CS 687
Jana Kosecka

Advanced topics

Deep q-learning

These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley.

Deep Q-learning

- Previously Q(s,a) function approximation
- Learn optimal linear weighting of the features
- Analogy with the recursive least squares – update the weights

- Can we use more complicated functions and avoid the feature selection stage

2

Deep Q-learning

- Atari game break out

$Q(s, a; \theta) \approx Q^*(s, a)$

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy **π***)

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

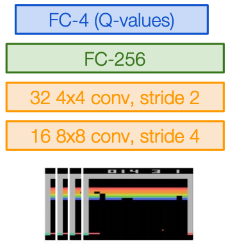
4

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$: neural network with weights θ

A single feedforward pass to compute Q-values for all actions from the current state => efficient!



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

5

Experience Replay

Learning from batches of consecutive samples is problematic:
- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**
- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Each transition can also contribute to multiple weight updates => greater data efficiency

6

Experience Replay

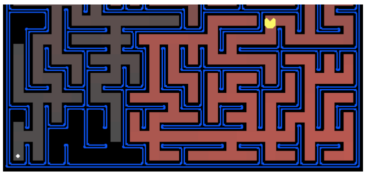
Algorithm 1 Deep Q-learning with Experience Replay

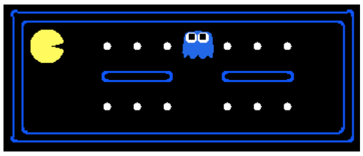
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

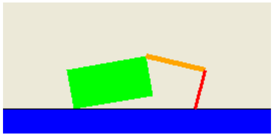
Atari Breakout

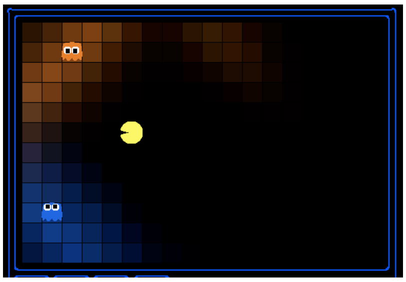
- Video Deep Q-learning

So Far: Foundational Methods

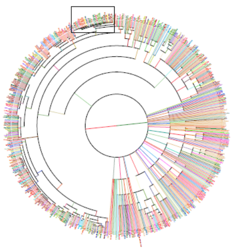


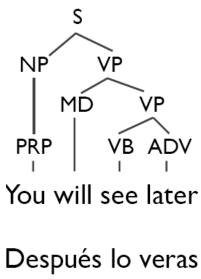






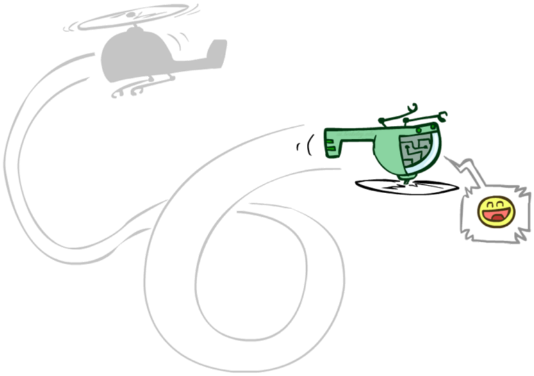
Now: Advanced Applications





AlphaGo

Robotic Helicopters



Motivating Example

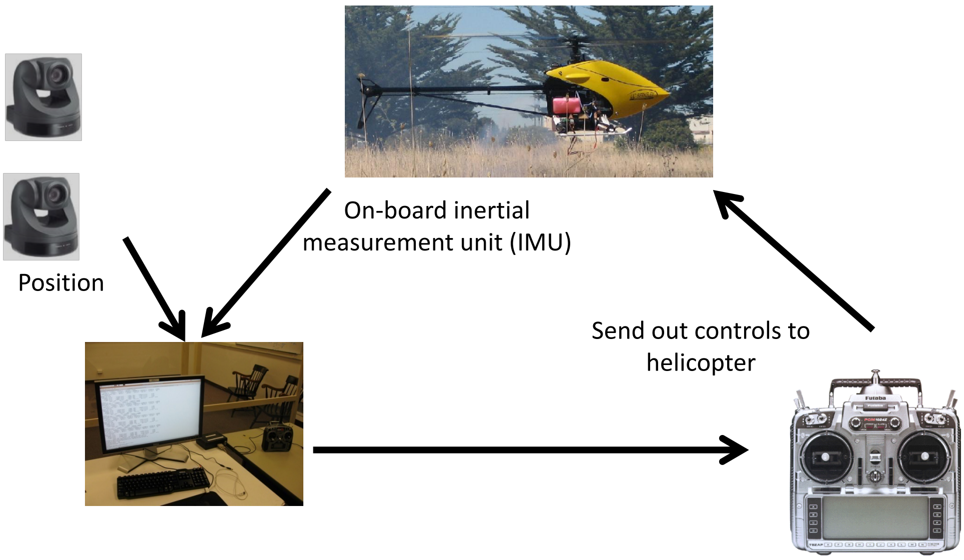
- How do we execute a task like this?

