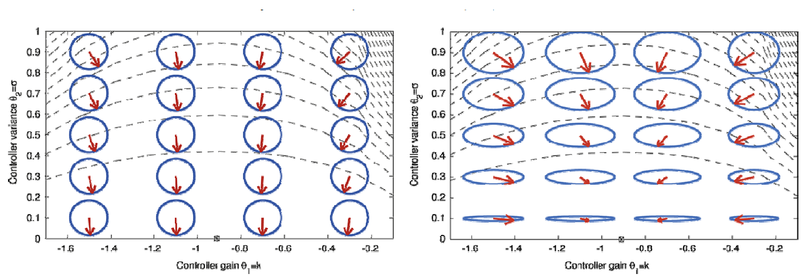
Both the Fisher information matrix G and the gradient need to be estimate from samples. We have seen many ways to estimate the gradient from samples. Remains to show how to estimate G.

$$\begin{aligned} G(\theta) &= \sum_\tau P(\tau) \nabla_\theta \log P(\tau; \theta) \nabla_\theta \log P(\tau; \theta)^\top \\ &\approx \frac{1}{m} \sum_{i=1}^m \nabla_\theta \log P(\tau^{(i)}; \theta) \nabla_\theta \log P(\tau^{(i)}; \theta)^\top \end{aligned}$$

As we have seen earlier, we can compute the expression $\nabla_\theta \log P(\tau^{(i)})$ even without access to the dynamics model:

$$\begin{aligned} \nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \sum_{t=0}^{H-1} \log T(s_t, u_t, s_{t+1}) + \log \pi_\theta(u_t | s_t) \\ &= \nabla_\theta \sum_{t=0}^{H-1} \log \pi_\theta(u_t | s_t) \end{aligned}$$

Example



(a) Vanilla policy gradient. (b) Natural policy gradient.

Kober and Peters, NIPS 2009

Announcements

- Project milestone due tomorrow 23:59pm
  - = 1 page progress update.
  - Format: pdf
- Assignment #2: out tomorrow night, due in 2 weeks
  - Topic: RL
  - Start early!
- Late day policy: 7 days total; -20pts (out of 100pts of the thing you are submitting late) per day beyond that
- Assignment #3 will be released in 2 weeks and will be very small compared to #1 and #2.

Thomas Daniel, University of Washington

A Tale of Two (or Three?) Gyroscopes: Inertial measurement units (IMUs) in flying insects

Date: Thursday, November 5, 2009
Time: 4:00 PM
Place: 2040 Valley Life Sciences Building

Animals use a combination of sensory modalities to control their movement including visual, mechanosensory and chemosensory information. Mechanosensory systems that can detect inertial forces are capable of responding much more rapidly than visual systems and, as such, are thought to play a critical role in rapid course correction during flight. This talk focusses on two gryoscopic organs: halteres of flies and antennae of moths. Both have mechanical and neural components play critical roles in encoding relatively tiny Coriolis forces associated with body rotations, both of which will be reviewed along with new data that suggests each have complex circuits that connect visual systems to mechanosensory systems. But, insects are bristling with mechanosensory structures, including the wings themselves. It is not clear whether these too could serve an IMU function in addition to their obvious aerodynamic roles.

“MODULARITY, POLYRHYTHMS, AND WHAT ROBOTICS AND CONTROL MAY YET LEARN FROM THE BRAIN”

Jean-Jacques Slotine, Nonlinear Systems Laboratory, MIT

Thursday, Nov 5th, 4:00 p.m., 3110 Etcheverry Hall

ABSTRACT

Although neurons as computational elements are 7 orders of magnitude slower than their artificial counterparts, the primate brain grossly outperforms robotic algorithms in all but the most structured tasks. Parallelism alone is a poor explanation, and much recent functional modelling of the central nervous system focuses on its modular, heavily feedback-based computational architecture, the result of accumulation of subsystems throughout evolution. We discuss this architecture from a global functionality point of view, and show why evolution is likely to favor certain types of aggregate stability. We then study synchronization as a model of computations at different scales in the brain, such as pattern matching, restoration, priming, temporal binding of sensory data, and mirror neuron response. We derive a simple condition for a general dynamical system to globally converge to a regime where diverse groups of fully synchronized elements coexist, and show accordingly how patterns can be transiently selected and controlled by a very small number of inputs or connections. We also quantify how synchronization mechanisms can protect general nonlinear systems from noise. Applications to some classical questions in robotics, control, and systems neuroscience are discussed.

The development makes extensive use of nonlinear contraction theory, a comparatively recent analysis tool whose main features will be briefly reviewed.

Reinforcement learning: remaining topics

- approximate LP, pomdp's, reward shaping, exploration vs. exploitation, hierarchical methods

Approximate LP

- Exact Bellman LP

$$\begin{aligned} \min_V \quad & \sum_s c(s)V(s) \\ \text{s.t.} \quad & V(s) \geq \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V(s')\right) \quad \forall s, \forall a \end{aligned}$$

- Approximate LP

$$\begin{aligned} \min_\theta \quad & \sum_{s \in S'} c(s)\theta^\top \phi(s) \\ \text{s.t.} \quad & \theta^\top \phi(s) \geq \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma \theta^\top \phi(s')\right) \quad \forall s \in S', \forall a \end{aligned}$$

Approximate LP guarantees

- When retaining all constraints, yet introducing function approximation.

$$\begin{aligned} \min_\theta \quad & \sum_{s \in S} c(s)\theta^\top \phi(s) \\ \text{s.t.} \quad & \theta^\top \phi(s) \geq \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma \theta^\top \phi(s')\right) \quad \forall s \in S, \forall a \end{aligned}$$

Theorem. [**de Farias and Van Roy**] If one of the basis function satisfies $\phi_i(s) = 1$ for all $s \in S$, then the LP has a feasible solution and the optimal solution $\hat{\theta}$ satisfies:

$$\|V^* - \Phi\theta\|_{1,c} \leq \frac{2}{1-\alpha} \min_\theta \|V^* - \Phi\theta\|_\infty$$

Constraint sampling

Consider a convex optimization problem with a very large number of constraints:

$$\begin{array}{ll} \min & c^\top x \\ s.t. & g_i(x) \leq b \;\; i = 1, 2, \ldots, m \end{array}$$

where $x \in \Re^n$, g_i convex, and $m >> n$.

We obtain the sampled approximation by sampling the sequence $\{i_1, i_2, \ldots, i_N\}$ IID according to some measure over the constraints μ. This gives us:

$$\begin{array}{ll} \min & c^\top x \\ s.t. & g_j(x) \leq b \;\; j = i_1, i_2, \ldots, i_N \end{array}$$

Let $\hat{x}_N$ be the optimal solution to the sampled convex problem.

Theorem. [**Calafiore and Campi, 2005**] For arbitrary $\epsilon > 0, \delta > 0$, if $N \geq \frac{n}{\epsilon\delta} - 1$, then

$$\text{Prob}\left(\mu\left(\{i | g_i(\hat{x}_N) > b_i\}\right) \leq \epsilon\right) \geq 1 - \delta$$

where the probability is taken over the random sampling of constraints.

This result can be leveraged to show that the solution to the sampled approximate LP is close to V^* with high probablity. (de Farias and Van Roy, 2001)

Reward shaping

- What freedom do we have in specifying the reward function? Can we choose it such that learning is faster?
- Examples:
  - \+ Tetris: set the reward equal to the distance between highest filled square and the top of the board vs. a reward of 1 for placing a block
  - \- Bicycle control task: provide a positive reward for motion towards the goal
  - \- Soccer task: provide a positive reward for touching the ball

Potential based shaping

- Let $F(s,a,s') = \gamma\,\phi(s') - \phi(s)$
- Shaped reward = R + F

- **Theorem** [Ng, Harada & Russell, 1999]

  Potential based reward shaping is a necessary and sufficient condition to guarantee that the optimal policy in the shaped MDP M' = (S, A, T, γ, R+F) is also an optimal policy in the original MDP M = (S, A, T, γ, R)

  [In fact even stronger: all policies retain their relative value.]

Intuition of proof

- In the new MDP, for a trace $s_0, a_0, s_1, a_1, \ldots$ we obtain:

$$R(s_0, a_0, s_1) + \gamma\,\phi(s_1) - \phi(s_0)$$
$$+ \gamma\,(R(s_1, a_1, s_2) + \gamma\,\phi(s_2) - \phi(s_1))$$
$$+ \gamma^2 (R(s_2, a_2, s_3) + \gamma\,\phi(s_3) - \phi(s_2))$$
$$+ \ldots$$

$$= \quad -\phi(s_0) + R(s_0, a_0, s_1) + \gamma\,R(s_1, a_1, s_2) + \gamma^2\,R(s_2, a_2, s_3) + \ldots$$

→ For any policy π we have: (M: original MDP, M': MDP w/shaped reward)

$$V^{\pi}_{M'}(s_0) = V^{\pi}_{M}(s_0) - \phi(s_0)$$

A good potential?

- Let ϕ = V*
- Then in one update we have:

$$
\begin{aligned}
V(s) &\leftarrow \max_a \sum_{s'} T(s,a,s')\left(R(s,a,s') + F(s,a,s') + \gamma V(s')\right) \\
&= \max_a \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V^*(s') - V(s) + \gamma V(s')\right) \\
&= -V^*(s) + \max_a \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V^*(s') + \gamma V(s')\right)
\end{aligned}
$$

- If we initialize V = 0, we obtain:

$$
\begin{aligned}
V(s) &\leftarrow -V^*(s) + \max_a \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V^*(s')\right) \\
&= -V^*(s) + V^*(s) \\
&= 0
\end{aligned}
$$

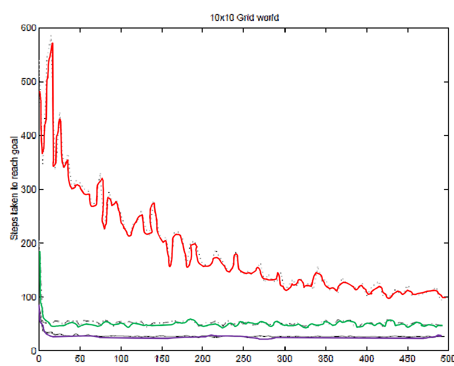
→ V=0 satisfies the Bellman equation; → this particular choice of potential function / reward shaping, we can find the solution to the shaped MDP very quickly

Example

10x10 grid world, 1 goal state = absorbing, other states R=-1;
Prob(action successful) = 80%

→ Shaping function: ϕ_0(s) = - (manhattan distance to goal) / 0.8

Plot shows performance of Sarsa(0) with epsilon=0.1 greedy, learning rate = .2
(a) no shaping vs. (b) $\phi = 0.5 * \phi_0$ vs. (c) $\phi = \phi_0$



[From Ng, Harada & Russell 1999]

Relationship to value function intialization

- Potential based shaping can be shown to be equivalent to initializing the value function to the shaping potential.

[see course website for the technical note describing this]

→ If restricting ourselves to potential based shaping, can implement it in 2 ways.

Partial observability

- = no direct observation of the state
- Instead: might have noisy measurements

→ Partially Observable Markov Decision Process (POMDP)

- Main ideas:
  - Based on the noisy measurements and knowledge about the dynamics, keep track of a probability distribution for the current state
  - Define a new MDP for which the probability distribution over current state is considered the state

Exploration vs. exploitation

- One of the milestone results: Kearns and Singh, Explicit Exploration and Exploitation (E^3), 2002
- Question/Problem:
  - Given an MDP with unknown transition model.
  - Can we
    - (a) explicitly decide to take actions that will assist us in building a transition model of the MDP, and
    - (b) detect when we have a sufficiently accurate model and start exploiting?

Basic idea of E^3 algorithm

- Repeat forever
  - Based on all data seen so far, partition the state space is a set of “known” states and a set of “unknown” states. [A state is known when each action in that state has been observed sufficiently often.]
  - If currently in a “known” state:
    - Lump all unknown states together in one absorbing meta-state. Give the meta-state a reward of 1. Give all known states a reward of zero. Find the optimal policy in this new MDP.
      - If the optimal policy has a value of zero (or low enough): exit. [No need for exploration anymore.]
      - Otherwise: Execute the optimal action for the current state.
  - If currently in a “unknown” state:
    - Take the action that has been taken least often in this state.

Technical aspects underneath E^3

- Simulation lemma: if the transition models and reward models of two MDPs are sufficiently close, then the optimal policy in one will also be close to optimal in the other
- After having seen a state-action pair sufficiently often, with high probability the data based transition model estimate will be accurate

→ Their analysis provides a finite time result (as opposed to asymptotic, such as for Q learning, sarsa, etc.)

- Various extensions since:
  - Brafman and Tenneholtz, Rmax
  - Kakade + al, Metric E3
  - Kearns and Koller, E3 in MDP w/transition model ~ temporal Bayes net

Hierarchical RL

- Main idea: use hierarchical domain knowledge to speed up RL
- I posted one representative paper onto the course website, should be a reasonable starting point if you wanted to find out more.

RL summary

- Exact methods: VI, PI, GPI, LP
- Model-free methods: TD, Q, sarsa
  - Batch versions: LSTD (recursive version: RLSTD), LSPI
- Function approximation:
  - Contractions – infinity norm, 2norm weighted by state visitation frequency
  - Approximate LP
- Policy gradient methods:
  - analytical, finite difference, likelihood ratio
  - Gradient <-> natural gradient
- Imitation learning:
  - Behavioral cloning <-> inverse RL

CS 287: Advanced Robotics
Fall 2009

Lecture 21:
HMMs, Bayes filter, smoother, Kalman filters

Pieter Abbeel
UC Berkeley EECS

Overview

- Thus far:
  - Optimal control and reinforcement learning
  - We always assumed we got to observe the state at each time and the challenge was to choose a good action
- Current and next set of lectures
  - *The state is not observed*
  - Instead, we get some sensory information about the state

  → Challenge: compute a probability distribution over the state which accounts for the sensory information ("evidence") which we have observed.

Examples

- **Helicopter**
  - A choice of state: position, orientation, velocity, angular rate
  - Sensors:
    - GPS : noisy estimate of position (sometimes also velocity)
    - Inertial sensing unit: noisy measurements from (i) 3-axis gyro [=angular rate sensor], (ii) 3-axis accelerometer [=measures acceleration + gravity; e.g., measures (0,0,0) in free-fall], (iii) 3-axis magnetometer
- **Mobile robot inside building**
  - A choice of state: position and heading
  - Sensors:
    - Odometry (=sensing motion of actuators): e.g., wheel encoders
    - Laser range finder: measures time of flight of a laser beam between departure and return (return is typically happening when hitting a surface that reflects the beam back to where it came from)

Probability review

For any random variables X, Y we have:

- **Definition of conditional probability:**

  $P(X=x \mid Y=y) = P(X=x, Y=y) / P(Y=y)$

- **Chain rule:** (follows directly from the above)

  $P(X=x, Y=y) = P(X=x)\, P(Y=y \mid X=x)$

  $= P(Y=y)\, P(X=x \mid Y=y)$

- **Bayes rule:** (really just a re-ordering of terms in the above)

  $P(X=x \mid Y=y) = P(Y=y \mid X=x)\, P(X=x) / P(Y=y)$

- **Marginalization:**

  $P(X=x) = \sum_y P(X=x, Y=y)$

Note: no assumptions beyond X, Y being random variables are made for any of these to hold true (and when we divide by something, that something is not zero)

Probability review

For any random variables X, Y, Z, W we have:

- **Conditional probability:** (can condition on a third variable z throughout)

  $P(X=x \mid Y=y, Z=z) = P(X=x, Y=y \mid Z=z) / P(Y=y \mid Z=z)$

- **Chain rule:**

  $P(X=x, Y=y, Z=z, W=w) = P(X=x)\, P(Y=y \mid X=x)\, P(Z=z \mid X=x, Y=y)\, P(W=w \mid X=x, Y=y, Z=z)$

- **Bayes rule:** (can condition on other variable z throughout)

  $P(X=x \mid Y=y, Z=z) = P(Y=y \mid X=x, Z=z)\, P(X=x \mid Z=z) / P(Y=y \mid Z=z)$

- **Marginalization:**

  $P(X=x \mid W=w) = \sum_{y,z} P(X=x, Y=y, Z=z \mid W=w)$

Note: no assumptions beyond X, Y, Z, W being random variables are made for any of these to hold true (and when we divide by something, that something is not zero)

Independence

- Two random variables X and Y are independent iff

  for all x, y : $P(X=x, Y=y) = P(X=x)\, P(Y=y)$

- Representing a probability distribution over a set of random variables $X_1, X_2, \ldots, X_T$ in its most general form can be expensive.
  - E.g., if all X_i are binary valued, then there would be a total of 2^T possible instantiations and it would require 2^T-1 numbers to represent the probability distribution.
- However, if we assumed the random variables were independent, then we could very compactly represent the joint distribution as follows:
  - $P(X_1=x_1, X_2=x_2, \ldots, X_T=x_T) = P(X_1=x_1)\, P(X_2=x_2) \ldots P(X_T=x_T)$
  - Thanks to the independence assumptions, for the binary case, we went from requiring 2^T-1 parameters, to only requiring T parameters!
- Unfortunately independence is often too strong an assumption …

Conditional independence

- Two random variables X and Y are conditionally independent given a third random variable Z iff

  for all x, y, z : P(X=x, Y=y | Z=z) = P(X=x | Z=z) P(Y=y | Z=z)

- Chain rule (which holds true for all distributions, no assumptions needed):
  - P(X=x,Y=y,Z=z,W=w) = P(X=x)P(Y=y|X=x)P(Z=z|X=x,Y=y)P(W=w|X=x,Y=y,Z=z)

  → For binary variables the representation requires 1 + 2*1 + 4*1 + 8*1 = 2^4-1 numbers (just like a full joint probability table)

- Now assume Z independent of X given Y, and assume W independent of X and Y given Z, then we obtain:
  - P(X=x,Y=y,Z=z,W=w) = P(X=x)P(Y=y|X=x)P(Z=z|Y=y)P(W=w|Z=z)
  - → For binary variables the representation requires 1 + 2*1 + 2*1 + 2*1 = 1+(4-1)*2 numbers --- significantly less!!

