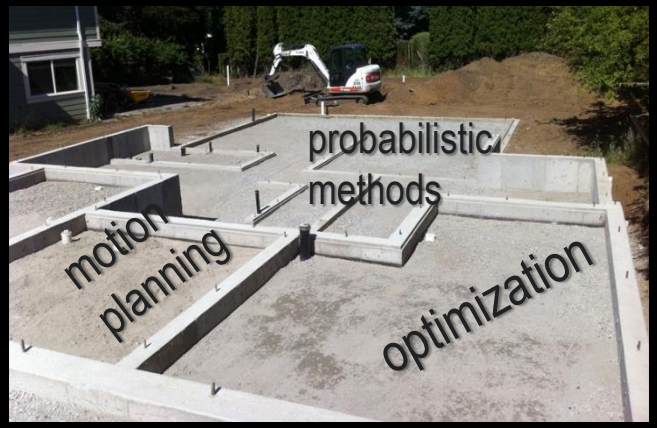
Putting it All Together

WHAT IS ALGORITHMIC ROBOTICS?

What we learned in this course

- Built a **foundation** for more advanced robotics courses/concepts



- Let's review some of the most important concepts

Fundamentals

Using Matrix Inversion

- Probably the most common problem in linear algebra: Given a matrix $\mathbf{A}$ and vector b, and the following equation

$$\mathbf{A}x = b$$

solve for the vector x

- Use this to solve a system of linear equations. For example:

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

$$\begin{array}{ccccc} \mathbf{A} & & x & & b \\ \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & = & \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \end{array}$$

Solving $\mathbf{A}x = b$

- If $\mathbf{A}$ is <u>n x n</u> and b is n x 1
  - Check rank or determinant of A to see if it is invertible.
  - If so, use the matrix inverse:

$$\mathbf{A}^{-1}\mathbf{A}x = \mathbf{A}^{-1}b$$
$$\mathbf{I}x = \mathbf{A}^{-1}b$$
$$x = \mathbf{A}^{-1}b$$

  - If not, no solution

- What if $\mathbf{A}$ is not square?

The Pseudo-inverse

- The Moore-Penrose Pseudo-inverse is defined as
  - Left pseudo-inverse:

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{T}$$

    - Has some of the properties of the inverse, most importantly:

$$\mathbf{A}^{+}\mathbf{A} = \mathbf{I}$$

  - Derivation

$$\mathbf{I} = (\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}(\mathbf{A}^{T}\mathbf{A})$$
$$\mathbf{I} = [(\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{T}]\mathbf{A}$$
$$\mathbf{I} = \mathbf{A}^{+}\mathbf{A}$$

- The right pseudo-inverse is derived similarly to get $\mathbf{A}\mathbf{A}^{+} = \mathbf{I}$

The Pseudo-inverse

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-1}\mathbf{A}^{T}$$

- Works even when $\mathbf{A}$ is not square
- What about $(\mathbf{A}^{T}\mathbf{A})^{-1}$?
  - $(\mathbf{A}^{T}\mathbf{A})$ is automatically square
  - But we need to check if $(\mathbf{A}^{T}\mathbf{A})$ is invertable

- If $\mathbf{A}$ is square and invertable, then $\mathbf{A}^{+} = \mathbf{A}^{-1}$
  - We don't loose any generality by always using the pseudo-inverse

The Pseudo-inverse

- Can use the pseudo-inverse like an inverse to solve $\mathbf{A}x = b$ when $\mathbf{A}$ is m x n and b is m x 1 :

$$\mathbf{A}^{+}\mathbf{A}x = \mathbf{A}^{+}b$$
$$\mathbf{I}x = \mathbf{A}^{+}b$$
$$x = \mathbf{A}^{+}b$$

- This is known as the least-squares solution
- Remember that $(\mathbf{A}^{T}\mathbf{A})$ must be invertable

Rotation Matrix

- Looks the same …
  but:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$r_{11} r_{12} + r_{21} r_{22} = 0$$
$$r_{11} r_{11} + r_{21} r_{21} = 1$$
$$r_{12} r_{12} + r_{22} r_{22} = 1$$

Determinant(**R**)= 1

- Can be used to effect rotation.
- Preserves linearity <u>AND distance</u> (hence, areas and angles).