[VIDEO: tictoc_results.wmv]

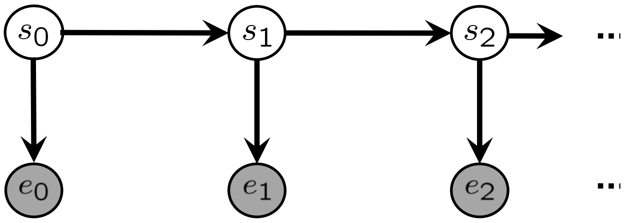
Autonomous Helicopter Flight

- Key challenges:
  - Track helicopter position and orientation during flight
  - Decide on control inputs to send to helicopter

Autonomous Helicopter Setup



HMM for Tracking the Helicopter



- State: $s = (x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$
- Measurements: [observation update]
  - 3-D coordinates from vision, 3-axis magnetometer, 3-axis gyro, 3-axis accelerometer
- Transitions (dynamics): [time elapse update]
  - $s_{t+1} = f(s_t, a_t) + w_t$ f: encodes helicopter dynamics, w: noise

Helicopter MDP

- State: $s = (x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$
- Actions (control inputs):
  - a_{lon} : Main rotor longitudinal cyclic pitch control (affects pitch rate)
  - a_{lat} : Main rotor latitudinal cyclic pitch control (affects roll rate)
  - a_{coll} : Main rotor collective pitch (affects main rotor thrust)
  - a_{rud} : Tail rotor collective pitch (affects tail rotor thrust)

- Transitions (dynamics):
  - $s_{t+1} = f(s_t, a_t) + w_t$

  [f encodes helicopter dynamics]

  [w is a probabilistic noise model]

- Can we solve the MDP yet?

Problem: What's the Reward?

- Reward for hovering:

$$
\begin{aligned}
R(s) \quad = \quad & -\alpha_x (x - x^*)^2 \\
& -\alpha_y (y - y^*)^2 \\
& -\alpha_z (z - z^*)^2 \\
& -\alpha_{\dot{x}} \dot{x}^2 \\
& -\alpha_{\dot{y}} \dot{y}^2 \\
& -\alpha_{\dot{z}} \dot{z}^2
\end{aligned}
$$

RL: Helicopter Flight

[Andrew Ng]

[Video: HELICOPTER]

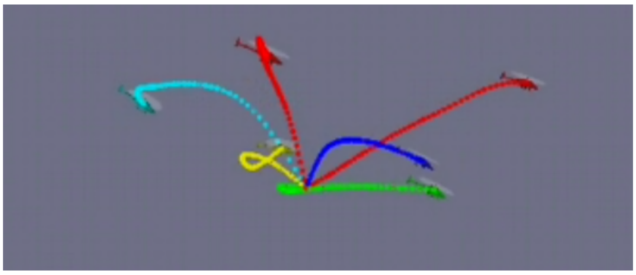
Problem for More General Case: What's the Reward?

- Rewards for "Flip"?
  - Problem: what's the target trajectory?
  - Just write it down by hand?

Flips (?)

[VIDEO: 20061204---bad.wmv]

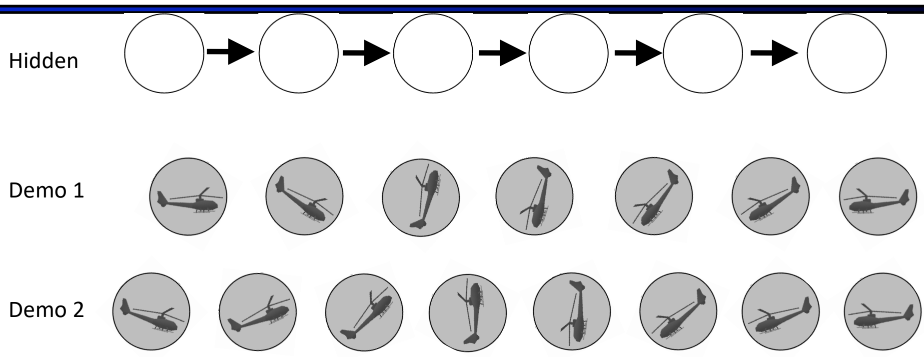
Helicopter Apprenticeship?



[VIDEO: airshow_unaligned.wmv]

Demonstrations

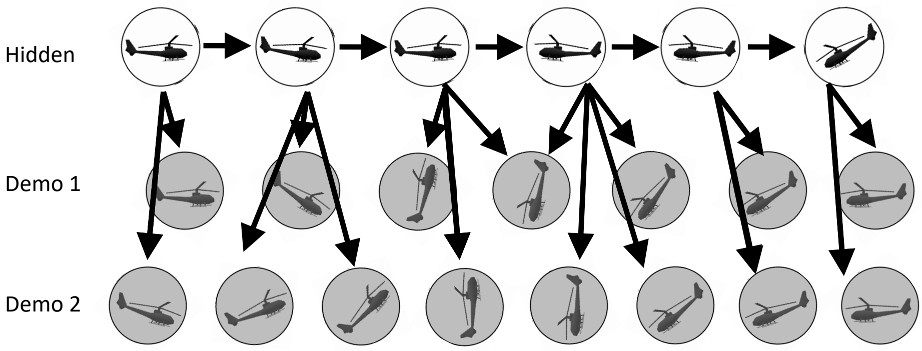
Learning a Trajectory



- HMM-like generative model
  - Dynamics model used as HMM transition model
  - Demos are observations of hidden trajectory
- Problem: how do we align observations to hidden trajectory?