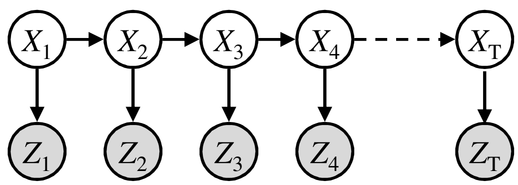
Markov Models

- Models a distribution over a set of random variables $X_1, X_2, \ldots, X_T$ where the index is typically associated with some notion of time.
- Markov models make the assumption:
  - X_t is independent of $X_1, \ldots, X_{t-2}$ when given X_{t-1}
- Chain rule: (always holds true, not just in Markov models!)
  - $P(X_1 = x_1, X_2 = x_2, \ldots, X_T = x_T) = \prod_t P(X_t = x_t \mid X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \ldots, X_1 = x_1)$
- Now apply the Markov conditional independence assumption:
  - $P(X_1 = x_1, X_2 = x_2, \ldots, X_T = x_T) = \prod_t P(X_t = x_t \mid X_{t-1} = x_{t-1})$ (1)
  - → in binary case: 1 + 2*(T-1) numbers required to represent the joint distribution over all variables (vs. $2^T - 1$)
- Graphical representation: a variable X_t receives an arrow from the variables appearing in its conditional probability in the expression for the joint distribution (1) [called a Bayesian network or Bayes net representation]

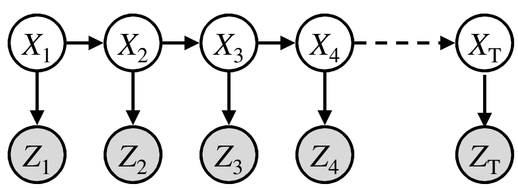
$$X_1 \to X_2 \to X_3 \to X_4 \dashrightarrow X_T$$

Hidden Markov Models

- Underlying Markov model over states X_t
  - Assumption 1: X_t independent of $X_1, \ldots, X_{t-2}$ given X_{t-1}
- For each state X_t there is a random variable Z_t which is a sensory measurement of X_t
  - Assumption 2: Z_t is assumed conditionally independent of the other variables *given* X_t
- This gives the following graphical (Bayes net) representation:



Hidden Markov Models



$P(X_1=x_1, Z_1=z_1, X_2=x_2, Z_2=z_2, \ldots, X_T=x_T, Z_T=z_T) =$

- Chain rule: (no assumptions)

  $P(X_1 = x_1)$

  $P(Z_1 = z_1 \mid X_1 = x_1)$

  $P(X_2 = x_2 \mid X_1 = x_1 , Z_1 = z_1)$

  $P(Z_2 = z_2 \mid X_1 = x_1, Z_1 = z_1, X_2 = x_2)$

  …

  $P(X_T = x_T \mid X_1 = x_1, Z_1 = z_1, \ldots , X_{T-1} = x_{T-1}, Z_{T-1} = z_{T-1})$

  $P(Z_T = z_T \mid X_1 = x_1, Z_1 = z_1, \ldots , X_{T-1} = x_{T-1}, Z_{T-1} = z_{T-1} , X_T = x_T)$

- HMM assumptions:

  $P(X_1 = x_1)$

  $P(Z_1 = z_1 \mid X_1 = x_1)$

  $P(X_2 = x_2 \mid X_1 = x_1)$

  $P(Z_2 = z_2 \mid X_2 = x_2)$

  …

  $P(X_T = x_T \mid X_{T-1} = x_{T-1})$

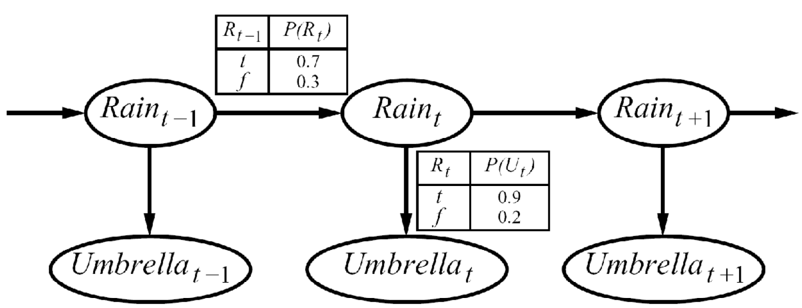
  $P(Z_T = z_T \mid X_T = x_T)$

Mini quiz

- What would the graph look like for a Bayesian network with no conditional independence assumptions?

- Our particular choice of ordering of variables in the chain rule enabled us to easily incorporate the HMM assumptions. What if we had chosen a different ordering in the chain rule expansion?

Example



- The HMM is defined by:
  - Initial distribution: $P(X_1)$
  - Transitions: $P(X|X_{-1})$
  - Observations: $P(Z|X)$

Real HMM Examples

- Robot localization:
  - Observations are range readings (continuous)
  - States are positions on a map (continuous)
- Speech recognition HMMs:
  - Observations are acoustic signals (continuous valued)
  - States are specific positions in specific words (so, tens of thousands)
- Machine translation HMMs:
  - Observations are words (tens of thousands)
  - States are translation options

Filtering / Monitoring

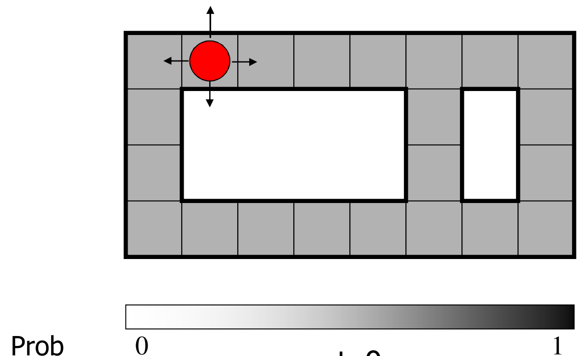
- Filtering, or monitoring, is the task of tracking the distribution

  $P(X_t \mid Z_1 = z_1 , Z_2 = z_2 , \ldots, Z_t = z_t)$

  over time. This distribution is called the belief state.
- We start with $P(X_0)$ in an initial setting, usually uniform
- As time passes, or we get observations, we update the belief state.
- The Kalman filter was invented in the 60's and first implemented as a method of trajectory estimation for the Apollo program. [See course website for a historical account on the Kalman filter. "From Gauss to Kalman"]

Example: Robot Localization

Example from Michael Pfeiffer



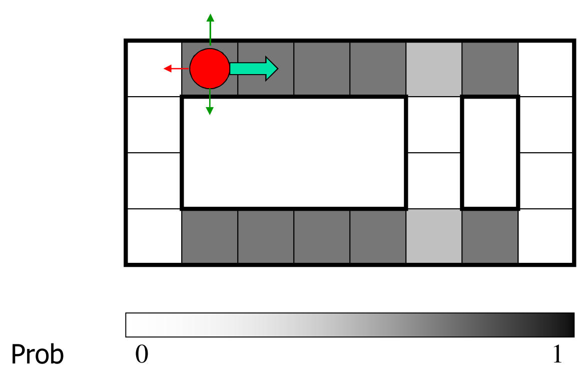
t=0

Sensor model: never more than 1 mistake

Know the heading (North, East, South or West)

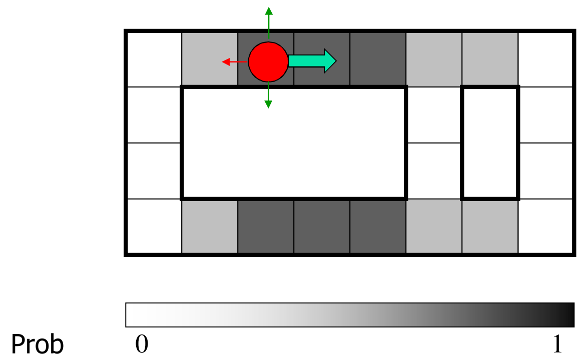
Motion model: may not execute action with small prob.

Example: Robot Localization



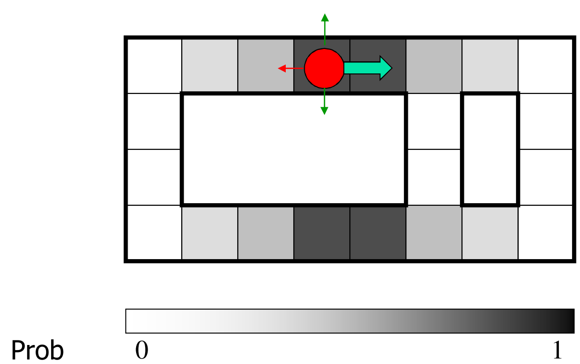
t=1

Example: Robot Localization



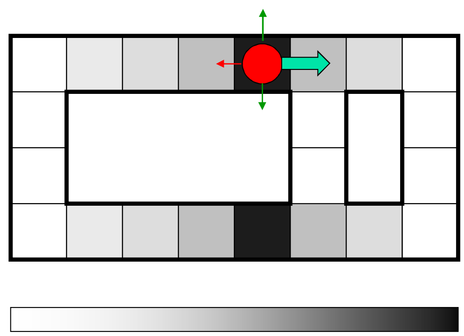
t=2

Example: Robot Localization



t=3

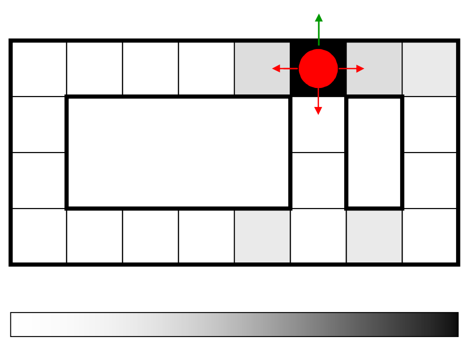
Example: Robot Localization



Prob 0 1

t=4

Example: Robot Localization

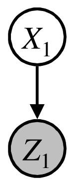


Prob 0 1

t=5

Inference: Base Cases

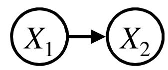
Incorporate observation



$$P(X_1|z_1)$$

$$\begin{aligned} P(x_1|z_1) &= P(x_1, z_1)/P(z_1) \\ &\propto P(x_1, z_1) \\ &= P(x_1)P(z_1|x_1) \end{aligned}$$

Time update



$$P(X_2)$$

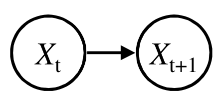
$$\begin{aligned} P(x_2) &= \sum_{x_1} P(x_1, x_2) \\ &= \sum_{x_1} P(x_1)P(x_2|x_1) \end{aligned}$$

Time update

- Assume we have current belief P(X | evidence to date)

$$P(x_t|e_{1:t})$$

- Then, after one time step passes:

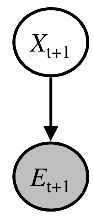


$$P(x_{t+1}|e_{1:t}) = \sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t})$$

Observation update

- Assume we have:

  $P(x_{t+1}|e_{1:t})$



- Then:

$$P(x_{t+1}|e_{1:t+1}) \propto P(e_{t+1}|x_{t+1})P(x_{t+1}|e_{1:t})$$

Algorithm

- Init $P(x_1)$ [e.g., uniformly]
- Observation update for time 0:

  $P(x_1|z_1) \propto P(z_1|x_1)P(x_1)$

- For t = 1, 2, …
  - Time update

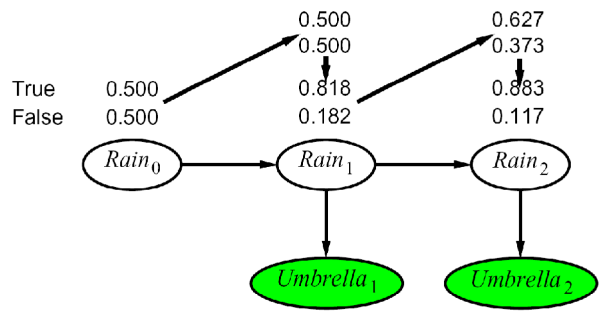
    $P(x_{t+1}|z_{1:t}) = \sum_{x_t} P(x_{t+1}|x_t)P(x_t|z_{1:t})$

  - Observation update

    $P(x_{t+1}|z_{1:t+1}) \propto P(z_{t+1}|x_{t+1})P(x_{t+1}|z_{1:t})$

- For continuous state / observation spaces: simply replace summation by integral

Example HMM



R_{t-1}	$P(R_t)$
T	0.7
F	0.3

R_t	$P(U_t)$
T	0.9
F	0.2

The Forward Algorithm

- Time/dynamics update and observation update in one:

$$P(x_t, z_{1:t}) = \sum_{x_{t-1}} P(x_{t-1}, x_t, z_{1:t})$$

$$= \sum_{x_{t-1}} P(x_{t-1}, z_{1:t-1}) P(x_t|x_{t-1}) P(z_t|x_t)$$

$$= P(z_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, z_{1:t-1})$$

- → recursive update
- Normalization:
  - Can be helpful for numerical reasons
  - However: lose information!

  → Can renormalize (for numerical reasons) + keep track of the normalization factor (to enable recovering all information)

The likelihood of the observations

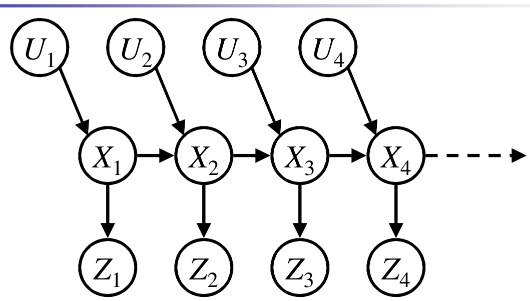
$$P(z_{1:t}) = \sum_{x_1, x_2, \ldots, x_t} P(x_{1:t}, z_{1:t}) = \sum_{x_1, x_2, \ldots, x_t} \prod_{k=1}^{t-1} P(x_{k+1}|x_k) P(z_k|x_k) P(z_t|x_t)$$

- The forward algorithm first sums over x_1, then over x_2 and so forth, which allows it to efficiently compute the likelihood at all times t, indeed:

$$P(z_{1:t}) = \sum_{x_t} P(x_t, z_{1:t})$$

- Relevance:
  - Compare the fit of several HMM models to the data
  - Could optimize the dynamics model and observation model to maximize the likelihood
  - Run multiple simultaneous trackers --- retain the best and split again whenever applicable (e.g., loop closures in SLAM, or different flight maneuvers)

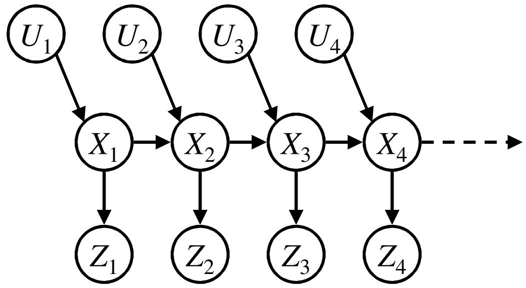
With control inputs



We know track:

$$P(x_t|z_{1:t}, u_{1:t})$$

With control inputs



- Control inputs known:
  - They can be simply seen as selecting a particular dynamics function
- Control inputs unknown:
  - Assume a distribution over them
- Above drawing assumes open-loop controls. This is rarely the case in practice. [Markov assumption is rarely the case either. Both assumptions seem to have given quite satisfactory results.]

Smoothing

- Thus far, filtering, which finds:
  - The distribution over states at time t given all evidence until time t:

$$P(x_t | z_{1:t}, u_{1:t})$$

  - The likelihood of the evidence up to time t:

$$P(z_{1:t} | u_{1:t})$$

- How about?

$$P(x_t | z_{1:T}, u_{1:T})$$

  - T < t : can simply run the forward algorithm until time t, but stop incorporating evidence from time T+1 onwards
  - T > t : need something else

Smoothing

$$
\begin{aligned}
P(x_t|z_{1:T}) &\propto P(x_t, z_{1:T}) \\
&= \sum_{x_1,x_2,\ldots,x_{t-1},x_{t+1},\ldots,x_T} P(x_{1:T}, z_{1:T}) \\
&= \sum_{x_1,x_2,\ldots,x_{t-1},x_{t+1},\ldots,x_T} \prod_{k=1}^{T} P(x_k|x_{k-1})P(z_k|x_k)
\end{aligned}
$$

- Sum as written has a number of terms exponential in T
- Key idea: order in which variables are being summed out affects computational complexity
  - *Forward algorithm* exploits summing out $x_1, x_2, \ldots, x_{t-1}$ in this order
  - Can similarly run a *backward algorithm*, which sums out $x_T, x_{T-1}, \ldots, x_{t+1}$ in this order

Smoothing

$$
\begin{aligned}
P(x_t, z_{1:T}) &= \sum_{x_1,x_2,\ldots,x_{t-1},x_{t+1},\ldots,x_T} P(x_{1:T}, z_{1:T}) \\
&= \sum_{x_1,x_2,\ldots,x_{t-1},x_{t+1},\ldots,x_T} \prod_{k=1}^{T} P(x_k|x_{k-1})P(z_k|x_k) \\
&= \underbrace{\left(\sum_{x_1,x_2,\ldots,x_{t-1}} \prod_{k=1}^{t} P(x_k|x_{k-1})P(z_k|x_k) \right)}_{\text{Forward algorithm computes this}} \underbrace{\left(\sum_{x_{t+1},x_{t+2},\ldots,x_T} \prod_{k=t+1}^{T} P(x_k|x_{k-1})P(z_k|x_k) \right)}_{\text{Backward algorithm computes this}}
\end{aligned}
$$

- Can be easily verified from the equations:
  - The factors in the right parentheses only contain $X_{t+1}, \ldots, X_T$, hence they act as a constant when summing out over $X_1, \ldots, X_{t-1}$ and can be brought outside the summation
- Can also be read off from the Bayes net graph / conditional independence assumptions:
  - $X_1, \ldots, X_{t-1}$ are conditionally of $X_{t+1}, \ldots, X_T$ *given* X_t

Backward algorithm

- Sum out x_T:

$$
\begin{aligned}
P(x_t, e_{1:T}) &= \sum_{x_1, x_2, \ldots, x_{t-1}, x_{t+1}, \ldots, x_T} \prod_{k=1}^{T} P(x_k|x_{k-1}) P(e_k|x_k) \\
&= \sum_{x_1, x_2, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{T-1}} \sum_{x_T} \prod_{k=1}^{T} P(x_k|x_{k-1}) P(e_k|x_k) \\
&= \sum_{x_1, x_2, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{T-1}} \prod_{k=1}^{T-1} P(x_k|x_{k-1}) P(e_k|x_k) \sum_{x_T} P(x_T|x_{T-1}) P(e_T|x_T) \\
&= \sum_{x_1, x_2, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{T-1}} \prod_{k=1}^{T-1} P(x_k|x_{k-1}) P(e_k|x_k) f_{T-1}(x_{T-1})
\end{aligned}
$$

- Can recursively compute for l=T, T-1, …:

$$
f_{l-1}(x_{l-1}) = \sum_{x_l} P(x_l|x_{l-1}) P(e_l|x_l) f_l(x_l)
$$

Smoother algorithm

- Run forward algorithm, which gives
  - $P(x_t, z_1, \ldots, z_t)$ for all t
- Run backward algorithm, which gives
  - $f_t(x_t)$ for all t
- Find
  - $P(x_t, z_1, \ldots, z_T) = P(x_t, z_1, \ldots, z_t)\, f_t(x_t)$
  - If desirable, can renormalize and find $P(x_t \mid z_1, \ldots, z_T)$

Bayes filters

- Recursively compute
  - $P(x_t, z_{1:t-1}) = \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1} \mid z_{1:t-1})$
  - $P(x_t, z_{1:t}) = P(x_t, z_{1:t-1}) P(z_t \mid x_t)$
- Tractable cases:
  - State space finite and sufficiently small

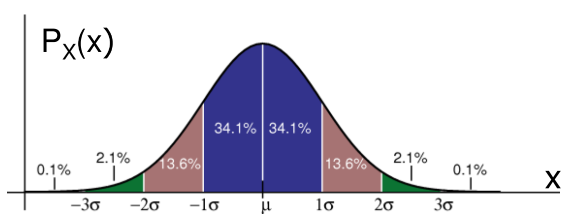
    (what we have in some sense considered so far)
  - Systems with linear dynamics and linear observations and Gaussian noise

    → Kalman filtering

Univariate Gaussian

- Gaussian distribution with mean μ, and standard deviation σ:

$$N(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



Properties of Gaussians

$$N(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

- Mean:

$$EX = \int x \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) = \mu$$

- Variance:

$$E((X-\mu)^2) = \int (x-\mu)^2 \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) = \sigma^2$$

Central limit theorem (CLT)

- Classical CLT:
  - Let $X_1, X_2, \ldots$ be an infinite sequence of *independent* random variables with $E X_i = \mu$, $E(X_i - \mu)^2 = \sigma^2$
  - Define $Z_n = ((X_1 + \ldots + X_n) - n \mu) / (\sigma n^{1/2})$
  - Then for the limit of n going to infinity we have that Z_n is distributed according to N(0,1)
- Crude statement: things that are the result of the addition of lots of small effects tend to become Gaussian.