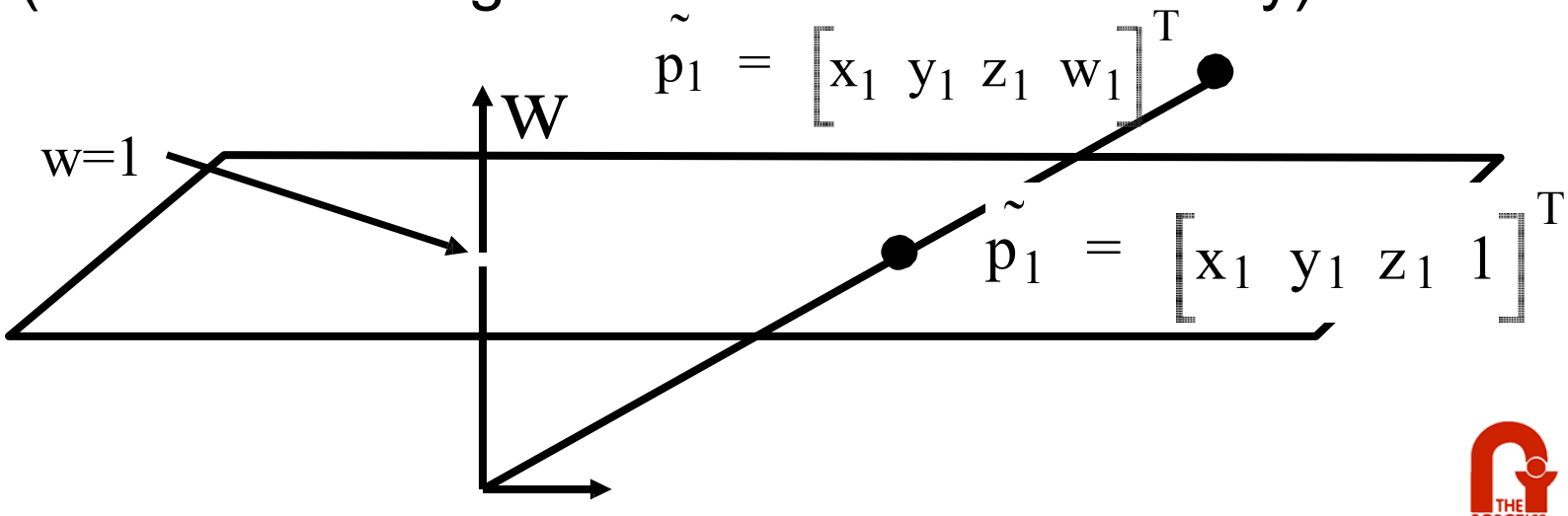
Slides by Al Kelly, minor edits by Dmitry Berenson

Homogeneous Coordinates

- Coordinates which are <u>unique up to a scale factor</u>. i.e

$$\underline{x} = 6\underline{x} = -12\underline{x} = 3.14\underline{x} = \text{same thing}$$

- The numbers in the vectors are not the same but we interpret them to mean the same thing (in fact. the thing whose scale factor is unity).

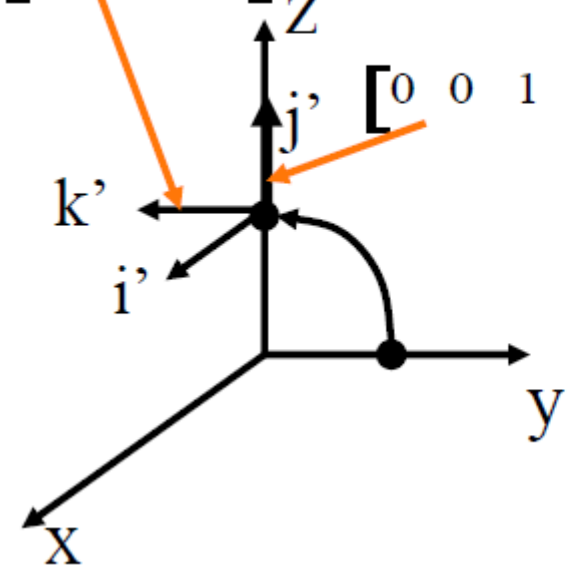


Slides by Al Kelly, minor edits by Dmitry Berenson

Example: Operating on a Frame

- Each column of this result is the transformation of the corresponding column in the original identity matrix

$$[0\ -1\ 0\ 0]^T \qquad I' = Rotx(\pi/2)Trans(0, v, 0)I$$



$$[0\ 0\ 1\ 0]^T$$

$$I' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slides by Al Kelly, minor edits by Dmitry Berenson