Abbeel, Coates, Ng, IJRR 2010

Probabilistic Alignment using a Bayes' Net



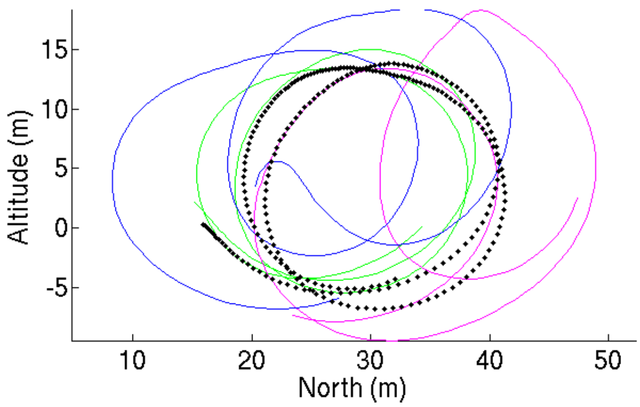
- Dynamic Time Warping
  (Needleman&Wunsch 1970, Sakoe&Chiba, 1978)
- Extended Kalman filter / smoother

Abbeel, Coates, Ng, IJRR 2010

[VIDEO: airshow_unaligned.wmv]

Aligned Demonstrations

Alignment of Samples



- Result: inferred sequence is much cleaner!

Learned Behavior

[VIDEO: airshow_trimmed.wmv]

[Abbeel, Coates, Quigley, Ng, 2010]

Learning a model for MDP

- Before state transtion probabilities and rewards known
- These are usually not given
- We can have a simulator and observed a set of trails
- Estimate T(s,a,s’) as number of times we took actio a in state a we got to state s’/number of times we too action a in state s

Continuous State MDP

- To obtain a model – learn one
- Given a simulator – execute some random policy
- Record actions and states – learn a model of dynamics $s_{t+1} = As_t + Ba_t$
- For linear model find such A and B to fit best the observed sequences, Get a deterministic model $s_{t+1} = As_t + Ba_t + \varepsilon_t$
- Stochastic model
- Or you can use locally weighted linear regression (to learn a non-linear model)

Appoximate Value Function

- E.g. linear combination of features (some functions of state)
- Approximate value function as

$$V(s) = \Theta^T \varphi(s)$$

- Now how to adopt value iteration ?
- Idea – repeatedly fit the values of parameters of value function Θ

Fitted Value Iteration

3. Repeat {

For $i = 1, \ldots, m$ {

For each action $a \in A$ {

Sample $s'_1, \ldots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP).

Set $q(a) = \frac{1}{k}\sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s'_j)$

// Hence, $q(a)$ is an estimate of $R(s^{(i)}) + \gamma \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

}

Set $y^{(i)} = \max_a q(a)$.

// Hence, $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_a \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

}

// In the original value iteration algorithm (over discrete states)

// we updated the value function according to $V(s^{(i)}) := y^{(i)}$.

// In this algorithm, we want $V(s^{(i)}) \approx y^{(i)}$, which we'll achieve

// using supervised learning (linear regression).

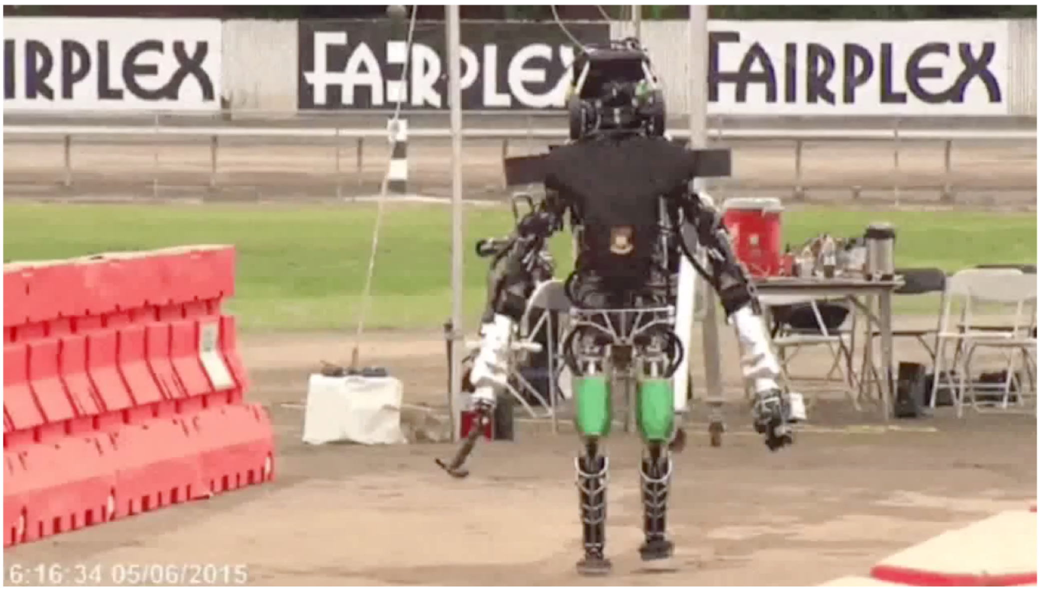
Set $\theta := \arg\min_\theta \frac{1}{2}\sum_{i=1}^{m}\left(\theta^T \phi(s^{(i)}) - y^{(i)}\right)^2$

Fitted Value Iteration

- Converge to optimal value function
- Issues: how to choose the features, how to choose the policy
- You cannot pre-compute the policy for each state
- Only when you are in some state, select the policy