Multi-variate Gaussians

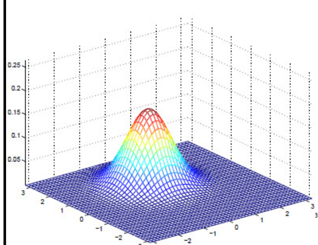
$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

$$EX = \int x p(x;\mu,\Sigma) = \mu$$

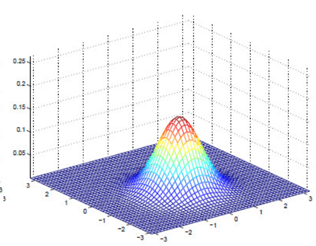
$$E[(X_i-\mu_i)(X_j-\mu_j)] = \int (x_i-\mu_i)(x_j-\mu_j)p(x;\mu,\Sigma) = \Sigma_{ij}$$

$$\int \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right) = 1$$

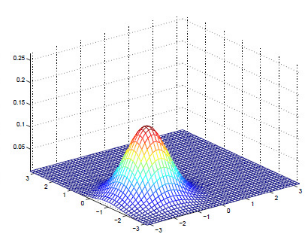
Multi-variate Gaussians: examples



- μ = [1; 0]
- Σ = [1 0; 0 1]

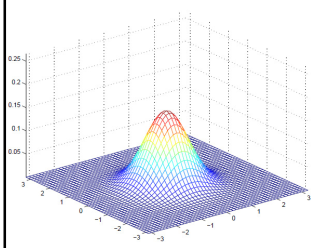


- μ = [-.5; 0]
- Σ = [1 0; 0 1]

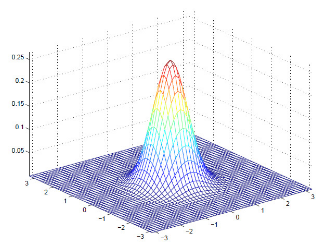


- μ = [-1; -1.5]
- Σ = [1 0; 0 1]

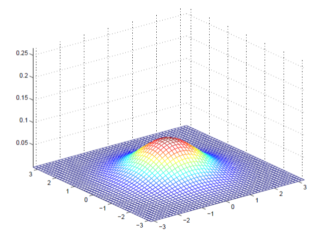
Multi-variate Gaussians: examples



- μ = [0; 0]
- Σ = [1 0 ; 0 1]

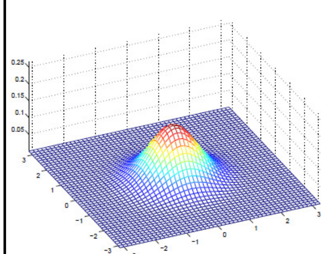


- μ = [0; 0]
- Σ = [.6 0 ; 0 .6]

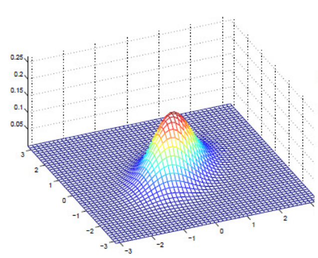


- μ = [0; 0]
- Σ = [2 0 ; 0 2]

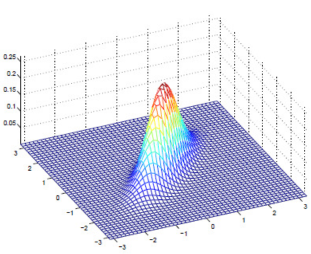
Multi-variate Gaussians: examples



- μ = [0; 0]
- Σ = [1 0; 0 1]

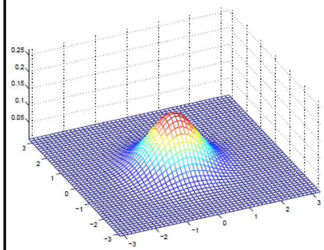
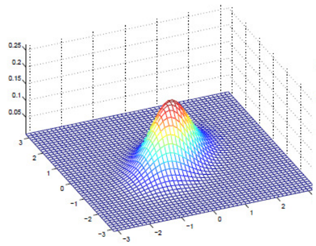
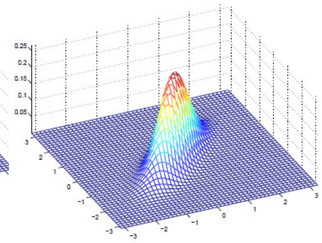


- μ = [0; 0]
- Σ = [1 0.5; 0.5 1]



- μ = [0; 0]
- Σ = [1 0.8; 0.8 1]

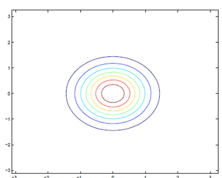
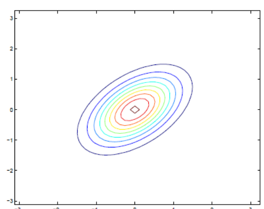
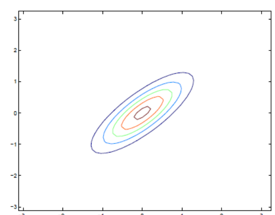
Multi-variate Gaussians: examples

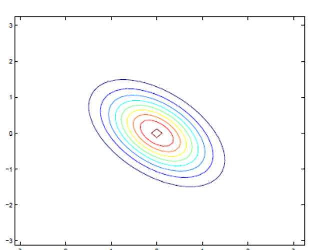
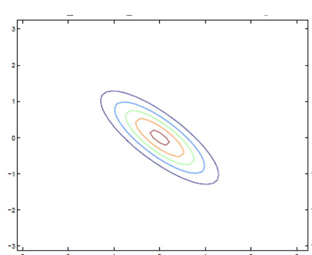
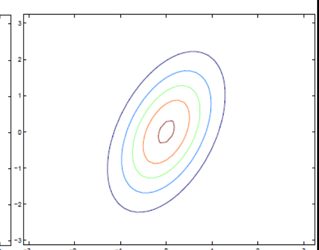
- μ = [0; 0]
- Σ = [1 0; 0 1]

- μ = [0; 0]
- Σ = [1 0.5; 0.5 1]

- μ = [0; 0]
- Σ = [1 0.8; 0.8 1]

Multi-variate Gaussians: examples

- μ = [0; 0]
- Σ = [1 -0.5 ; -0.5 1]

- μ = [0; 0]
- Σ = [1 -0.8 ; -0.8 1]

- μ = [0; 0]
- Σ = [3 0.8 ; 0.8 1]

Discrete Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

with a measurement

$$z_t = C_t x_t + \delta_t$$

47

Components of a Kalman Filter

A_t Matrix (nxn) that describes how the state evolves from t to $t\text{-}1$ without controls or noise.

B_t Matrix (nxl) that describes how the control u_t changes the state from t to $t\text{-}1$.

C_t Matrix (kxn) that describes how to map the state x_t to an observation z_t.

ε_t δ_t Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

48

Linear Gaussian Systems: Initialization

- Initial belief is normally distributed:

$$bel(x_0) = N\left(x_0; \mu_0, \Sigma_0\right)$$

53

Linear Gaussian Systems: Dynamics

- Dynamics are linear function of state and control plus additive noise:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

$$p(x_t \mid u_t, x_{t-1}) = N\left(x_t; A_t x_{t-1} + B_t u_t, R_t\right)$$

$$\begin{array}{cccc} \overline{bel}(x_t) = \int & p(x_t \mid u_t, x_{t-1}) & & bel(x_{t-1})\, dx_{t-1} \\ & \Downarrow & & \Downarrow \\ & \sim N\left(x_t; A_t x_{t-1} + B_t u_t, R_t\right) & & \sim N\left(x_{t-1}; \mu_{t-1}, \Sigma_{t-1}\right) \end{array}$$

54

Linear Gaussian Systems: Dynamics

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) \qquad bel(x_{t-1})\, dx_{t-1}$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$\sim N\left(x_t; A_t x_{t-1} + B_t u_t, R_t\right) \quad \sim N\left(x_{t-1}; \mu_{t-1}, \Sigma_{t-1}\right)$$

$$\Downarrow$$

$$\overline{bel}(x_t) = \eta \int \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1} (x_t - A_t x_{t-1} - B_t u_t)\right\}$$

$$\exp\left\{-\frac{1}{2}(x_{t-1} - \mu_{t-1})^T \Sigma_{t-1}^{-1} (x_{t-1} - \mu_{t-1})\right\} dx_{t-1}$$

$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

55

Proof: completion of squares

- To integrate out x_{t-1}, re-write the integrand in the format:

$$\int f(\mu_{t-1}, \Sigma_{t-1}, x_t) \left(\frac{1}{(2\pi)^{\frac{d}{2}} |S|^{\frac{1}{2}}} \exp\left(\frac{-1}{2}(x_{t-1} - m) S^{-1} (x_{t-1} - m) \right) \right) dx_{t-1}$$

- This integral is readily computed (integral of a multivariate Gaussian times a constant = that constant) to be

  $f(\mu_{t-1}, \Sigma_{t-1}, x_t)$

- Inspection of f will show that it is a multi-variate Gaussian in x_t with the mean and covariance as shown on previous slide.

Properties of Gaussians

- We just showed:

$$\left.\begin{array}{r} X \sim N(\mu,\Sigma) \\ Y = AX + B \end{array}\right\} \Rightarrow \quad Y \sim N(A\mu + B, A\Sigma A^T)$$

- We stay in the "Gaussian world" as long as we start with Gaussians and perform only linear transformations.
- Now we know this, we could find μ_Y and Σ_Y without computing integrals by directly computing the expected values:

$$\mathrm{E}[Y] = \mathrm{E}[AX + B] = A\mathrm{E}[X] + B = A\mu + B$$

$$\begin{aligned} \Sigma_{YY} &= \mathrm{E}[(Y - \mathrm{E}[Y])(Y - \mathrm{E}[Y])^\top] = \mathrm{E}[(AX + B - A\mu - B)(AX + B - A\mu - B)^\top] \\ &= \mathrm{E}[A(X - \mu)(X - \mu)^\top A^\top] = A\mathrm{E}[(X - \mu)(X - \mu)^\top]A^\top = A\Sigma A^\top \end{aligned}$$

Self-quiz

Test your understanding of the completion of squares trick! Let $A \in \mathbf{R}^{n \times n}$ be a positive definite matrix, $b \in \mathbf{R}^n$, and $c \in \mathbf{R}$. Prove that

$$\int_{x \in \mathbf{R}^n} \exp\left(-\frac{1}{2}x^T A x - x^T b - c\right) dx = \frac{(2\pi)^{n/2}}{|A|^{1/2} \exp(c - \frac{1}{2}b^T A^{-1} b)}.$$

Linear Gaussian Systems: Observations

- Observations are linear function of state plus additive noise:

$$z_t = C_t x_t + \delta_t$$

$$p(z_t \mid x_t) = N(z_t; C_t x_t, Q_t)$$

$$
\begin{array}{cccc}
bel(x_t) = & \eta & p(z_t \mid x_t) & \overline{bel}(x_t) \\
 & & \Downarrow & \Downarrow \\
 & & \sim N(z_t; C_t x_t, Q_t) & \sim N(x_t; \bar{\mu}_t, \bar{\Sigma}_t)
\end{array}
$$

59

Linear Gaussian Systems: Observations

$$
\begin{array}{cccc}
bel(x_t) = & \eta & p(z_t \mid x_t) & \overline{bel}(x_t) \\
 & & \Downarrow & \Downarrow \\
 & & \sim N(z_t; C_t x_t, Q_t) & \sim N(x_t; \bar{\mu}_t, \bar{\Sigma}_t) \\
 & & \Downarrow &
\end{array}
$$

$$bel(x_t) = \eta \exp\left\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t)\right\} \exp\left\{-\frac{1}{2}(x_t - \bar{\mu}_t)^T \bar{\Sigma}_t^{-1} (x_t - \bar{\mu}_t)\right\}$$

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \end{cases} \quad \text{with} \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

60

Proof: completion of squares

- Re-write the expression for *bel(x_t)* in the format:

$$f(\bar{\mu}_t, \bar{\Sigma}_t, C_t, Q_t)\left(\frac{1}{(2\pi)^{\frac{d}{2}}|S|^{\frac{1}{2}}}\exp\left(\frac{-1}{2}(x_t - m)S^{-1}(x_t - m)\right)\right)$$

- f is the normalization factor
- The expression in parentheses is a multi-variate Gaussian in x_t. Its parameters m and S can be identified to satisfy the expressions for the mean and covariance on the previous slide.

Kalman Filter Algorithm

Algorithm **Kalman_filter**(μ_{t-1}, Σ_{t-1}, u_t, z_t):

Prediction:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correction:

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$$

Return μ_t, Σ_t

62

How to derive these updates

- Simply work through the integrals
  - Key "trick": completion of squares

- If your derivation results in a different format → apply matrix inversion lemma to prove equivalence

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

CS 287: Advanced Robotics
Fall 2009

Lecture 22:
HMMs, Kalman filters

Pieter Abbeel
UC Berkeley EECS

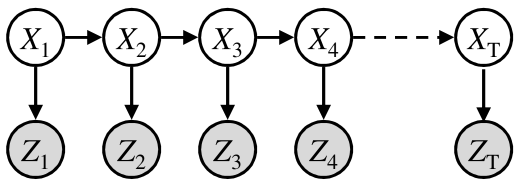
Overview

- Current and next set of lectures
  - *The state is not observed*
  - Instead, we get some sensory information about the state

  → Challenge: compute a probability distribution over the state which accounts for the sensory information (“evidence”) which we have observed.

Hidden Markov Models

- Underlying Markov model over states X_t
  - Assumption 1: X_t independent of $X_1, \ldots, X_{t-2}$ given X_{t-1}
- For each state X_t there is a random variable Z_t which is a sensory measurement of X_t
  - Assumption 2: Z_t is assumed conditionally independent of the other variables *given* X_t
- This gives the following graphical (Bayes net) representation:



Filtering in HMM

- Init $P(x_1)$ [e.g., uniformly]
- Observation update for time 0:

$$P(x_1|z_1) \propto P(z_1|x_1)P(x_1)$$

- For t = 1, 2, …
  - Time update

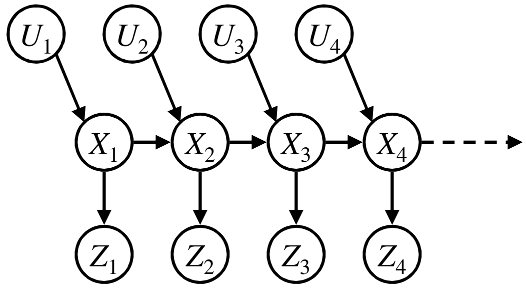
$$P(x_{t+1}|z_{1:t}) = \int_{x_t} P(x_{t+1}|x_t)P(x_t|z_{1:t})dx_t$$

  - Observation update

$$P(x_{t+1}|z_{1:t+1}) \propto P(z_{t+1}|x_{t+1})P(x_{t+1}|z_{1:t})$$

- For discrete state / observation spaces: simply replace integral by summation

With control inputs



- Control inputs known:
  - They can be simply seen as selecting a particular dynamics function
- Control inputs unknown:
  - Assume a distribution over them
- Above drawing assumes open-loop controls. This is rarely the case in practice. [Markov assumption is rarely the case either. Both assumptions seem to have given quite satisfactory results.]

Discrete-time Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

with a measurement

$$z_t = C_t x_t + \delta_t$$

6

Kalman Filter Algorithm

Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correction:

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

7

Intermezzo: information filter

- From an analytical point of view == Kalman filter
- Difference: keep track of the inverse covariance rather than the covariance matrix

  [matter of some linear algebra manipulations to get into this form]
- Why interesting?
  - Inverse covariance matrix = 0 is easier to work with than covariance matrix = infinity (case of complete uncertainty)
  - Inverse covariance matrix is often sparser than the covariance matrix --- for the "insiders": inverse covariance matrix entry (i,j) = 0 if x_i is conditionally independent of x_j given some set $\{x_k, x_l, \ldots\}$
  - Downside: when extended to non-linear setting, need to solve a linear system to find the mean (around which one can then linearize)
  - See Probabilistic Robotics pp. 78-79 for more in-depth pros/cons

Matlab code data generation example

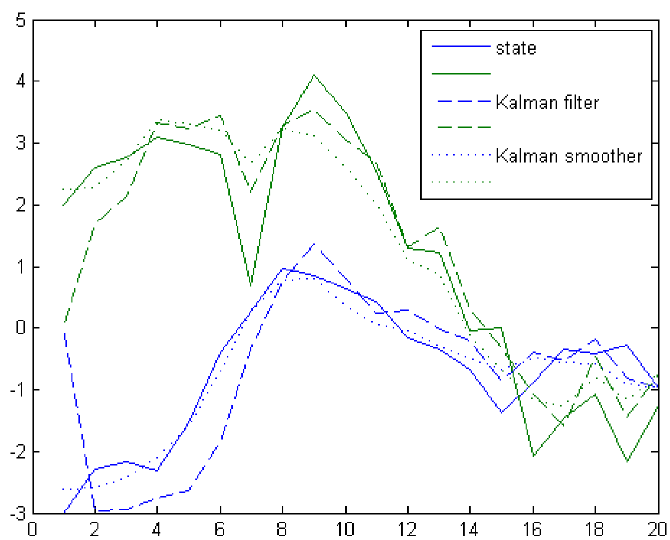
- A = [0.99 0.0074; -0.0136 0.99]; C = [1 1 ; -1 +1];
- x(:,1) = [-3;2];
- Sigma_w = diag([.3 .7]); Sigma_v = [2 .05; .05 1.5];
- w = randn(2,T); w = sqrtm(Sigma_w)*w; v = randn(2,T); v = sqrtm(Sigma_v)*v;
- for t=1:T-1

  x(:,t+1) = A * x(:,t) + w(:,t);

  y(:,t) = C*x(:,t) + v(:,t);

  end
- % now recover the state from the measurements
- P_0 = diag([100 100]); x0 =[0; 0];
- % run Kalman filter and smoother here
- % + plot

Kalman filter/smoother example



Kalman filter property

- If system is observable (=dual of controllable!) then Kalman filter will converge to the true state.
- System is observable iff

  $O = [C ; CA ; CA^2 ; \ldots ; CA^{n-1}]$ is full column rank (1)

Intuition: if no noise, we observe y_0, y_1, … and we have that the unknown initial state x_0 satisfies:

$y_0 = C\, x_0$

$y_1 = CA\, x_0$

...

$y_K = CA^K\, x_0$

This system of equations has a unique solution x_0 iff the matrix $[C; CA; \ldots CA^K]$ has full column rank. B/c any power of a matrix higher than n can be written in terms of lower powers of the same matrix, condition (1) is sufficient to check (i.e., the column rank will not grow anymore after having reached K=n-1).

Simple self-quiz

- The previous slide assumed zero control inputs at all times. Does the same result apply with non-zero control inputs? What changes in the derivation?

Kalman Filter Summary

- Highly efficient: Polynomial in measurement dimensionality k and state dimensionality n:
  $O(k^{2.376} + n^2)$

- Optimal for linear Gaussian systems!

- Most robotics systems are nonlinear!

→ Extended Kalman filter (EKF)
→ Unscented Kalman filter (UKF)

[And also: particle filter (next lecture)]

13

Announcement: PS2

- I provided a game log of me playing for your convenience.
- However, to ensure you fully understand, I suggest you follow the following procedure:
  - Code up a simple heuristic policy to collect samples from the state space for question 1. Then use these samples as your state samples for ALP and for approximate value iteration.
  - Play the game yourself for question 2 and have it learn to clone your playing style.
- You don't *need* to follow the above procedure, but I strongly suggest to in case you have any doubt about how the algorithms in Q1 and Q2 operate, b/c following the above procedure will force you more blatantly to see the differences (and can only increase you ability to hand in a good PS2).

Announcements

- PS1: will get back to you over the weekend, likely Saturday
- Milestone: ditto
- PS2: don't underestimate it!
- Office hours: canceled today. Feel free to set appointment over email. Also away on Friday actually. Happy to come in on Sat or Sun afternoon by appointment.
- Tuesday 4-5pm 540 Cory: Hadas Kress-Gazit (Cornell)

High-level tasks to correct low-level robot control
In this talk I will present a formal approach to creating robot controllers that ensure the robot satisfies a given high level task. I will describe a framework in which a user specifies a complex and reactive task in Structured English. This task is then automatically translated, using temporal logic and tools from the formal methods world, into a hybrid controller. This controller is guaranteed to control the robot such that its motion and actions satisfy the intended task, in a variety of different environments.