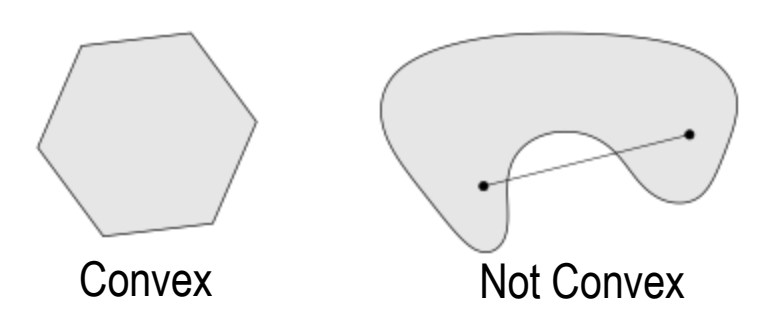
Optimization

Convex Sets

- Convex set: contains line segment between any two points in the set

$$x_1, x_2 \in C, \quad 0 \leq \theta \leq 1 \quad \Longrightarrow \quad \theta x_1 + (1-\theta)x_2 \in C$$

- Examples:



Convex Functions

The domain of the function

$f : \mathbf{R}^n \to \mathbf{R}$ is convex if $\mathbf{dom}\, f$ is a convex set and

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

for all $x, y \in \mathbf{dom}\, f$, $0 \leq \theta \leq 1$

- I.e. the line segment between (x, f(x)) and (y, f(y)) lies above the graph of f

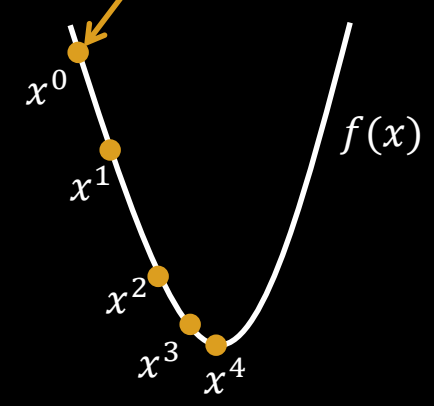
General descent algorithm

Many ways to do these

given a starting point $x \in \mathbf{dom}\, f$.
repeat
1. Determine a descent direction Δx.
2. *Line search.* Choose a step size $t > 0$.
3. *Update.* $x := x + t\Delta x$.

until stopping criterion is satisfied.

Input to the algorithm



Numerical differentiation

1. Pick a small h
2. Use one of two common Finite Difference methods:
   a) Newton's Difference Quotient

$$Df(x) \approx \frac{f(x+h) - f(x)}{h}$$

   b) Symmetric Difference Quotient

$$Df(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- There are other numerical methods which can give better estimates but use more function evaluations

$$\underset{x}{\text{minimize}}\ f(x)$$

- assume f is convex
- assume x is unconstrained

Is f <u>twice</u> continuously differentiable and is second derivative fast to compute?

- yes → Newton's method
- no → Is f <u>once</u> continuously differentiable?
  - yes → Can the gradient be computed quickly?
    - yes → Gradient Descent
    - no → Stochastic Gradient Descent (for $f(x) = f_1(x) + f_2(x) + \cdots$)
  - no → Can a <u>subgradient</u> be computed quickly?
    - yes → Subgradient Method
    - no → Gradient Descent with Numerical Differentiation ☹ (we will cover more methods for this later)

*Many more methods not covered!

The standard form general optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

- $x \in \mathbf{R}^n$ is the optimization variable
- $f_0 : \mathbf{R}^n \to \mathbf{R}$ is the objective or cost function
- $f_i : \mathbf{R}^n \to \mathbf{R}$, $i = 1, \ldots, m$, are the inequality constraint functions
- $h_i : \mathbf{R}^n \to \mathbf{R}$ are the equality constraint functions

$$p^\star = \inf\{f_0(x) \mid f_i(x) \leq 0,\ i = 1, \ldots, m,\ h_i(x) = 0,\ i = 1, \ldots, p\}$$

- $p^\star = \infty$ if problem is infeasible (no x satisfies the constraints)
- $p^\star = -\infty$ if problem is unbounded below