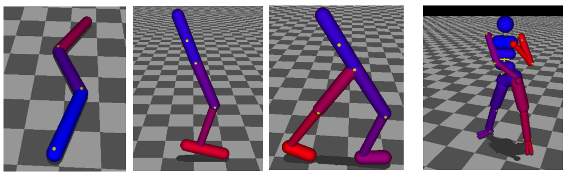
- LQR - Continuous state space, action space
  special form of reward function

For Perspective: Darpa Robotics Challenge (2015)



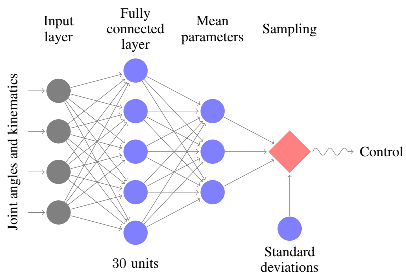
How About Continuous Control, e.g., Locomotion?

Robot models in physics simulator
(MuJoCo, from Emo Todorov)

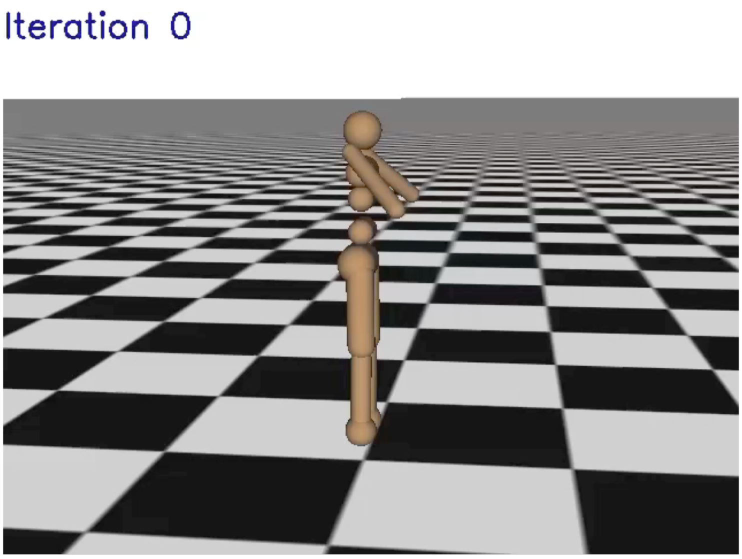


Input: joint angles and velocities
Output: joint torques

Neural network architecture:



Learning Locomotion



[Schulman, Moritz, Levine, Jordan, Abbeel, 2015]

Deep RL: Virtual Stuntman

[Peng, Abbeel, Levine, van de Panne, 2018]

Pieter Abbeel -- UC Berkeley | Gradescope | Covariant.AI

Quadruped

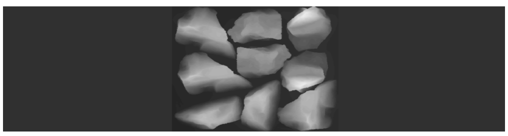
- Low-level control problem: moving a foot into a new location
  → search with successor function ~ moving the motors
- High-level control problem: where should we place the feet?
  - Reward function R(x) = w . f(s)   [25 features]

[Kolter, Abbeel & Ng, 2008]

Reward Learning + Reinforcement Learning

- Demonstrate path across the "training terrain"

- Learn the reward function
- Receive "testing terrain"---height map.



- Find the optimal policy with respect to the *learned reward function* for crossing the testing terrain.

[Kolter, Abbeel & Ng, 2008]

Without reward learning

With reward learning

Grand Challenge 2005: Barstow, CA, to Primm, NV

- 150 mile off-road robot race across the Mojave desert
- Natural and manmade hazards
- No driver, no remote control
- No dynamic passing

Autonomous Vehicles

Autonomous vehicle slides adapted from Sebastian Thrun

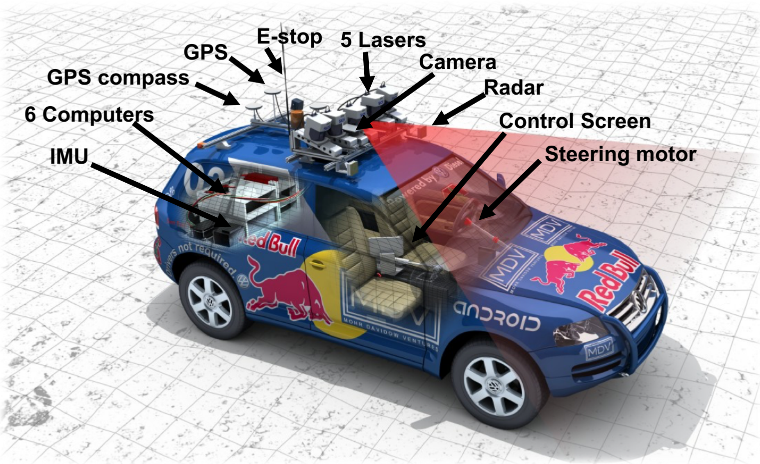
Grand Challenge 2005 Nova Video

[VIDEO: nova-race-supershort.mp4]