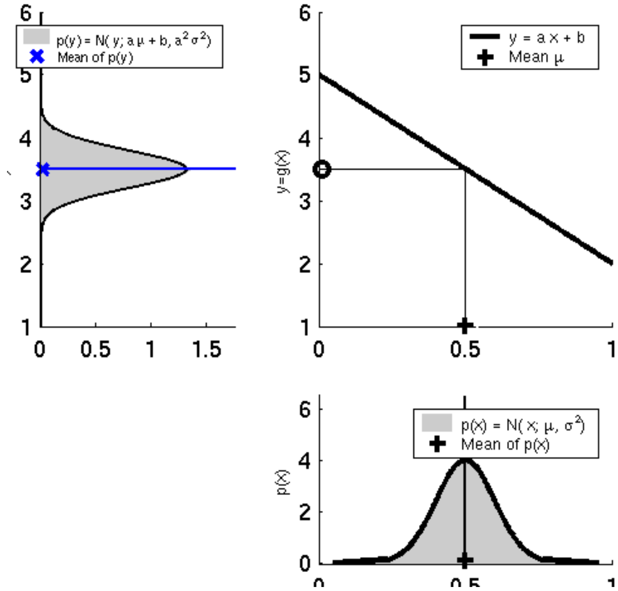
Nonlinear Dynamic Systems

- Most realistic robotic problems involve nonlinear functions

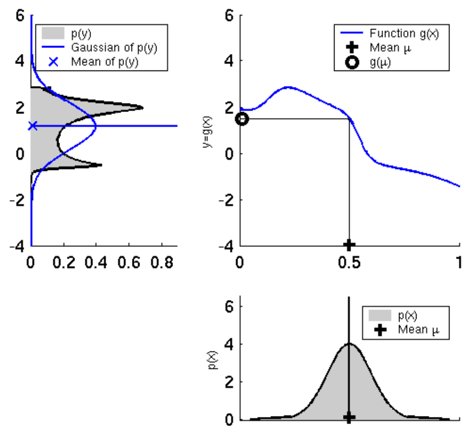
$$x_t = g(u_t, x_{t-1}) + noise$$

$$z_t = h(x_t) + noise$$

Linearity Assumption Revisited

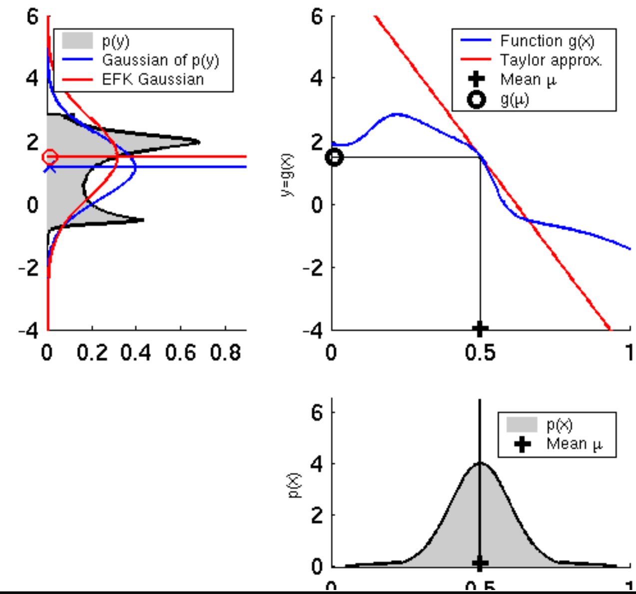


Non-linear Function

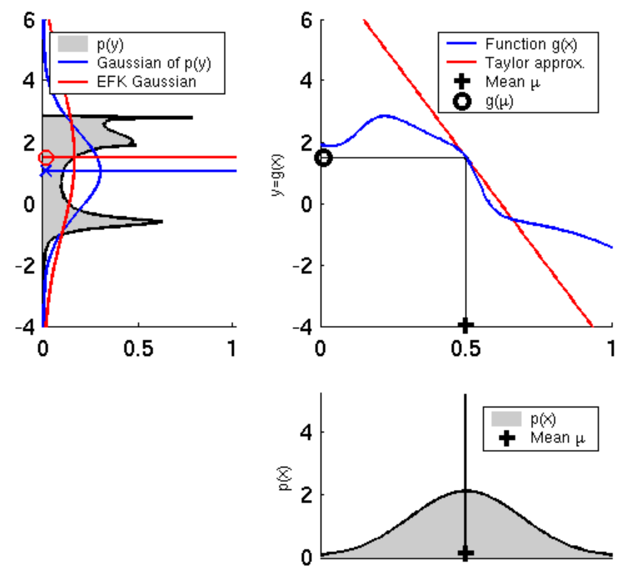


Throughout: "Gaussian of P(y)" is the Gaussian which minimizes the KL-divergence with P(y). It turns out that this means the Gaussian with the same mean and covariance as P(y).

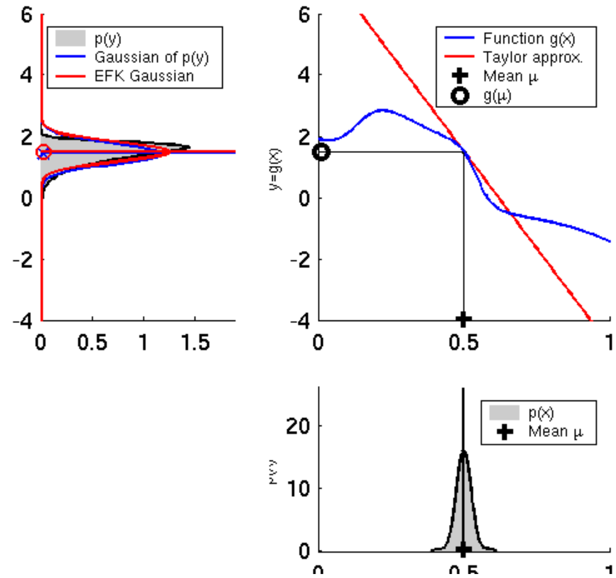
EKF Linearization (1)



EKF Linearization (2)



EKF Linearization (3)



EKF Linearization: First Order Taylor Series Expansion

- Prediction:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1})$$

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \, (x_{t-1} - \mu_{t-1})$$

- Correction:

$$h(x_t) \approx h(\overline{\mu}_t) + \frac{\partial h(\overline{\mu}_t)}{\partial x_t} (x_t - \overline{\mu}_t)$$

$$h(x_t) \approx h(\overline{\mu}_t) + H_t \, (x_t - \overline{\mu}_t)$$

EKF Algorithm

Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \quad \longleftarrow \quad \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \quad \longleftarrow \quad \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correction:

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \quad \longleftarrow \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) \quad \longleftarrow \quad \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$

$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \quad \longleftarrow \quad \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

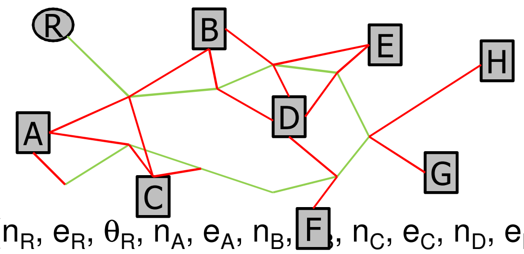
Return μ_t, Σ_t

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \qquad G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

Simultaneous Localization and Mapping (SLAM)

- Robot navigating in unknown environment
- Perception
  - Environment: typically through laser-range finders, sonars, cameras
  - Odometry (its own motion): inertial sensing, wheel encoders, (if outdoors and clear sky-view: gps)

Simultaneous Localization and Mapping (SLAM)



- State: (n_R, e_R, θ_R, n_A, e_A, n_B, e_B, n_C, e_C, n_D, e_D, n_E, e_E, n_F, e_F, n_G, e_G, n_H, e_H)
- Transition model:
  - Robot motion model; Landmarks stay in place

Simultaneous Localization and Mapping (SLAM)

- In practice: robot is not aware of all landmarks from the beginning
- Moreover: no use in keeping track of landmarks the robot has not received any measurements about

→ Incrementally grow the state when new landmarks get encountered.

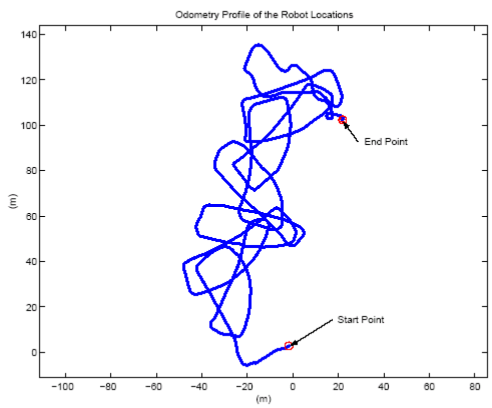
Simultaneous Localization and Mapping (SLAM)

- Landmark measurement model: robot measures [x_k; y_k], the position of landmark k expressed in coordinate frame attached to the robot:
  - $h(n_R, e_R, \theta_R, n_k, e_k) = [x_k; y_k] = R(\theta) \, ([n_k; e_k] - [n_R; e_R])$
- Often also some odometry measurements
  - E.g., wheel encoders
  - As they measure the control input being applied, they are often incorporated directly as control inputs (why?)

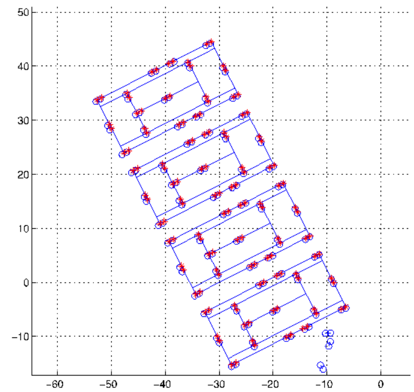
EKF SLAM Application

[courtesy by J. Leonard]

EKF SLAM Application



odometry



estimated trajectory

[courtesy by John Leonard]

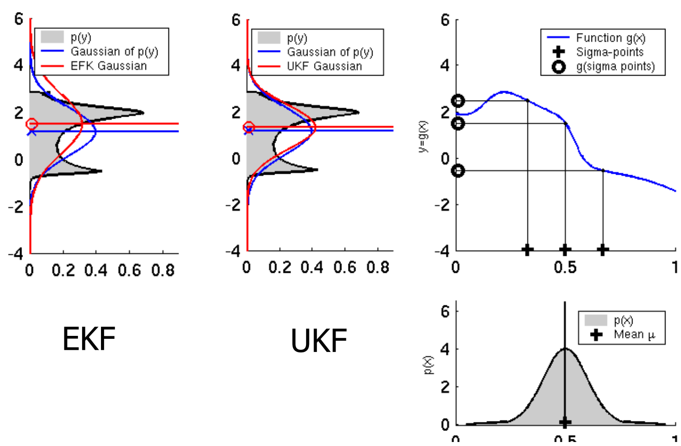
EKF-SLAM: practical challenges

- Defining landmarks
  - Laser range finder: Distinct geometric features (e.g. use RANSAC to find lines, then use corners as features)
  - Camera: “interest point detectors”, textures, color, …
- Often need to track multiple hypotheses
  - Data association/Correspondence problem: when seeing features that constitute a landmark --- Which landmark is it?
  - Closing the loop problem: how to know you are closing a loop?
  - → Can split off multiple EKFs whenever there is ambiguity;
  - → Keep track of the likelihood score of each EKF and discard the ones with low likelihood score
- Computational complexity with large numbers of landmarks.

EKF Summary

- Highly efficient: Polynomial in measurement dimensionality *k* and state dimensionality *n*:
  $O(k^{2.376} + n^2)$
- Not optimal!
- Can diverge if nonlinearities are large!
- Works surprisingly well even when all assumptions are violated!
- Note duality with linearizing a non-linear system and then running LQR back-ups to obtain the optimal linear controller!

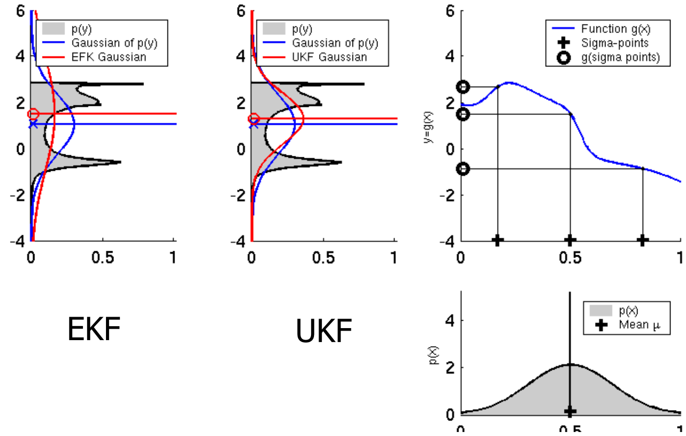
Linearization via Unscented Transform



EKF

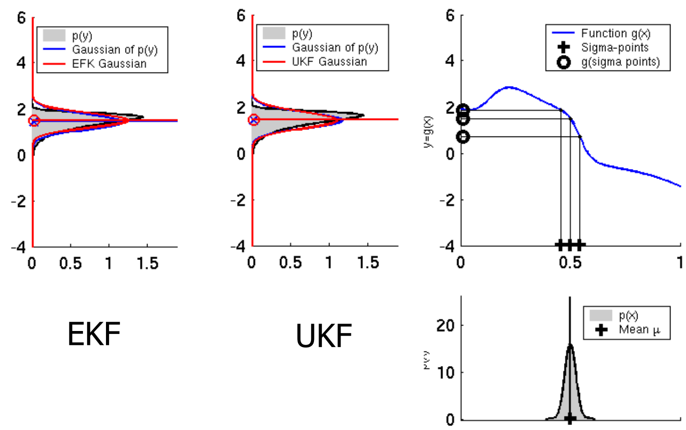
UKF

UKF Sigma-Point Estimate (2)

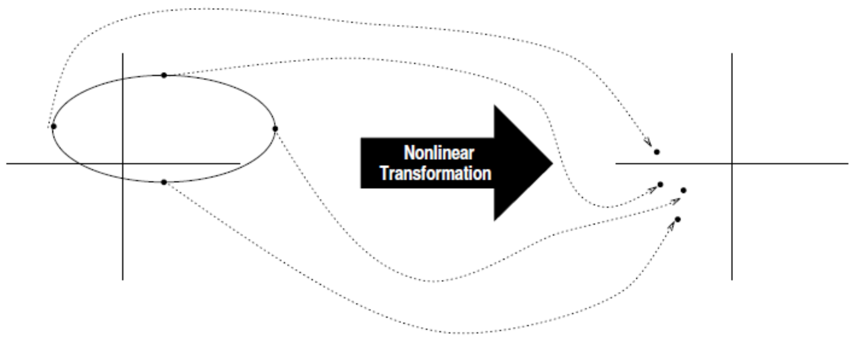


38

UKF Sigma-Point Estimate (3)



UKF Sigma-Point Estimate (4)



UKF intuition why it can perform better

- Assume we know the distribution over X and it has a mean \bar{x}
- Y = f(X)

$$\mathbf{f}[\mathbf{x}] = \mathbf{f}[\bar{\mathbf{x}} + \boldsymbol{\delta}\mathbf{x}]$$
$$= \mathbf{f}[\bar{\mathbf{x}}] + \nabla\mathbf{f}\boldsymbol{\delta}\mathbf{x} + \frac{1}{2}\nabla^2\mathbf{f}\boldsymbol{\delta}\mathbf{x}^2 + \frac{1}{3!}\nabla^3\mathbf{f}\boldsymbol{\delta}\mathbf{x}^3 + \frac{1}{4!}\nabla^4\mathbf{f}\boldsymbol{\delta}\mathbf{x}^4 + \cdots$$

$$\bar{\mathbf{y}} = \mathbf{f}[\bar{\mathbf{x}}] + \frac{1}{2}\nabla^2\mathbf{f}\,\mathbf{P}_{xx} + \frac{1}{2}\nabla^4\mathbf{f}\,\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right] + \cdots$$

$$\mathbf{P}_{yy} = \nabla\mathbf{f}\,\mathbf{P}_{xx}(\nabla\mathbf{f})^T + \frac{1}{2\times 4!}\nabla^2\mathbf{f}\left(\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right] - \mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^2\mathbf{P}_{yy}\right] - \mathrm{E}\left[\mathbf{P}_{yy}\boldsymbol{\delta}\mathbf{x}^2\right] + \mathbf{P}_{yy}^2\right)(\nabla^2\mathbf{f})^T +$$
$$\frac{1}{3!}\nabla^3\mathbf{f}\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right](\nabla\mathbf{f})^T + \cdots.$$

[Julier and Uhlmann, 1997]

UKF intuition why it can perform better

- Assume
  - 1. We represent our distribution over x by a set of sample points.
  - 2. We propagate the points directly through the function f.
- Then:
  - We don't have any errors in f !!
  - The accuracy we obtain can be related to how well the first, second, third, … moments of the samples correspond to the first, second, third, … moments of the true distribution over x.

Self-quiz

- When would the UKF significantly outperform the EKF?

Original unscented transform

- Picks a minimal set of sample points that match 1st, 2nd and 3rd moments of a Gaussian:

$$
\begin{aligned}
\mathcal{X}_0 &= \mathbf{x} & W_0 &= \kappa/(n+\kappa) \\
\mathcal{X}_i &= \bar{\mathbf{x}} + \left(\sqrt{(n+\kappa)\mathbf{P}_{xx}}\right)_i & W_i &= 1/2(n+\kappa) \\
\mathcal{X}_{i+n} &= \mathbf{x} - \left(\sqrt{(n+\kappa)\mathbf{P}_{xx}}\right)_i & W_{i+n} &= 1/2(n+\kappa)
\end{aligned}
$$

- \bar{x} = mean, P_{xx} = covariance, i → i'th row, $x \in \Re^n$
- \kappa : extra degree of freedom to fine-tune the higher order moments of the approximation; when x is Gaussian, n+\kappa = 3 is a suggested heuristic

[Julier and Uhlmann, 1997]

Unscented Kalman filter

- Dynamics update:
  - Can simply use unscented transform and estimate the mean and variance at the next time from the sample points
- Observation update:
  - Use sigmapoints from unscented transform to compute the covariance matrix between x_t and z_t. Then can do the standard update.

Algorithm Unscented_Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

1. $\mathcal{X}_{t-1} = (\mu_{t-1} \quad \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} \quad \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}})$
2. $\bar{\mathcal{X}}_t^* = g(\mu_t, \mathcal{X}_{t-1})$
3. $\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$
4. $\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^\top + R_t$
5. $\bar{\mathcal{X}}_t = \left(\bar{\mu}_t \quad \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t} \quad \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t}\right)$
6. $\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$
7. $\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}$
8. $S_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)\left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^\top + Q_t$
9. $\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t\right)\left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^\top$
10. $K_t = \bar{\Sigma}_t^{x,z} S_t^{-1}$
11. $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$
12. $\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^\top$
13. **return** μ_t, Σ_t

[Table 3.4 in Probabilistic Robotics]

UKF Summary

- Highly efficient: Same complexity as EKF, with a constant factor slower in typical practical applications
- Better linearization than EKF: Accurate in first two terms of Taylor expansion (EKF only first term)
- Derivative-free: No Jacobians needed
- Still not optimal!

Announcements

- Final project: 45% of the grade, 10% presentation, 35% write-up
  - Presentations: in lecture Dec 1 and 3
  - If you have constraints, inform me by email by Wednesday night, we will assign the others at random on Thursday
- PS2: due Friday 23:59pm.
- Tuesday 4-5pm 540 Cory: Hadas Kress-Gazit (Cornell)

High-level tasks to correct low-level robot control

In this talk I will present a formal approach to creating robot controllers that ensure the robot satisfies a given high level task. I will describe a framework in which a user specifies a complex and reactive task in Structured English. This task is then automatically translated, using temporal logic and tools from the formal methods world, into a hybrid controller. This controller is guaranteed to control the robot such that its motion and actions satisfy the intended task, in a variety of different environments.