Duality

The Primal Problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

The Dual Problem

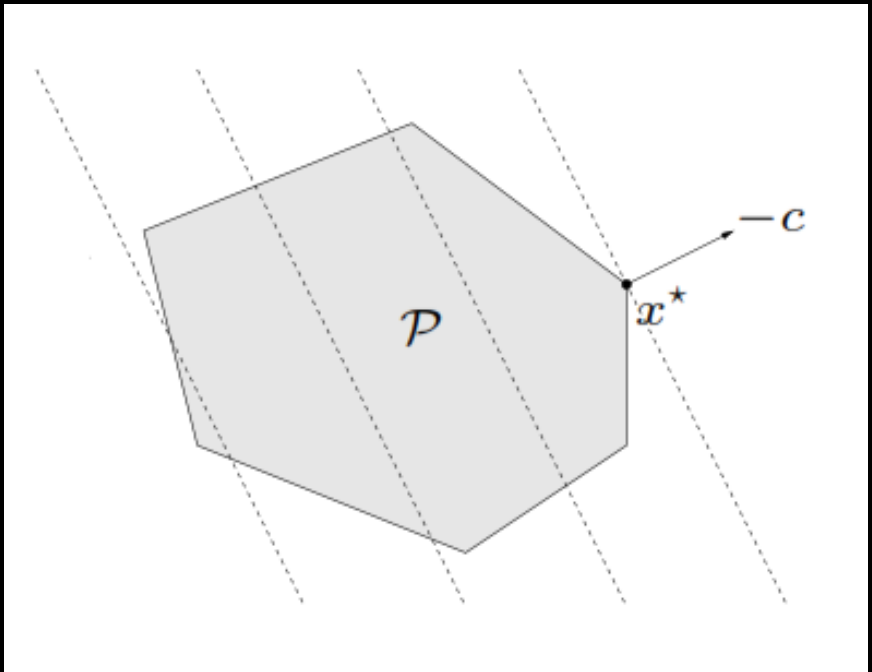
$$\begin{array}{ll} \underline{\text{maximize}} & g(\lambda, \nu) \\ \text{subject to} & \lambda \succeq 0 \end{array}$$

- Solution to dual problem is a lower-bound on solution to primal problem p^* (more on this soon)
- *The Dual problem is always convex*, regardless of convexity of primal
  - We can use local methods to solve it!
- λ, ν are dual feasible if $\lambda \geq 0, (\lambda, \nu) \in \mathbf{dom}\ g$

Linear Programming

- The feasible set is a polyhedron

$$\begin{array}{ll} \text{minimize} & c^T x + d \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

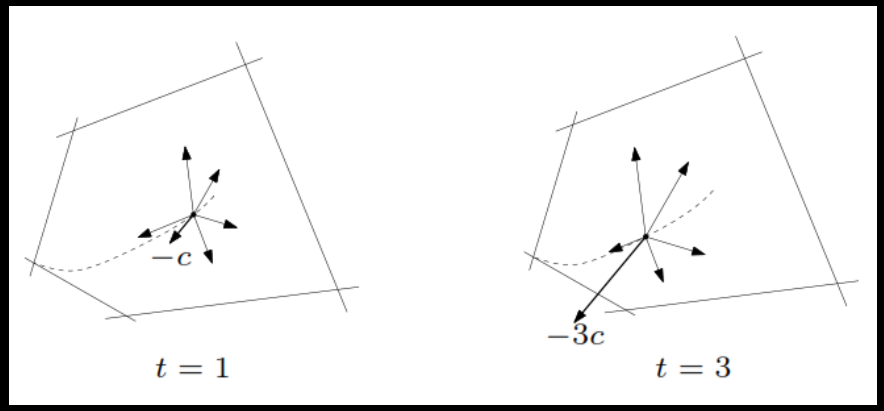


Barrier Method

- Idea: Trace out the central path by increasing t until you get to optimum
- Example for LP:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \ldots, 6 \end{array}$$

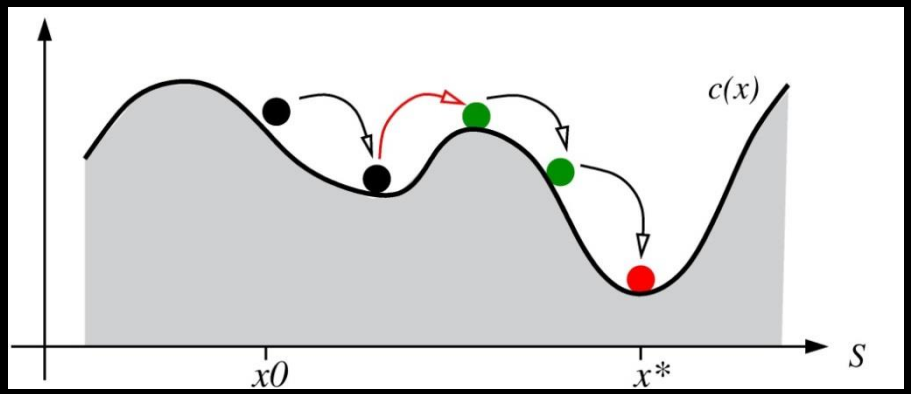
$$x^*(t) = \min_{feasible\ x} tc^T x - \sum_{i=1}^{m} \log(-a_i^T x + b_i)$$