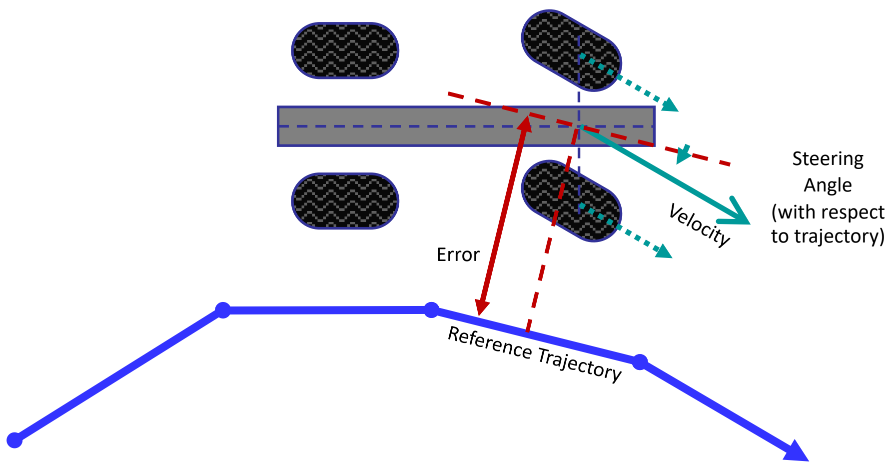
Grand Challenge 2005 – Bad

[VIDEO: grand challenge – bad.wmv]