CS 287: Advanced Robotics
Fall 2009

Lecture 23:

HMMs: Kalman filters, particle filters

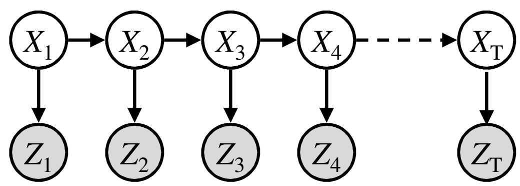
Pieter Abbeel

UC Berkeley EECS

Hidden Markov Models

Joint distribution is assumed to be of the form:

$P(X_1 = x_1)\ P(Z_1 = z_1 \mid X_1 = x_1)\ P(X_2 = x_2 \mid X_1 = x_1)\ P(Z_2 = z_2 \mid X_2 = x_2) \ldots$
$P(X_T = x_T \mid X_{T-1} = x_{T-1})\ P(Z_T = z_T \mid X_T = x_T)$



Filtering in HMM

- Init $P(x_1)$ [e.g., uniformly]
- Observation update for time 0:

  $$P(x_1|z_1) \propto P(z_1|x_1)P(x_1)$$

- For t = 1, 2, …
  - Time update

    $$P(x_{t+1}|z_{1:t}) = \int_{x_t} P(x_{t+1}|x_t)P(x_t|z_{1:t})dx_t$$

  - Observation update

    $$P(x_{t+1}|z_{1:t+1}) \propto P(z_{t+1}|x_{t+1})P(x_{t+1}|z_{1:t})$$

- For discrete state / observation spaces: simply replace integral by summation

Discrete-time Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

with a measurement

$$z_t = C_t x_t + \delta_t$$

5

Kalman Filter Algorithm

Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Correction:

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

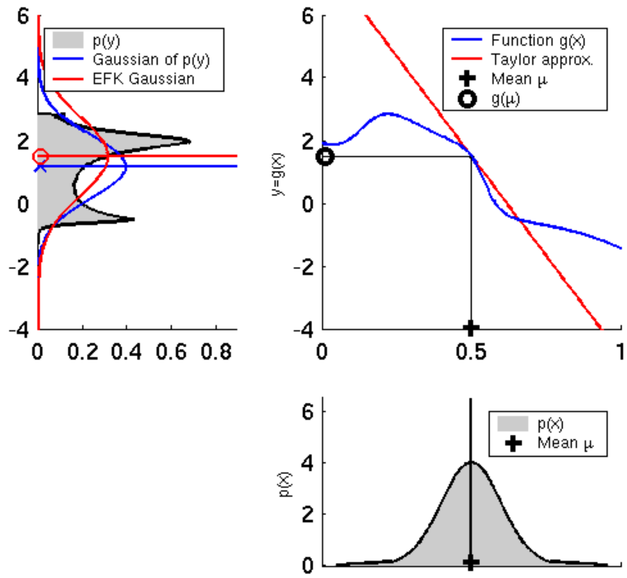
6

Nonlinear systems

$$x_t = g(u_t, x_{t-1}) + noise$$

$$z_t = h(x_t) + noise$$

Extended Kalman filter (EKF)



EKF Linearization: First Order Taylor Series Expansion

- Prediction:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1})$$
$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \ (x_{t-1} - \mu_{t-1})$$

- Correction:

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t} (x_t - \bar{\mu}_t)$$
$$h(x_t) \approx h(\bar{\mu}_t) + H_t \ (x_t - \bar{\mu}_t)$$

EKF Algorithm

Extended_Kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t):

Prediction:

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \quad \longleftarrow \quad \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \quad \longleftarrow \quad \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

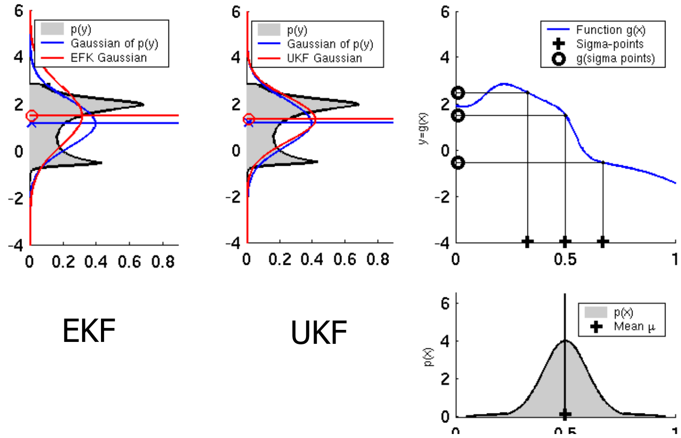
Correction:

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \quad \longleftarrow \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) \quad \longleftarrow \quad \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \quad \longleftarrow \quad \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

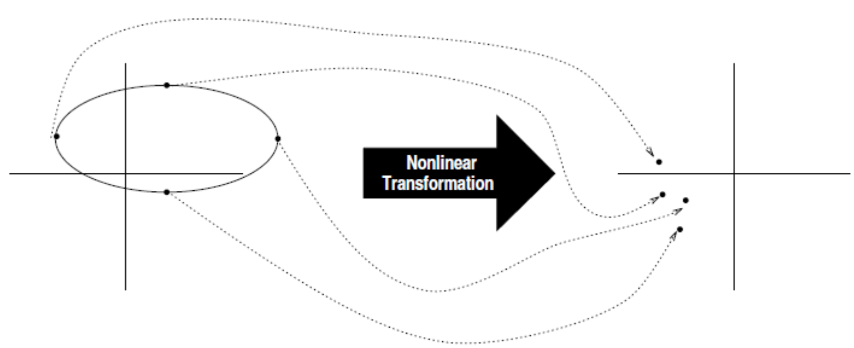
$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \qquad G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

Linearization via Unscented Transform



11

UKF Sigma-Point Estimate (4)



UKF intuition why it can perform better

- Assume we know the distribution over X and it has a mean \bar{x}
- Y = f(X)

$$\mathbf{f}[\mathbf{x}] = \mathbf{f}[\bar{\mathbf{x}} + \boldsymbol{\delta}\mathbf{x}]$$
$$= \mathbf{f}[\bar{\mathbf{x}}] + \nabla\mathbf{f}\boldsymbol{\delta}\mathbf{x} + \frac{1}{2}\nabla^2\mathbf{f}\boldsymbol{\delta}\mathbf{x}^2 + \frac{1}{3!}\nabla^3\mathbf{f}\boldsymbol{\delta}\mathbf{x}^3 + \frac{1}{4!}\nabla^4\mathbf{f}\boldsymbol{\delta}\mathbf{x}^4 + \cdots$$

$$\bar{\mathbf{y}} = \mathbf{f}[\bar{\mathbf{x}}] + \frac{1}{2}\nabla^2\mathbf{f}\,\mathbf{P}_{xx} + \frac{1}{2}\nabla^4\mathbf{f}\,\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right] + \cdots$$

$$\mathbf{P}_{yy} = \nabla\mathbf{f}\,\mathbf{P}_{xx}(\nabla\mathbf{f})^T + \frac{1}{2\times 4!}\nabla^2\mathbf{f}\left(\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right] - \mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^2\mathbf{P}_{yy}\right] - \mathrm{E}\left[\mathbf{P}_{yy}\boldsymbol{\delta}\mathbf{x}^2\right] + \mathbf{P}_{yy}^2\right)(\nabla^2\mathbf{f})^T +$$
$$\frac{1}{3!}\nabla^3\mathbf{f}\mathrm{E}\left[\boldsymbol{\delta}\mathbf{x}^4\right](\nabla\mathbf{f})^T + \cdots.$$

[Julier and Uhlmann, 1997]

UKF intuition why it can perform better

- Assume
  - 1. We represent our distribution over x by a set of sample points.
  - 2. We propagate the points directly through the function f.
- Then:
  - We don’t have any errors in f !!
  - The accuracy we obtain can be related to how well the first, second, third, … moments of the samples correspond to the first, second, third, … moments of the true distribution over x.

Self-quiz

- When would the UKF significantly outperform the EKF?

Original unscented transform

- Picks a minimal set of sample points that match 1st, 2nd and 3rd moments of a Gaussian:

$$
\begin{aligned}
\mathcal{X}_0 &= \bar{\mathbf{x}} & W_0 &= \kappa/(n+\kappa) \\
\mathcal{X}_i &= \bar{\mathbf{x}} + \left(\sqrt{(n+\kappa)\mathbf{P}_{xx}}\right)_i & W_i &= 1/2(n+\kappa) \\
\mathcal{X}_{i+n} &= \bar{\mathbf{x}} - \left(\sqrt{(n+\kappa)\mathbf{P}_{xx}}\right)_i & W_{i+n} &= 1/2(n+\kappa)
\end{aligned}
$$

- \bar{x} = mean, P_{xx} = covariance, i → i'th row, $x \in \Re^n$
- \kappa : extra degree of freedom to fine-tune the higher order moments of the approximation; when x is Gaussian, n+\kappa = 3 is a suggested heuristic

[Julier and Uhlmann, 1997]

Unscented Kalman filter

- Dynamics update:
  - Can simply use unscented transform and estimate the mean and variance at the next time from the sample points
- Observation update:
  - Use sigmapoints from unscented transform to compute the covariance matrix between x_t and z_t. Then can do the standard update.

Algorithm Unscented_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

1. $\mathcal{X}_{t-1} = \left(\mu_{t-1} \quad \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} \quad \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}}\right)$
2. $\bar{\mathcal{X}}_t^* = g(\mu_t, \mathcal{X}_{t-1})$
3. $\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$
4. $\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^\top + R_t$
5. $\bar{\mathcal{X}}_t = \left(\bar{\mu}_t \quad \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t} \quad \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t}\right)$
6. $\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$
7. $\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}$
8. $S_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right) \left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^\top + Q_t$
9. $\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t\right) \left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^\top$
10. $K_t = \bar{\Sigma}_t^{x,z} S_t^{-1}$
11. $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$
12. $\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^\top$
13. **return** μ_t, Σ_t

[Table 3.4 in Probabilistic Robotics]

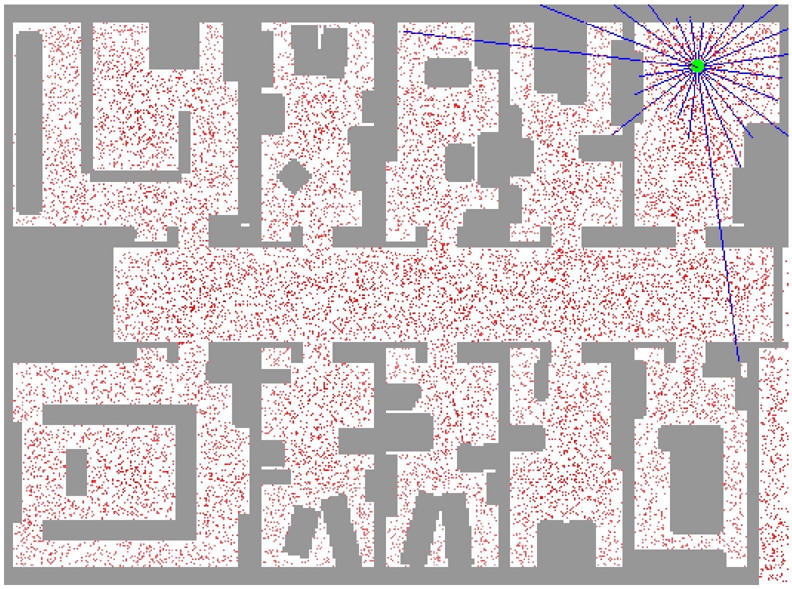
UKF Summary

- Highly efficient: Same complexity as EKF, with a constant factor slower in typical practical applications
- Better linearization than EKF: Accurate in first two terms of Taylor expansion (EKF only first term) + capturing more aspects of the higher order terms
- Derivative-free: No Jacobians needed
- Still not optimal!

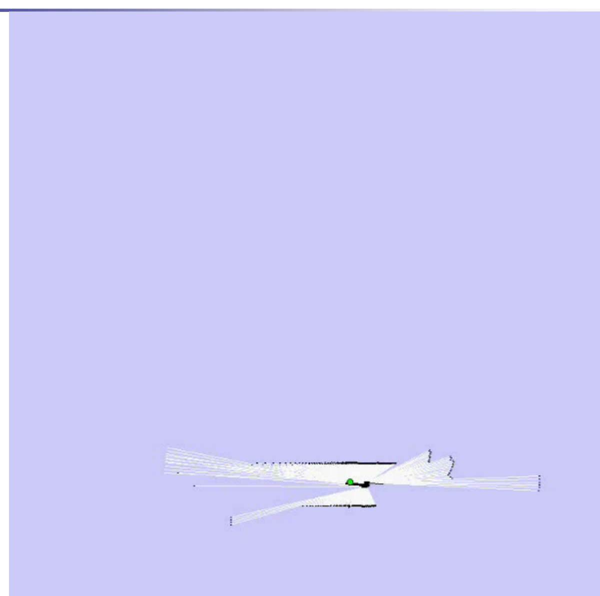
Particle filters motivation

- Particle filters are a way to **efficiently** represent **non-Gaussian distribution**
- Basic principle
  - Set of state hypotheses ("particles")
  - Survival-of-the-fittest

Sample-based Localization (sonar)

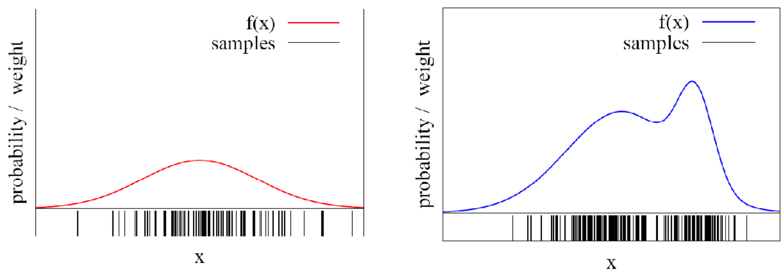


FastSLAM [particle filter + Rao-Blackwellization + occupancy grid mapping + scan matching based odometry]



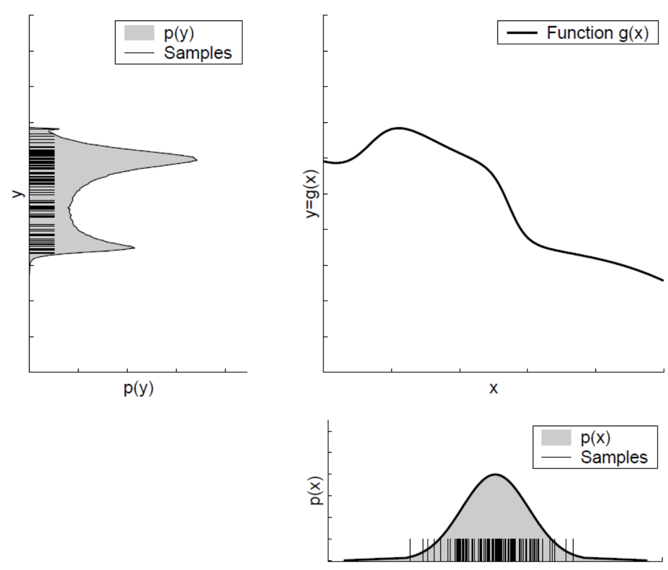
Sample-based Mathematical Description of Probability Distributions

- Particle sets can be used to approximate functions



- The more particles fall into an interval, the higher the probability of that interval
- If a continuous density needs to be extracted → can place a narrow Gaussian at the location of each particle
- How to efficiently draw samples in an HMM?

Dynamics update with sample representation of distribution: sample from $P(x_{t+1} \mid x_t)$



Observation update

- Goal: go from a sample-based representation of

  $P(x_{t+1} \mid z_1, \ldots, z_t)$

  to a sample-based representation of

  $P(x_{t+1} \mid z_1, \ldots, z_t, z_{t+1}) =$

  $C * P(x_{t+1} \mid z_1, \ldots, z_t) * P(z_{t+1} \mid x_{t+1})$

Importance sampling

- Interested in estimating:

$$
\begin{aligned}
\mathrm{E}_{X\sim P}[f(X)] &= \int_x f(x)P(x)dx \\
&= \int_x f(x)P(x)\frac{Q(x)}{Q(x)}dx \quad \text{if} Q(x)=0 \Rightarrow P(x)=0 \\
&= \int_x f(x)\frac{P(x)}{Q(x)}Q(x)dx \\
&= \mathrm{E}_{X\sim Q}[\frac{P(X)}{Q(X)}f(X)] \\
&\approx \frac{1}{m}\sum_{i=1}^{m}\frac{P(x^{(i)})}{Q(x^{(i)})}f(x^{(i)}) \quad \text{with } x^{(i)} \sim Q
\end{aligned}
$$

- ***Hence we could sample from an alternative distribution Q and simply re-weight the samples == Importance Sampling***

Sequential Importance Sampling (SIS) Particle Filter

- Sample $x^{(1)}_1, x^{(2)}_1, \ldots, x^{(N)}_1$ from $P(X_1)$
- Set $w^{(i)}_1 = 1$ for all $i=1,\ldots,N$
- For $t=1, 2, \ldots$
  - Dynamics update:
    - For $i=1, 2, \ldots, N$
      - Sample $x^{(i)}_t$ from $P(X_{t+1} \mid X_t = x^{(i)}_t)$
  - Observation update:
    - For $i=1, 2, \ldots, N$
      - $w^{(i)}_{t+1} = w^{(i)}_t * P(z_{t+1} \mid X_{t+1} = x^{(i)}_{t+1})$
- At any time t, the distribution is represented by the weighted set of samples

  $\{ \langle x^{(i)}_t, w^{(i)}_t \rangle ; i=1,\ldots,N\}$

SIS particle filter major issue

- The resulting samples are only weighted by the evidence
- The samples themselves are never affected by the evidence

→ Fails to concentrate particles/computation in the high probability areas of the distribution $P(x_t \mid z_1, \ldots, z_t)$

Resampling

- At any time t, the distribution is represented by the weighted set of samples

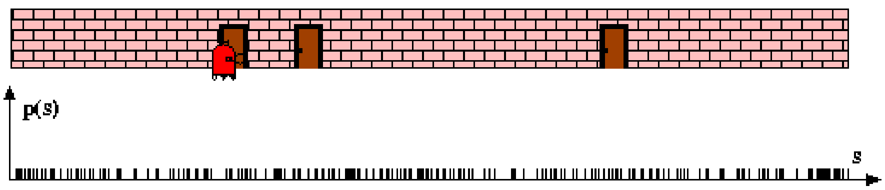
  { <$x^{(i)}_t$, $w^{(i)}_t$> ; i=1,…,N}

- → Sample N times from the set of particles
- → The probability of drawing each particle is given by its importance weight

→ More particles/computation focused on the parts of the state space with high probability mass

1. Algorithm **particle_filter**(S_{t-1}, u_{t-1} z_t):
2. $S_t = \emptyset, \quad \eta = 0$
3. **For** $i = 1 \ldots n$ — ***Generate new samples***
4. Sample index $j(i)$ from the discrete distribution given by w_{t-1}
5. Sample x_t^i from $p(x_t \mid x_{t-1}, u_{t-1})$ using $x_{t-1}^{j(i)}$ and u_{t-1}
6. $w_t^i = p(z_t \mid x_t^i)$ — ***Compute importance weight***
7. $\eta = \eta + w_t^i$ — ***Update normalization factor***
8. $S_t = S_t \cup \{< x_t^i, w_t^i >\}$ — ***Insert***
9. **For** $i = 1 \ldots n$
10. $w_t^i = w_t^i / \eta$ — ***Normalize weights***

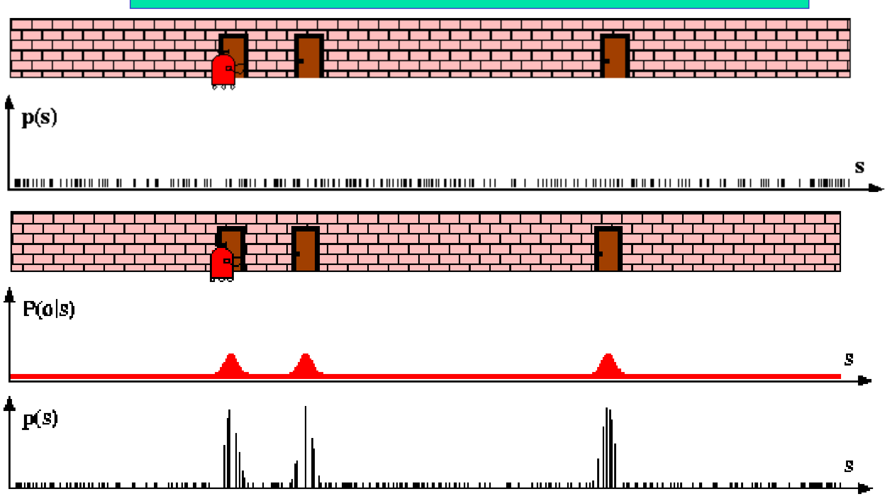
Particle Filters



Sensor Information: Importance Sampling

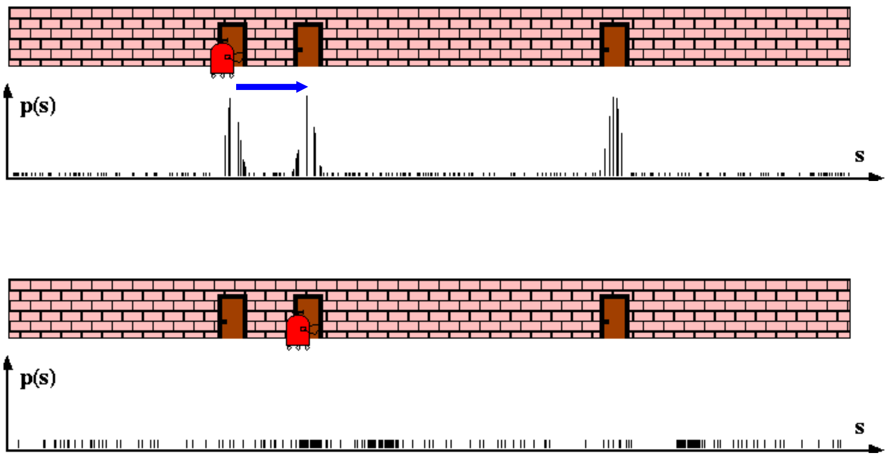
$$Bel(x) \leftarrow \alpha\, p(z \mid x)\, Bel^{-}(x)$$

$$w \leftarrow \frac{\alpha\, p(z \mid x)\, Bel^{-}(x)}{Bel^{-}(x)} = \alpha\, p(z \mid x)$$