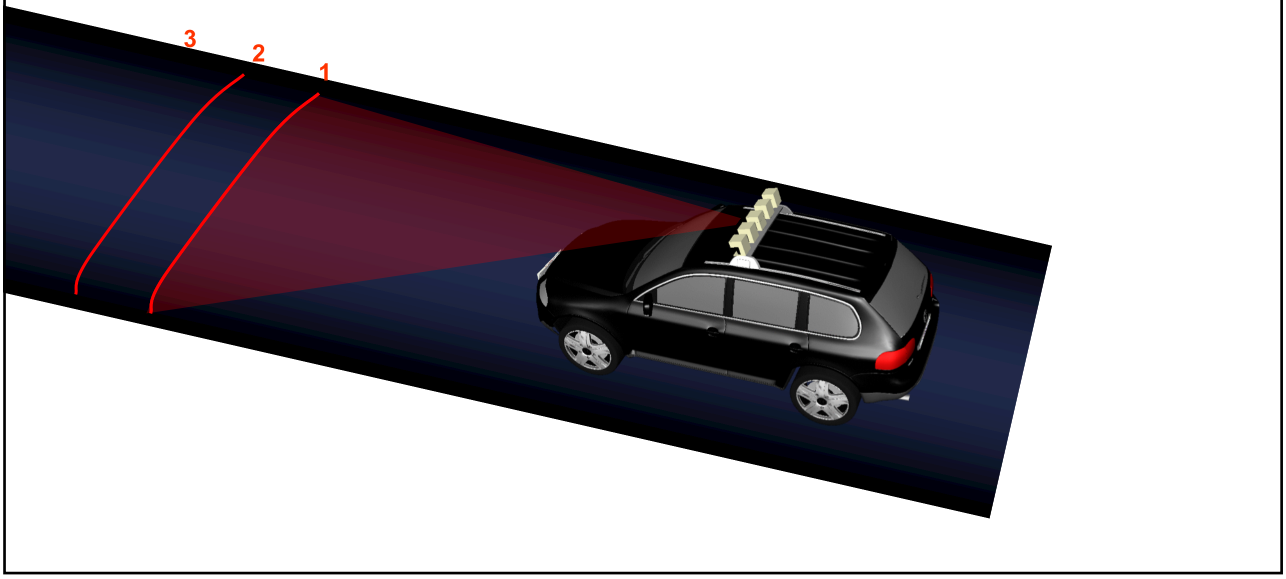
An Autonomous Car



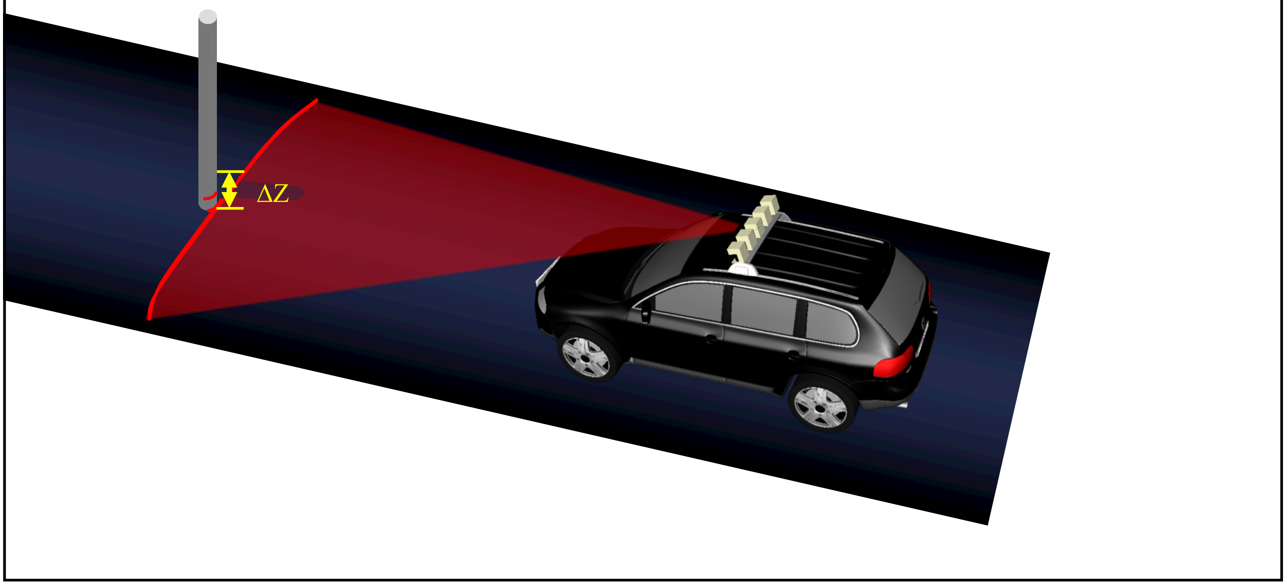
Actions: Steering Control



Laser Readings for Flat / Empty Road

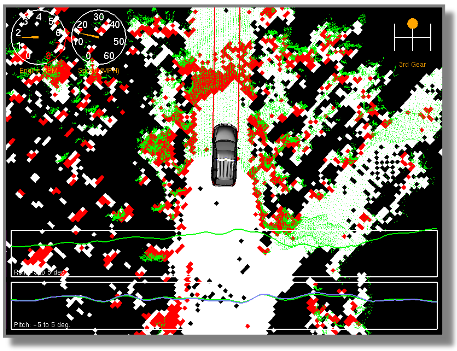


Laser Readings for Road with Obstacle



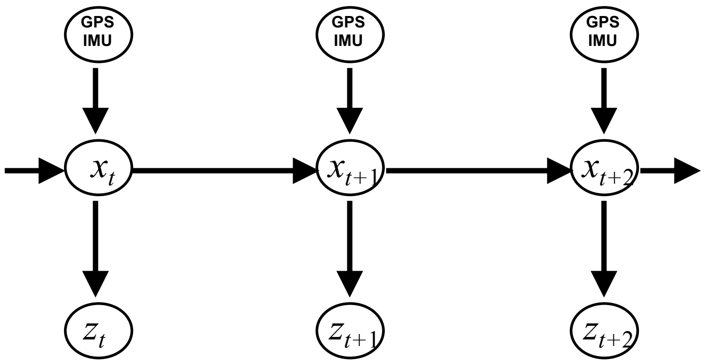
Obstacle Detection

Trigger if $|Z^i - Z^j| > 15\mathrm{cm}$ for nearby z^i, z^j

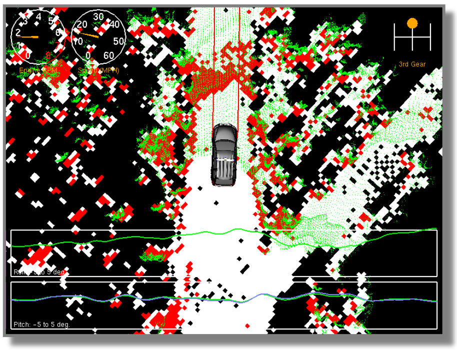


Raw Measurements: 12.6% false positives

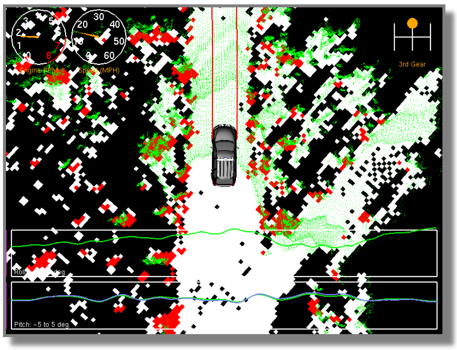
Probabilistic Error Model



HMMs for Detection

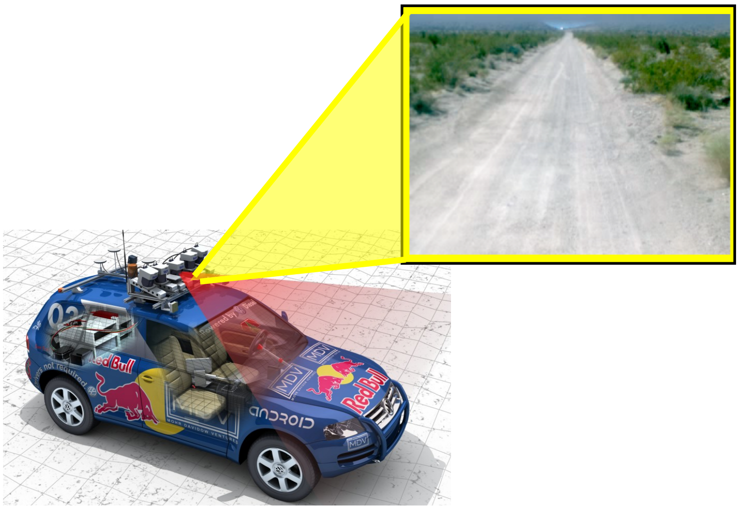


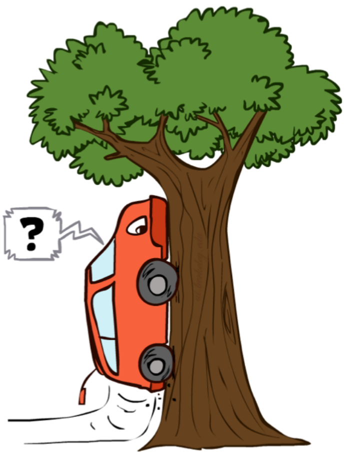
Raw Measurements: 12.6% false positives



HMM Inference: 0.02% false positives

Sensors: Camera





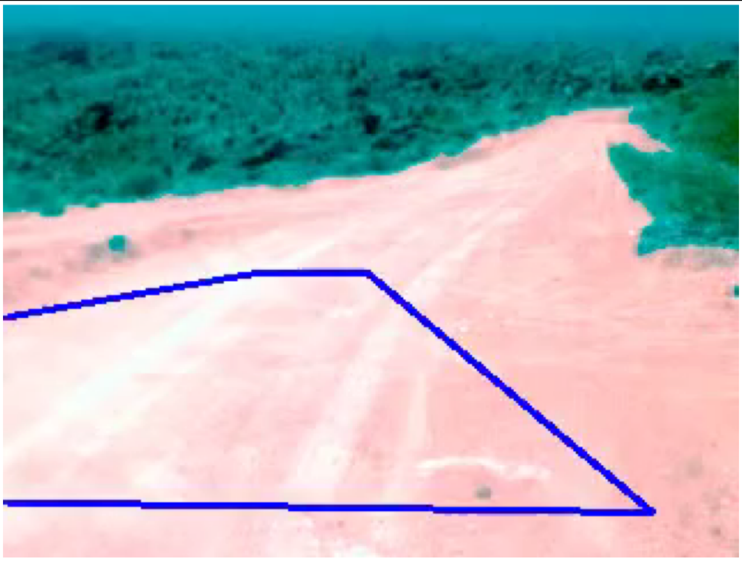
Vision for a Car

[VIDEO: lidar vision for a car]

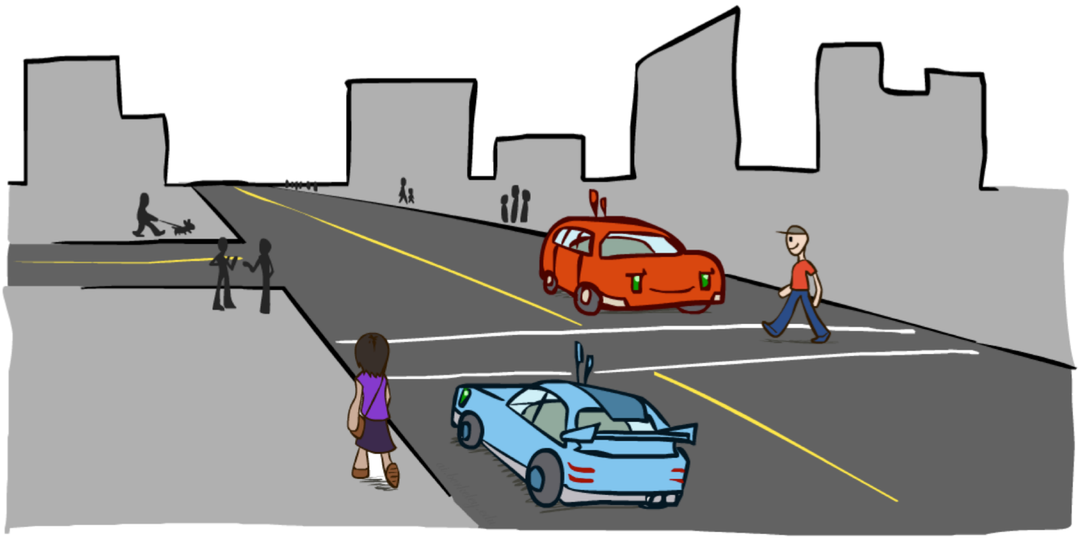
Vision for a Car

[VIDEO: self-supervised vision]

Self-Supervised Vision



Urban Environments

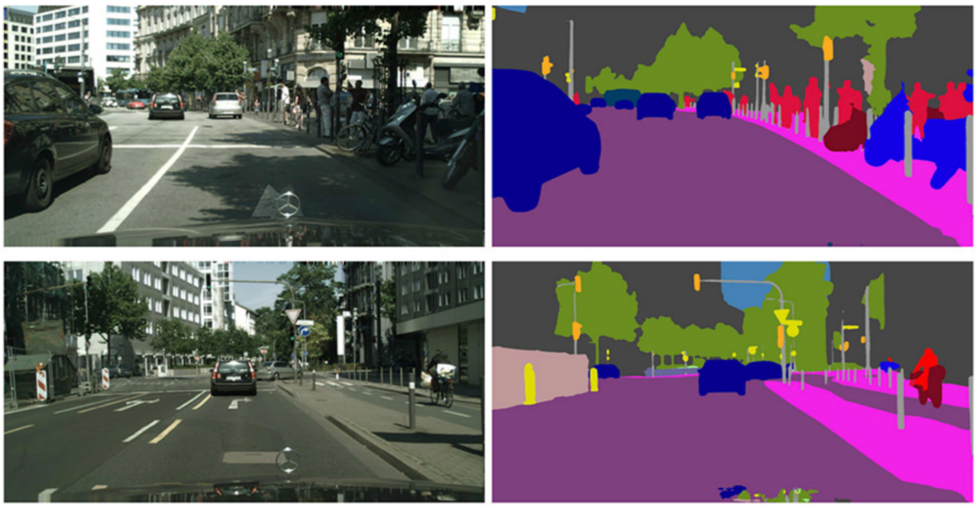


[VIDEO: ROBOTICS – gcar.m4v]