Robot Motion

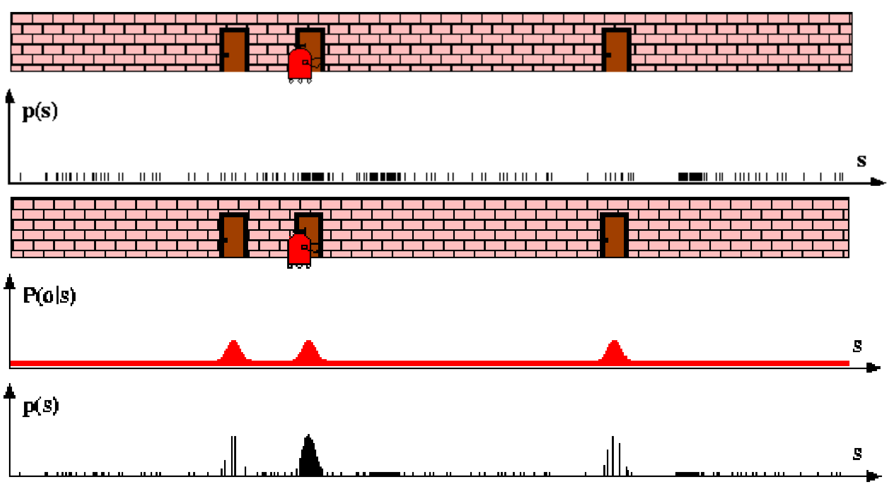
$$Bel^-(x) \leftarrow \int p(x \mid u, x')\, Bel(x')\, \mathrm{d}x'$$



Sensor Information: Importance Sampling

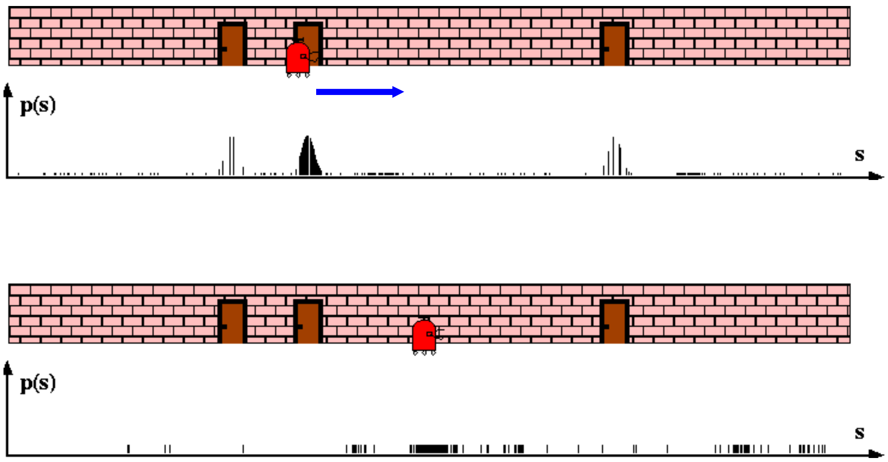
$$Bel(x) \leftarrow \alpha\, p(z \mid x)\, Bel^-(x)$$

$$w \leftarrow \frac{\alpha\, p(z \mid x)\, Bel^-(x)}{Bel^-(x)} = \alpha\, p(z \mid x)$$



Robot Motion

$$Bel^{-}(x) \leftarrow \int p(x \mid u, x')\, Bel(x')\, \mathrm{d}x'$$



Resampling issue

- Loss of samples …

Low Variance Resampling

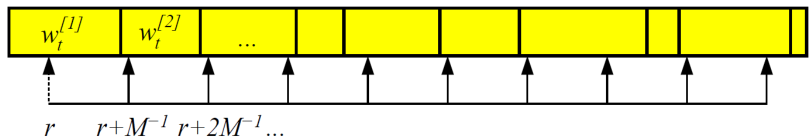


Figure 4.7 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

Low Variance Resampling

- Advantages:
  - More systematic coverage of space of samples
  - If all samples have same importance weight, no samples are lost
  - Lower computational complexity

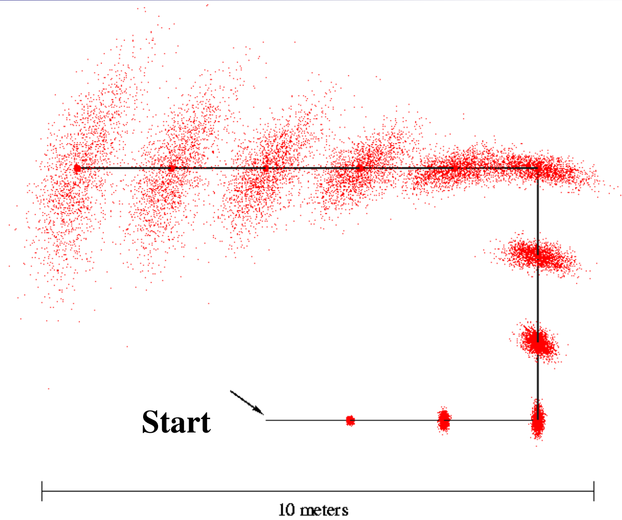
Regularization

- If no dynamics noise → all particles will start to coincide
- → regularization: resample from a (narrow) Gaussian around the particle

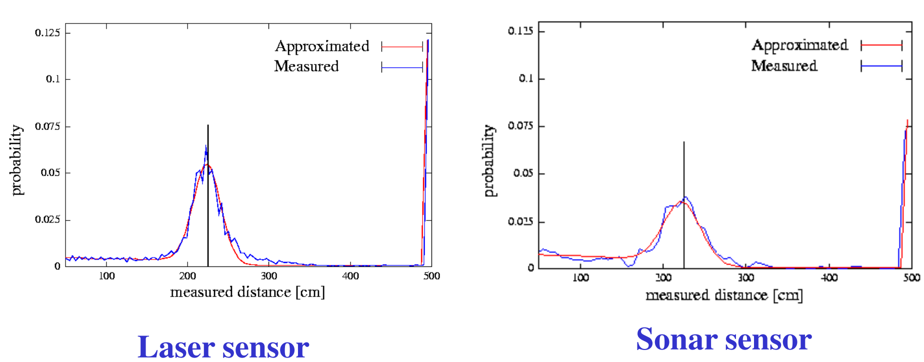
Mobile Robot Localization

- Each particle is a potential pose of the robot
- Proposal distribution is the motion model of the robot (prediction step)
- The observation model is used to compute the importance weight (correction step)

Motion Model Reminder

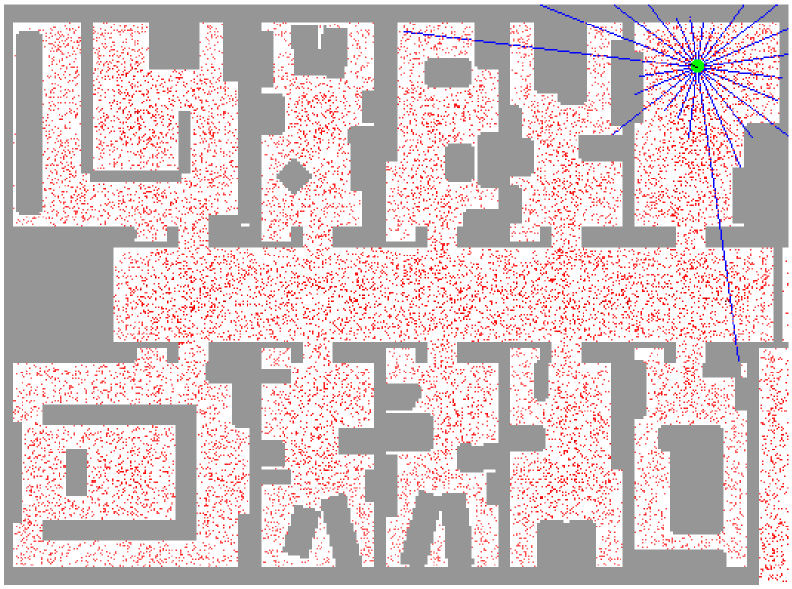


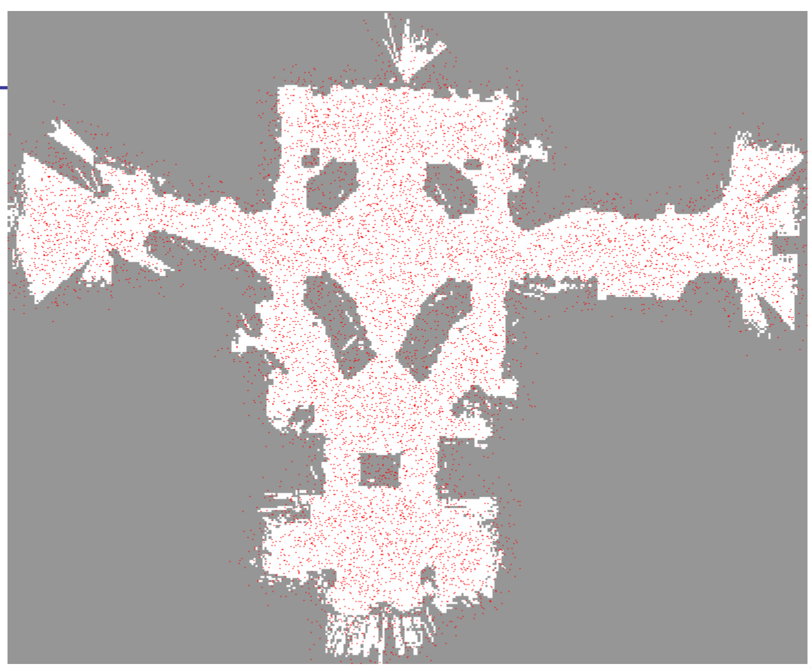
Proximity Sensor Model Reminder



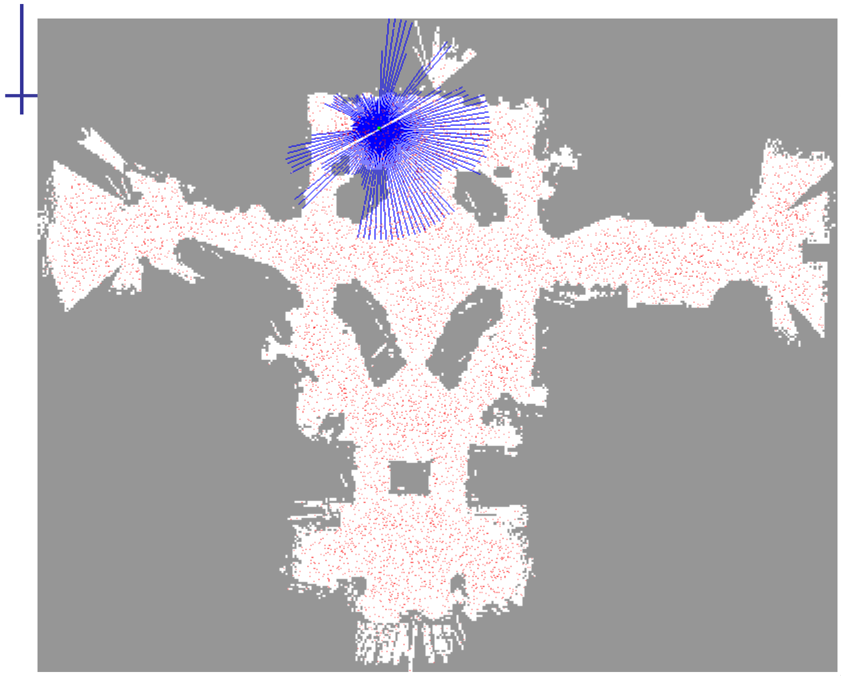
Note: sensor model is not Gaussian at all!

Sample-based Localization (sonar)

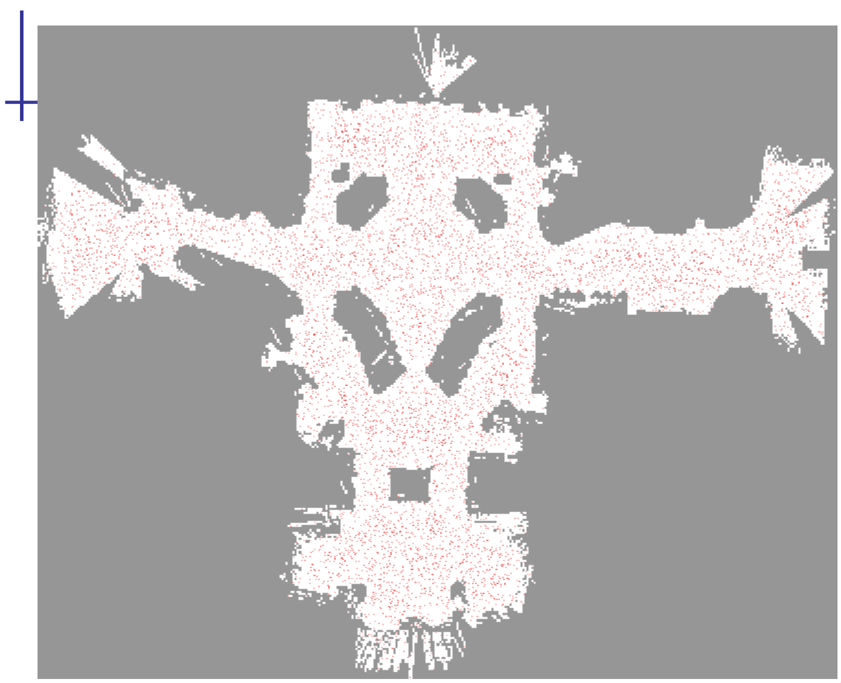




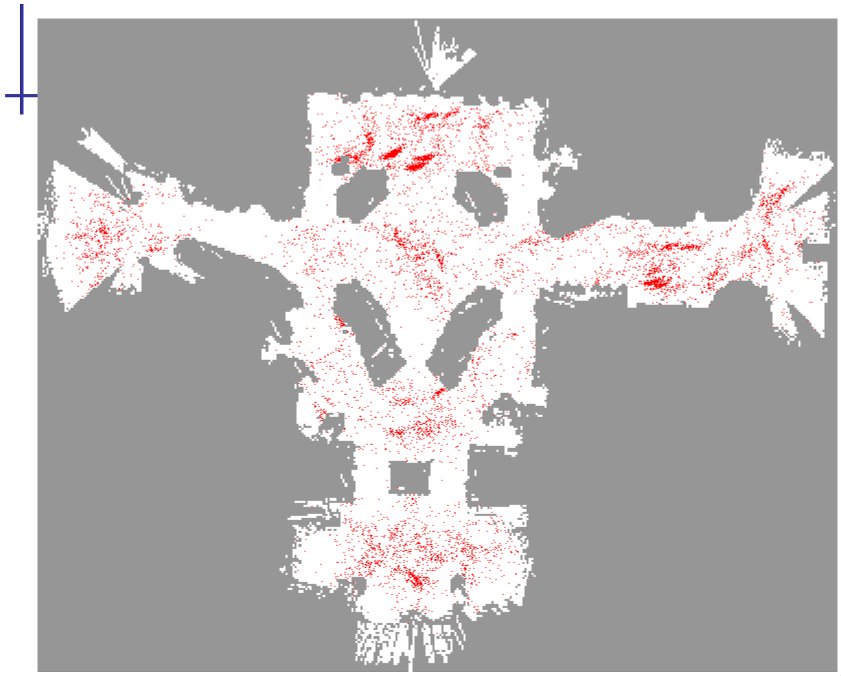
55



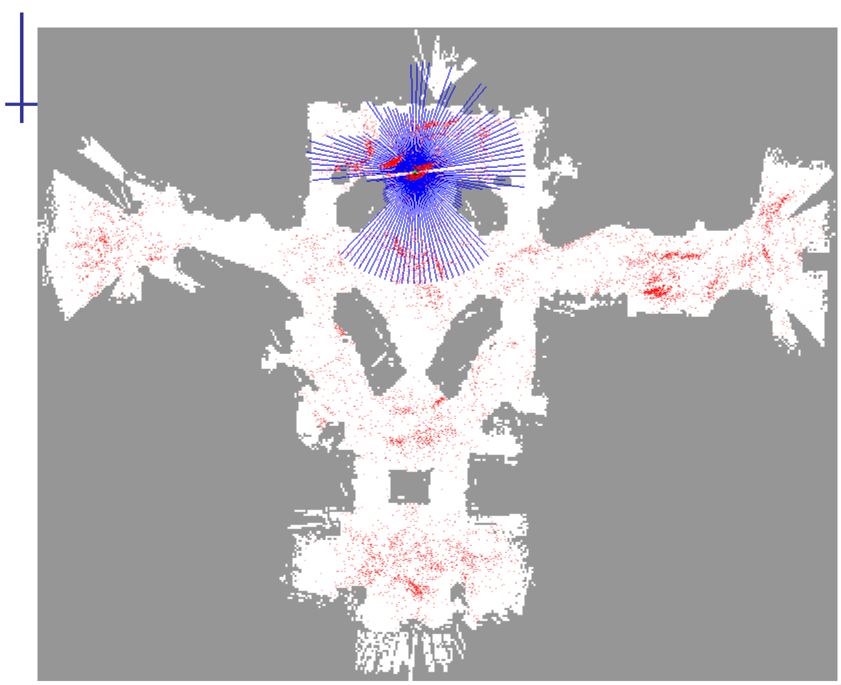
56



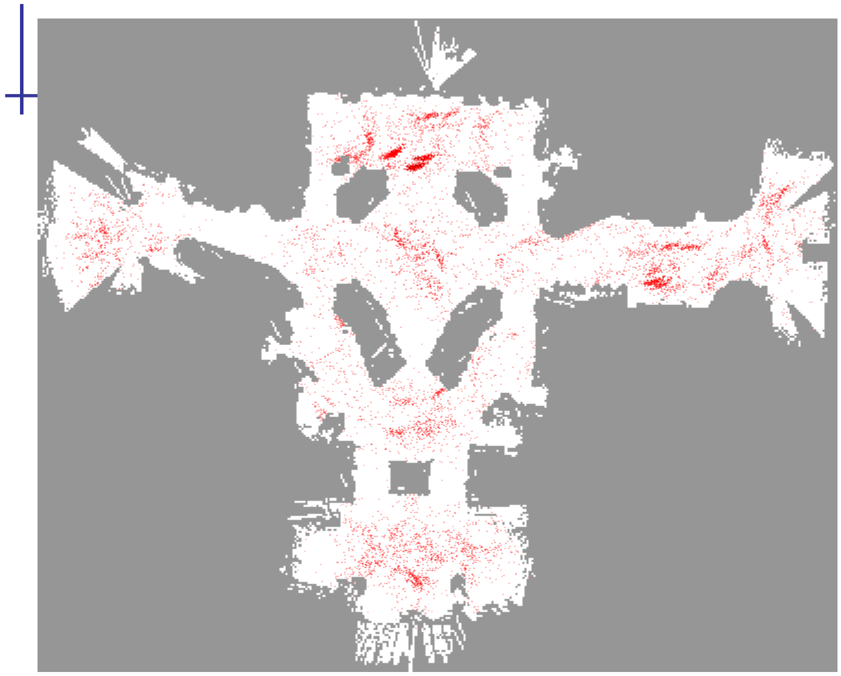
57



58



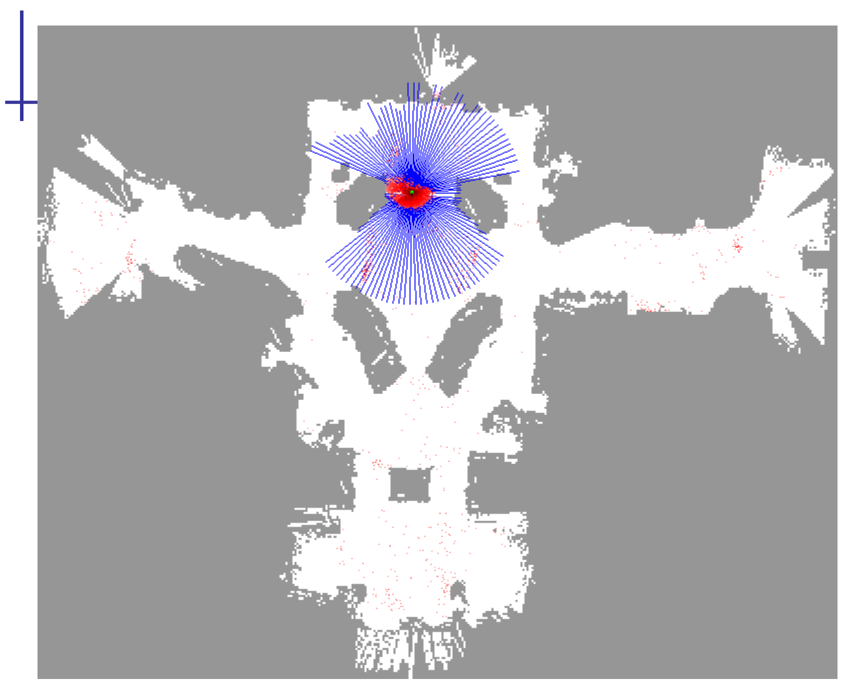
59



60

61

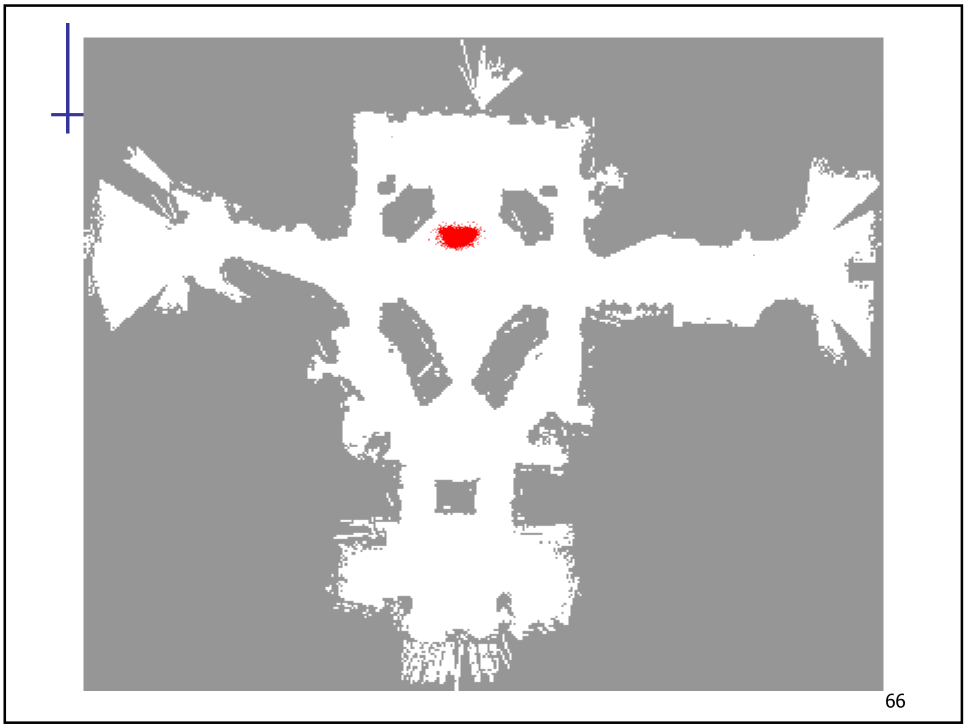
62

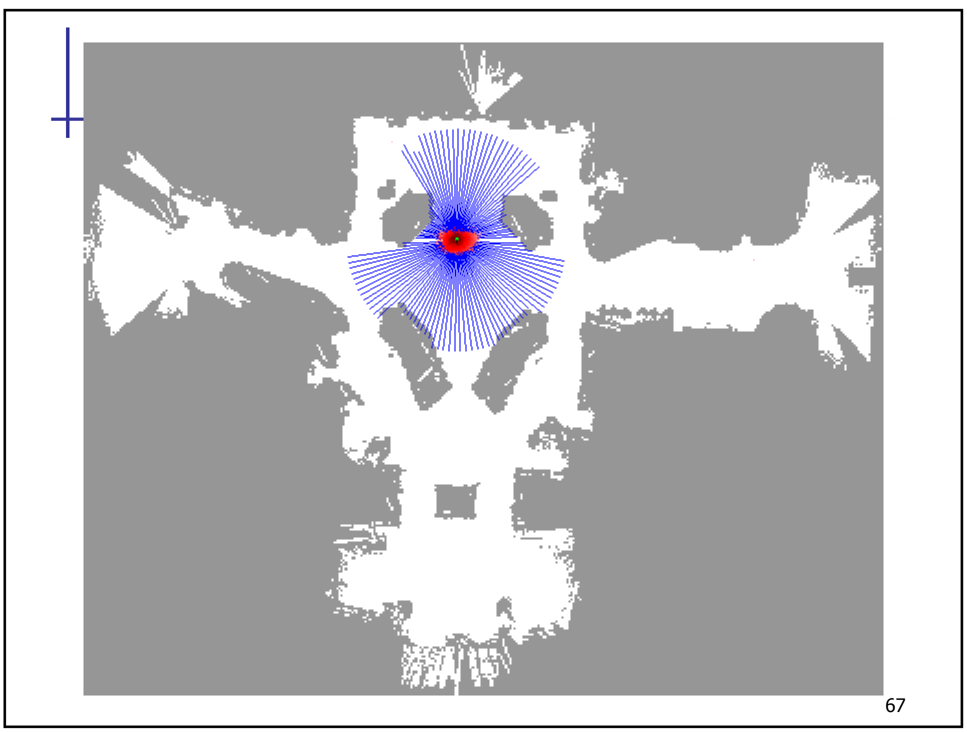


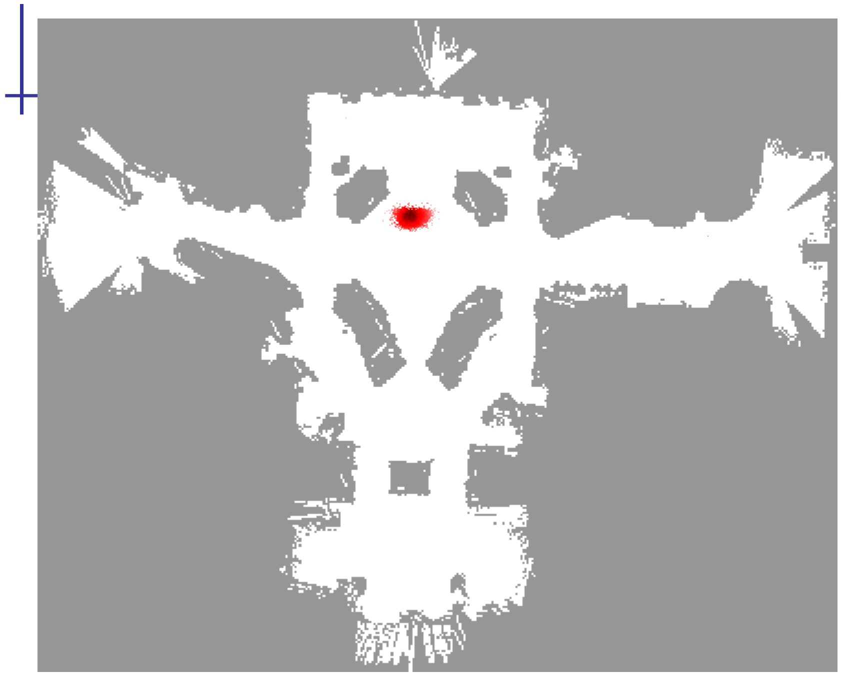
63

64

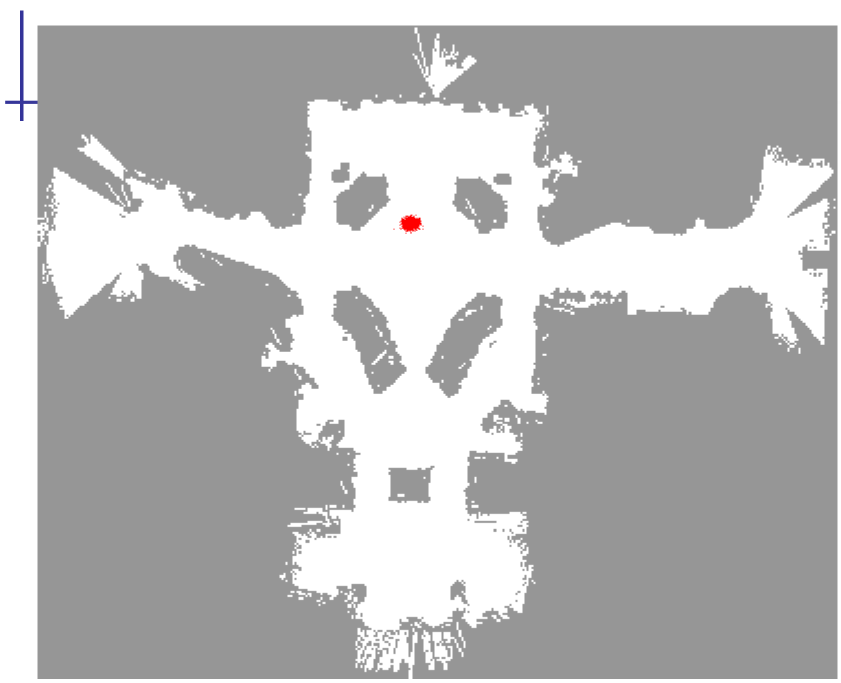
65


66

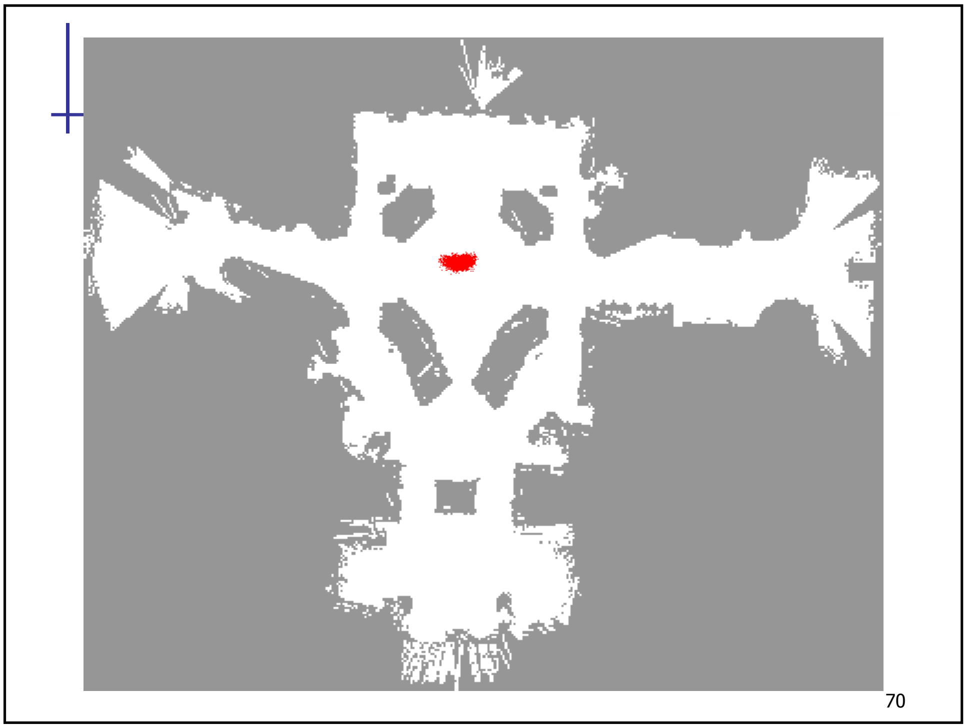

67

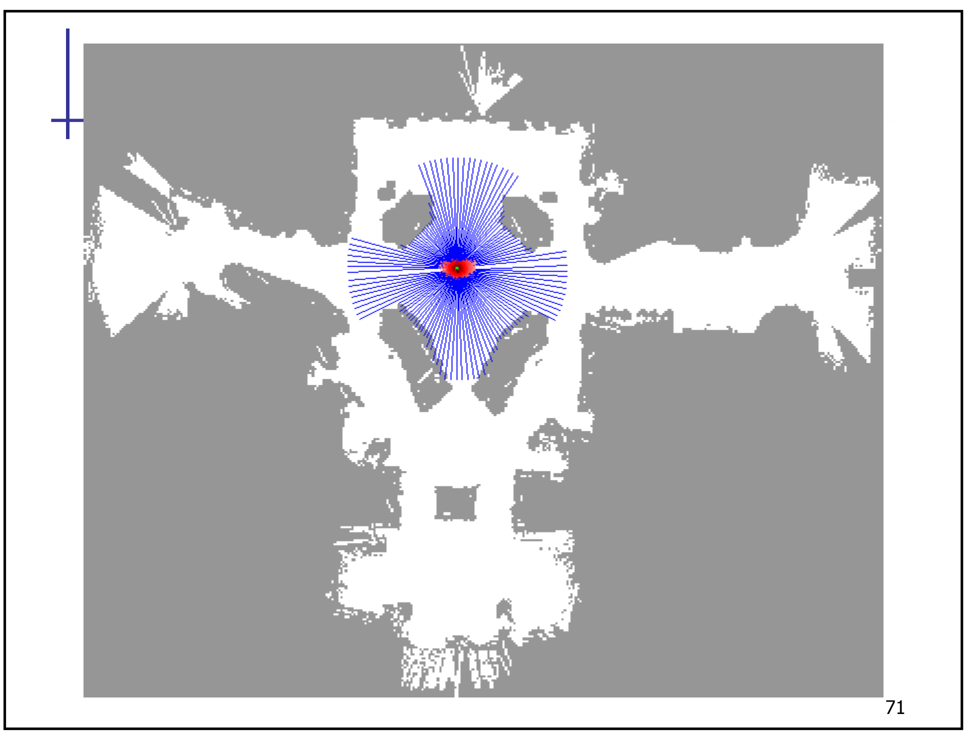


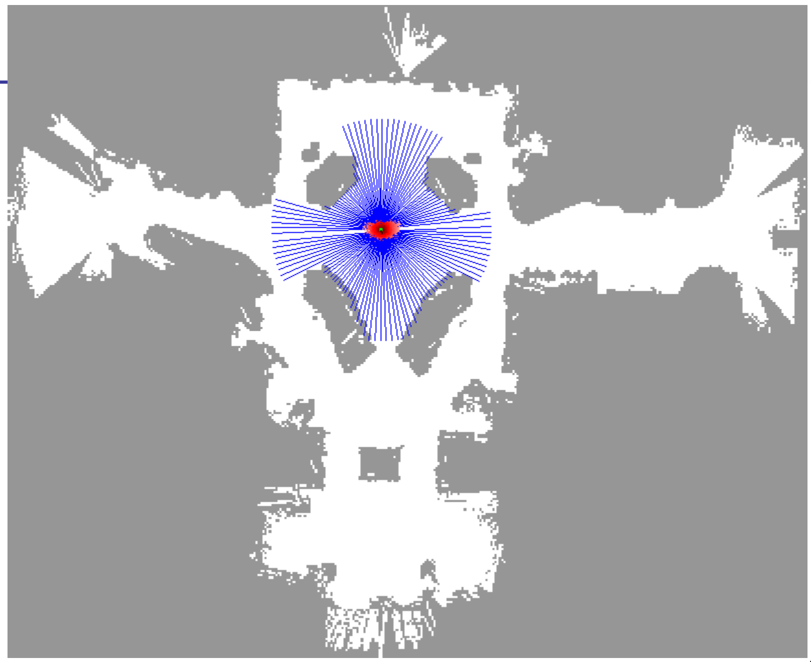
68



69







72

Summary – Particle Filters

- Particle filters are an implementation of recursive Bayesian filtering
- They represent the posterior by a set of weighted samples
- They can model non-Gaussian distributions
- Proposal to draw new samples
- Weight to account for the differences between the proposal and the target
- Monte Carlo filter, Survival of the fittest, Condensation, Bootstrap filter

77

Summary – PF Localization

- In the context of localization, the particles are propagated according to the motion model.
- They are then weighted according to the likelihood of the observations.
- In a re-sampling step, new particles are drawn with a probability proportional to the likelihood of the observation.

Announcements

- PS2: due Friday 23:59pm.
- Final project: 45% of the grade, 10% presentation, 35% write-up
  - Presentations: in lecture Dec 1 and 3 --- schedule:

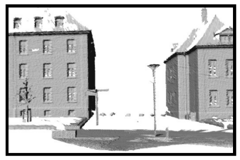
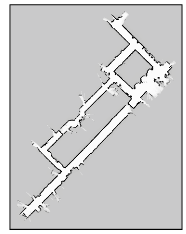
CS 287: Advanced Robotics
Fall 2009

Lecture 24:
SLAM

Pieter Abbeel
UC Berkeley EECS

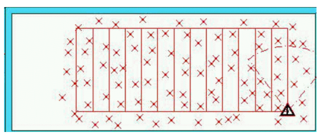
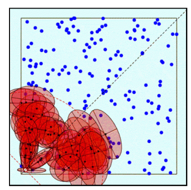
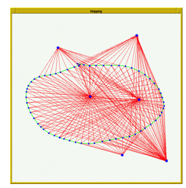
Types of SLAM-Problems

- Grid maps or scans

[Lu & Milios, 97; Gutmann, 98: Thrun 98; Burgard, 99; Konolige & Gutmann, 00; Thrun, 00; Arras, 99; Haehnel, 01;…]

- Landmark-based

[Leonard et al., 98; Castelanos et al., 99: Dissanayake et al., 2001; Montemerlo et al., 2002;…

Recap Landmark based SLAM

- State variables:
  - Robot pose
  - Coordinates of each of the landmarks
- Robot dynamics model: $P(x_{t+1} \mid x_t, u_t)$
- Sensor model: $P(z_{t+1} \mid x_t, m)$
  - Probability of landmark observations given the state
- Can run EKF, SEIF, various other approaches

→ Result: path of robot, location of landmarks

KF-type approaches are a good fit b/c they can keep track of correlations between landmarks

Note: Could then use path of robot + sensor log and build a map assuming known robot poses

Grid-based SLAM

- Can we solve the SLAM problem if no pre-defined landmarks are available?
- As with landmarks, the map depends on the poses of the robot during data acquisition
- If the poses are known, grid-based mapping is easy ("mapping with known poses")

6

Occupancy Grid Maps

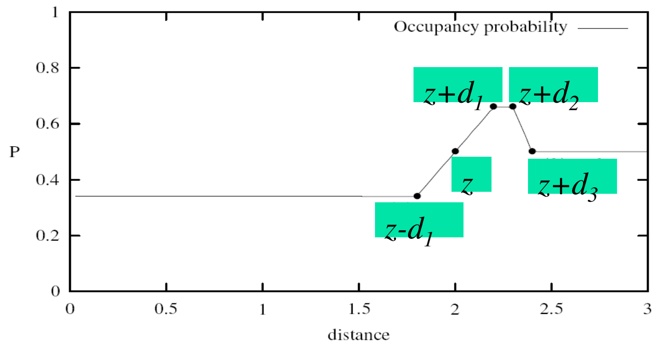
- Introduced by Moravec and Elfes in 1985
- Represent environment by a grid.
- Estimate the probability that a location is occupied by an obstacle.
- Key assumptions
  - Occupancy of individual cells (m[xy]) is independent

$$Bel(m_t) = P(m_t \mid u_1, z_2 \ldots, u_{t-1}, z_t) \\ = \prod_{x,y} Bel(m_t^{[xy]})$$

  - Robot positions are known!