Google Self-Driving Car (2013)

(mostly lidar)

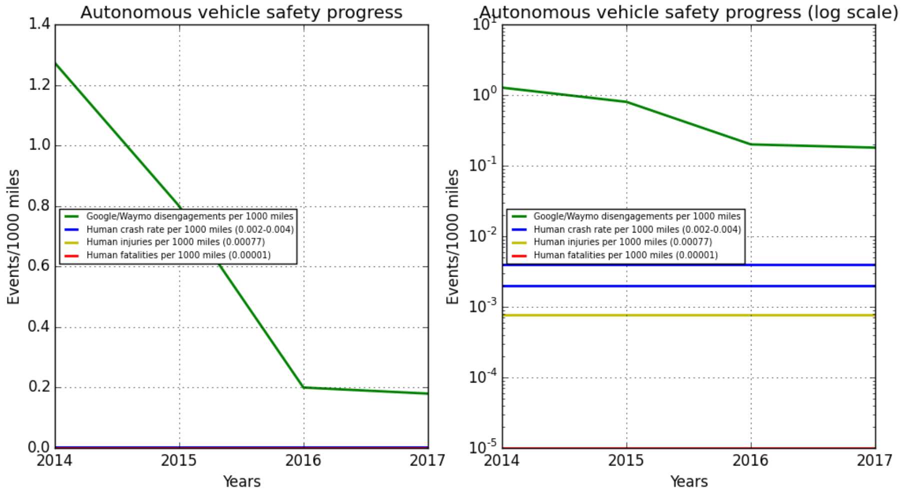
Recent Progress: NN Semantic Scene Segmentation



~ neural net classifies every pixel

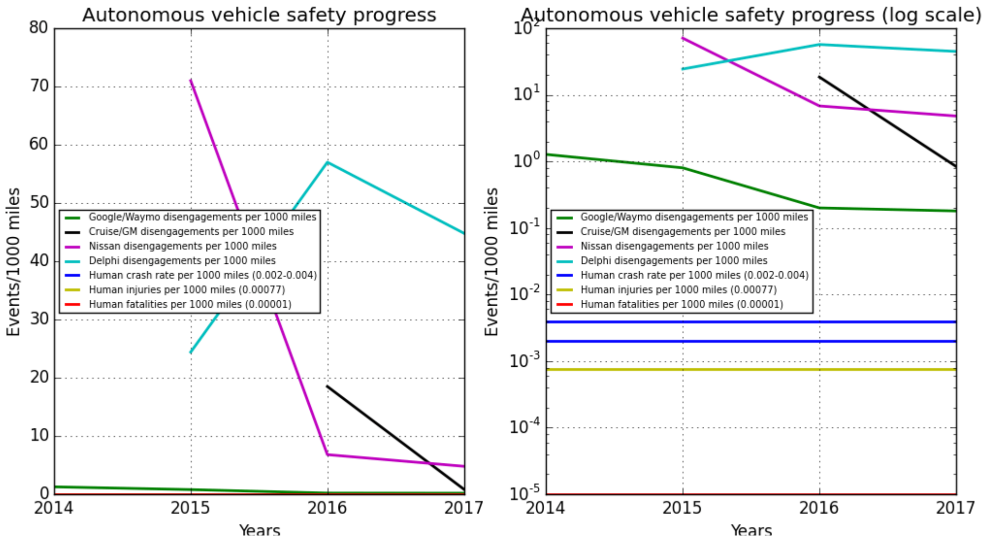
PSPNet50

Self-Driving Cars -- Stats



Pieter Abbeel -- UC Berkeley | Gradescope | Covariant.AI

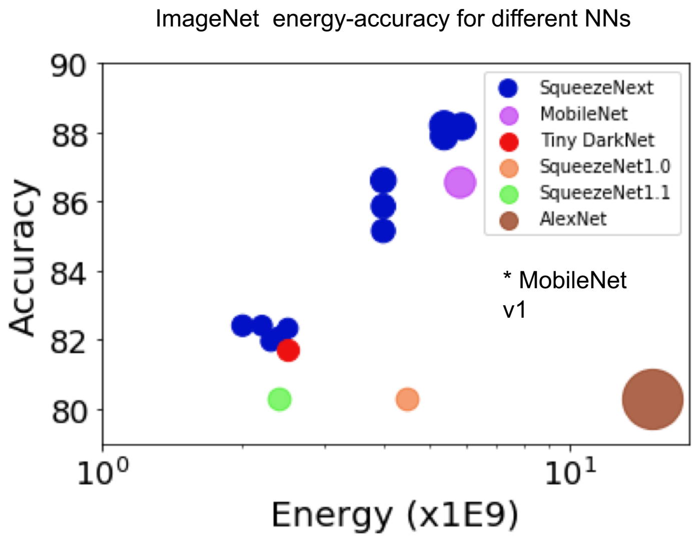
Self-Driving Cars -- Stats



Energy-Inference-Accuracy Landscape on the Squeezelator

SqueezeNext vs SqueezeNet/AlexNet

- 8% more accurate
- 2.25x better than SqueezeNet
- 7.5x better than AlexNet



[slide credit: Kurt Keutzer]