7

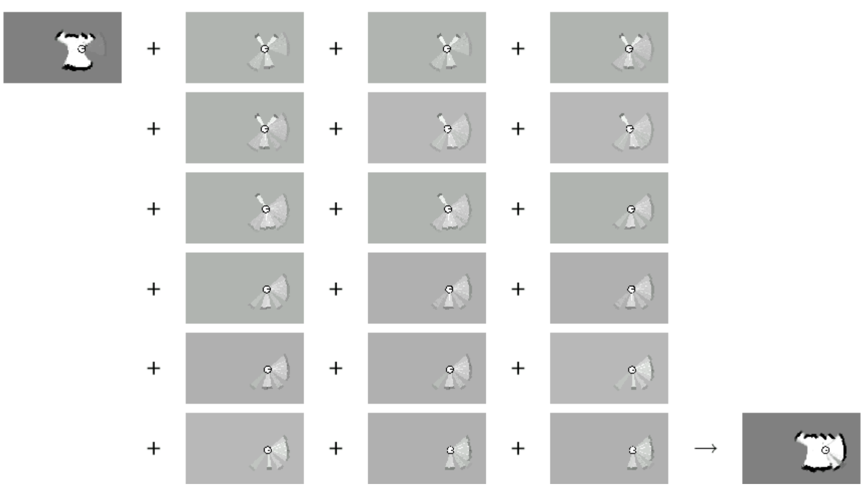
Occupancy Value Depending on the Measured Distance



$$\log \frac{P(m^{[xy]}=1)}{P(m^{[xy]}=0)} \leftarrow \log \frac{P(m^{[xy]}=1)}{P(m^{[xy]}=0)} + \log \frac{P(m^{[xy]}=1|z_t)}{P(m^{[xy]}=0|z_t)}$$

8

Incremental Updating of Occupancy Grids (Example)



9

Alternative: Simple Counting

- For every cell count
  - hits(x,y): number of cases where a beam ended at <x,y>
  - misses(x,y): number of cases where a beam passed through <x,y>

$$Bel(m^{[xy]}) = \frac{\text{hits}(x,y)}{\text{hits}(x,y) + \text{misses}(x,y)}$$

- Value of interest: $P(reflects(x,y))$

10

Difference between Occupancy Grid Maps and Counting

- The counting model determines how often a cell reflects a beam.
- The occupancy model represents whether or not a cell is occupied by an object.
- Although a cell might be occupied by an object, the reflection probability of this object might be very small.

12

Example Occupancy Map

13

Example Reflection Map

14

Example

- Out of 1000 beams only 60% are reflected from a cell and 40% intercept it without ending in it.
- Accordingly, the reflection probability will be 0.6.
- Suppose *p(occ | z)* = 0.55 when a beam ends in a cell and *p(occ | z)* = 0.45 when a cell is intercepted by a beam that does not end in it.
- Accordingly, after n measurements we will have

$$\left(\frac{0.55}{0.45}\right)^{n*0.6} * \left(\frac{0.45}{0.55}\right)^{n*0.4} = \left(\frac{11}{9}\right)^{n*0.6} * \left(\frac{11}{9}\right)^{-n*0.4} = \left(\frac{11}{9}\right)^{n*0.2}$$

- Whereas the reflection map yields a value of 0.6, the occupancy grid value converges to 1.

15

Mapping using Raw Odometry

16

Distribution over robot poses and maps

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1})$$

- Standard particle filter represents the distribution by a set of samples

$$\{< (x_{1:t}^{(i)}, m^{(i)}), w^{(i)} >\}$$

Rao-Blackwellization

poses map observations & movements

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}) =$$
$$p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(m \mid x_{1:t}, z_{1:t})$$

Factorization first introduced by Murphy in 1999

Rao-Blackwellization

poses map observations & movements

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}) =$$
$$p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(m \mid x_{1:t}, z_{1:t})$$

SLAM posterior

Robot path posterior

Mapping with known poses

Factorization first introduced by Murphy in 1999

19

Rao-Blackwellization

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}) =$$
$$p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(m \mid x_{1:t}, z_{1:t})$$

This is localization, use MCL

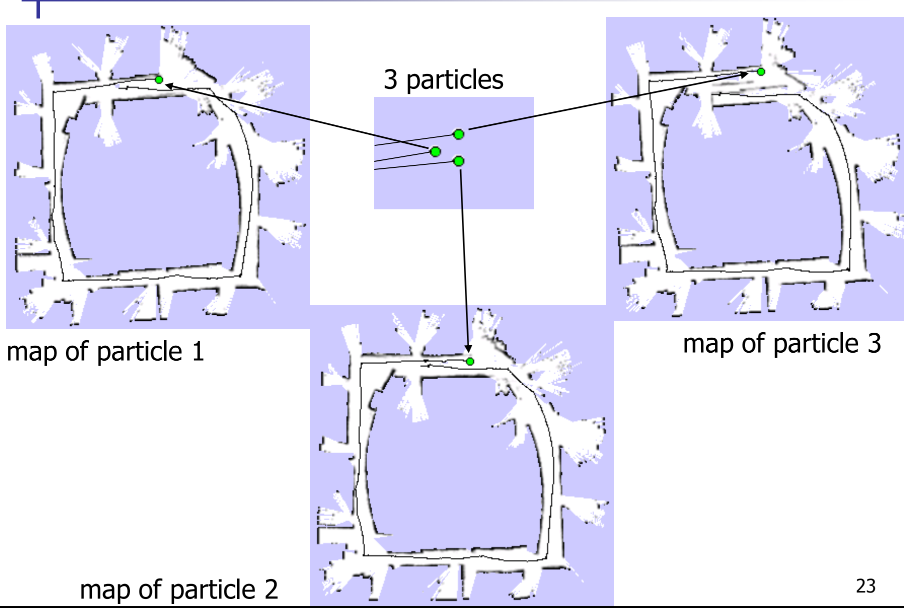
Use the pose estimate from the MCL part and apply mapping with known poses

20

Rao-Blackwellized Mapping

- Each particle represents a possible trajectory of the robot
- Each particle
  - maintains its own map and
  - updates it upon "mapping with known poses"
- Each particle survives with a probability proportional to the likelihood of the observations relative to its own map

Particle Filter Example



Problem

- Each map is quite big in case of grid maps
- Since each particle maintains its own map
- Therefore, one needs to keep the number of particles small

- **Solution**:
  Compute better proposal distributions!
- **Idea**:
  Improve the pose estimate **before** applying the particle filter

Pose Correction Using Scan Matching

Maximize the likelihood of the i-th pose and map relative to the (i-1)-th pose and map

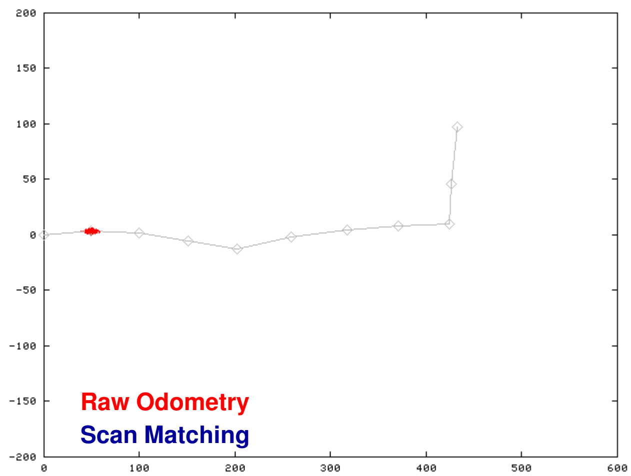
$$\hat{x}_t = \underset{x_t}{\mathrm{argmax}} \left\{ p(z_t \mid x_t, \hat{m}_{t-1}) \cdot p(x_t \mid u_{t-1}, \hat{x}_{t-1}) \right\}$$

current measurement

robot motion

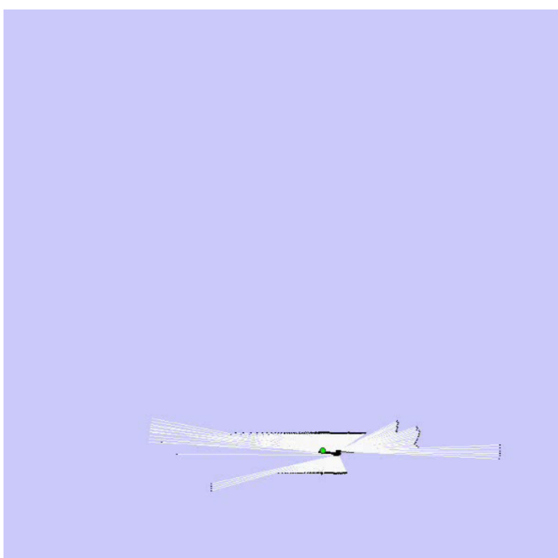
map constructed so far

Motion Model for Scan Matching



26

FastSLAM with Scan-Matching



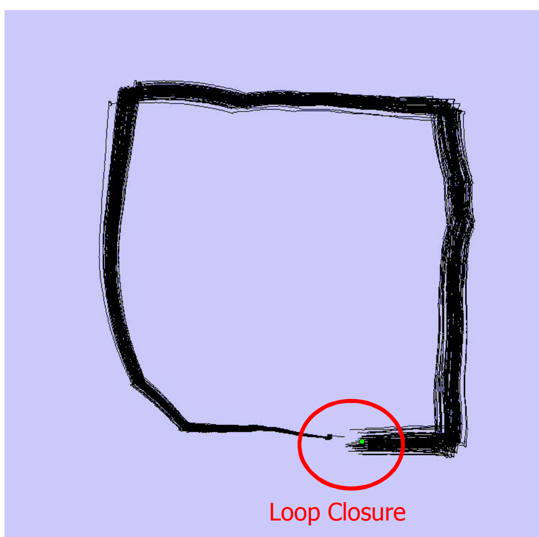
Map: Intel Research Lab Seattle

30

Map of the Intel Lab

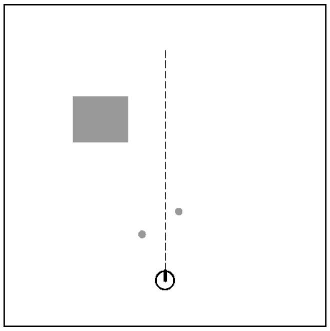
- 15 particles
- four times faster than real-time P4, 2.8GHz
- 5cm resolution during scan matching
- 1cm resolution in final map

FastSLAM with Scan-Matching

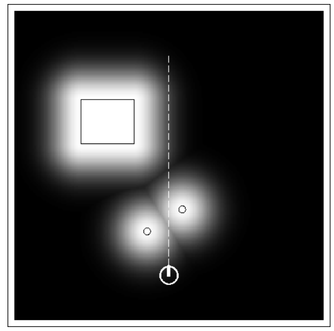


32

Scan matching: likelihood field



Map m



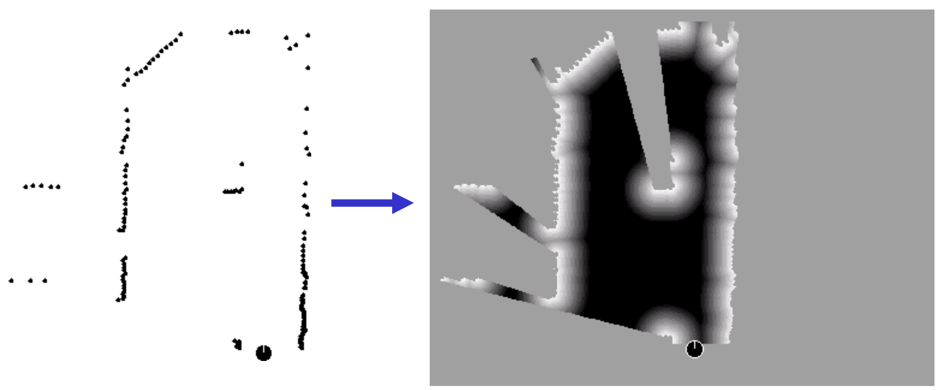
Likelihood field

=map convolved with a Gaussian

33

Scan Matching

- Extract likelihood field from scan and use it to match different scan.



34

FastSLAM recap

- Rao-Blackwellized representation:
  - Particle instantiates entire path of robot
  - Map associated with each path
- Scan matching: improves proposal distribution
- Original FastSLAM:
  - Map associated with each particle was a Gaussian distribution over landmark positions
- DP-SLAM: extension which has very efficient map management, enabling having a relatively large number of particles [Eliazar and Parr, 2002/2005]

SLAM thus far

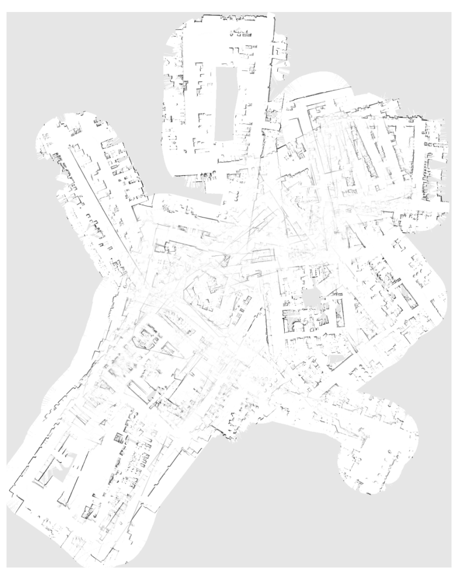
- Landmark based vs. occupancy grid
- Probability distribution representation:
  - EKF vs. particle filter vs. Rao-Blackwellized particle filter
- EKF, SEIF, FastSLAM are all “online”
- Currently popular 4^{th} alternative: GraphSLAM

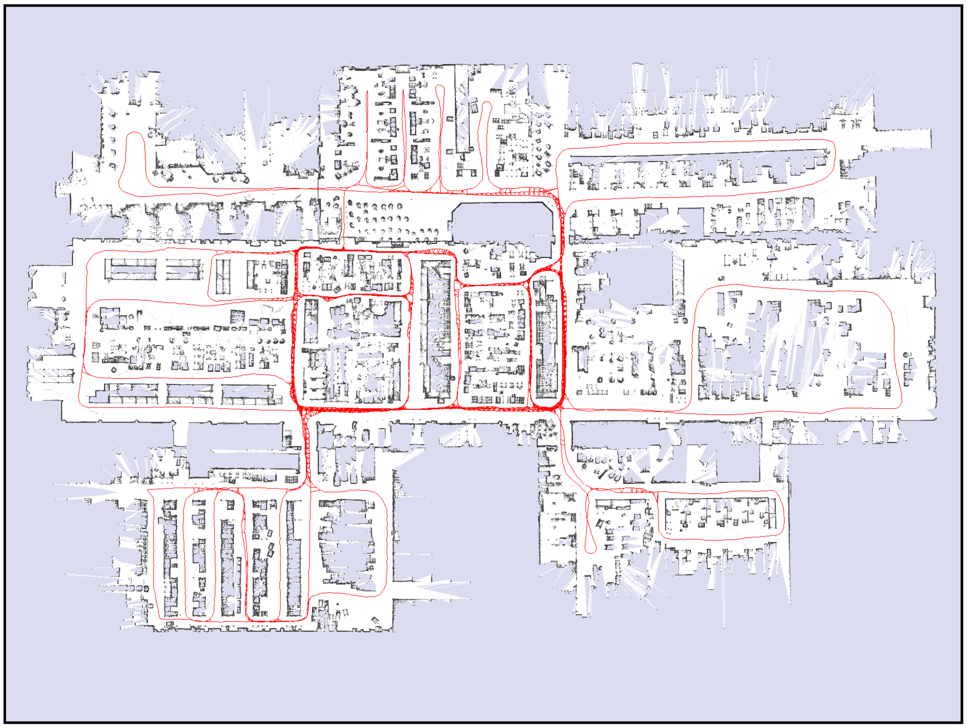
Graph-based Formulation

- Use a **graph** to represent the problem
- **Every node** in the graph **corresponds to a pose** of the robot during mapping
- **Every edge** between two nodes **corresponds to the spatial constraints** between them
- **Goal**:
  Find a configuration of the nodes that **minimize the error** introduced by the constraints

$$\begin{aligned} J_{\text{GraphSLAM}} &= x_0^\top \Omega_0 x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^\top R_t^{-1} (x_t - g(u_t, x_{t-1})) \\ &+ \sum_t \sum_i (z_t^i - h(x_t, m, c_t^i))^\top Q_t^{-1} (z_t^i - h(x_t, m, c_t^i)) \end{aligned}$$

The KUKA Production Site

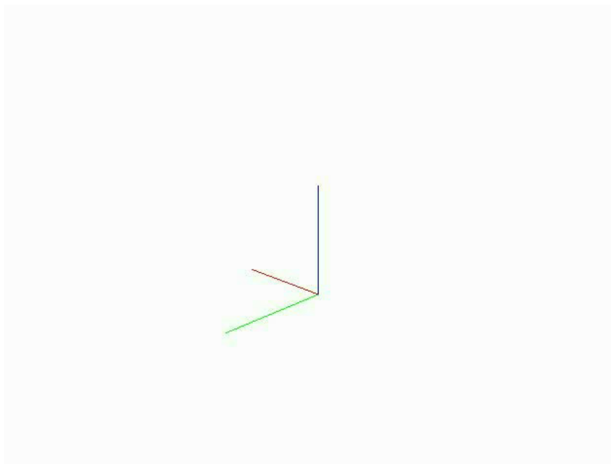




The KUKA Production Site

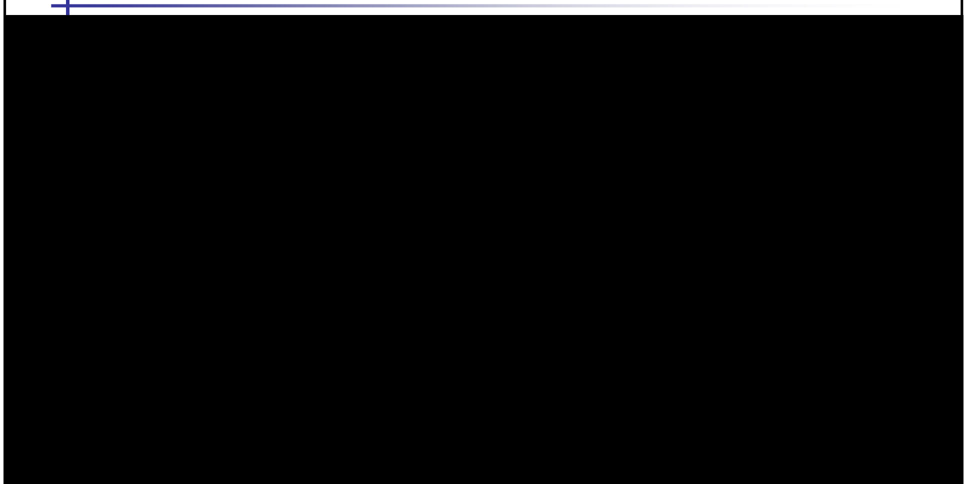
scans	59668
total acquisition time	4,699.71 seconds
traveled distance	2,587.71 meters
total rotations	262.07 radians
size	180 x 110 meters
processing time	< 30 minutes

GraphSLAM



Visual SLAM for Flying Vehicles
Bastian Steder, Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard

Autonomous Blimp



Recap – tentative syllabus

- **Control:** underactuation, controllability, Lyapunov, dynamic programming, LQR, feedback linearization, MPC
- **Reinforcement learning:** value iteration, policy iteration, linear programming, Q learning, TD, value function approximation, Sarsa, LSTD, LSPI, policy gradient, imitation learning, inverse reinforcement learning, reward shaping, exploration vs. exploitation
- **Estimation:** Bayes filters, KF, EKF, UKF, particle filter, occupancy grid mapping, EKF slam, GraphSLAM, SEIF, FastSLAM
- **Manipulation and grasping:** force closure, grasp point selection, visual servo-ing, more sub-topics tbd
- **Case studies:** autonomous helicopter, Darpa Grand/Urban Challenge, walking, mobile manipulation.
- **Brief coverage of:** system identification, simulation, pomdps, k-armed bandits, separation principle