Simulated annealing

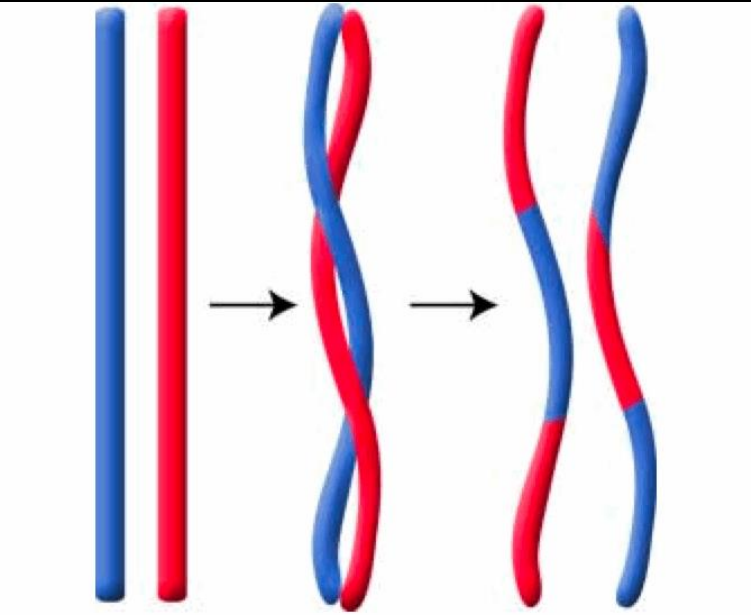
- Hill climbing problems
  - Gets stuck on plateaus
  - Returns sub-optimal solutions (local minima)

- Simulated annealing main idea: explicitly inject variability into the search process



Genetic algorithms

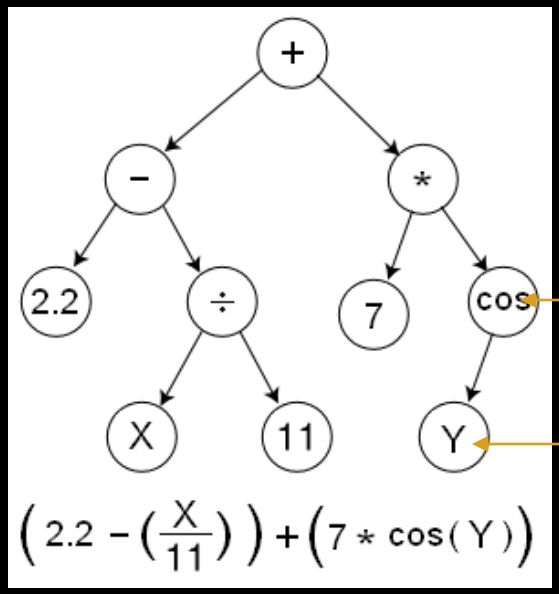
- Inspired (loosely) by the process of evolution in nature
- Operators
  - Crossover: New states generated from two *parent* states.
  - Mutation: Randomly change a component of the state



http://www.genomenewsnetwork.org/resources/whats_a_genome/Chp3_2.shtml

Genetic Programming

- Evolve *functions* instead of vectors of numbers
- Individuals represented as trees:



Non-leaf nodes are operators

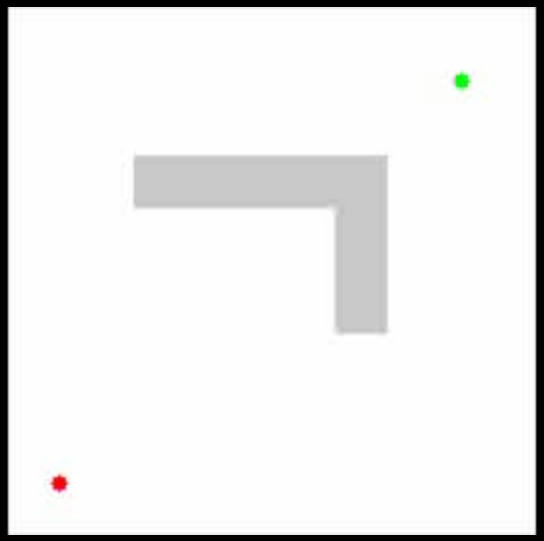
Leaf nodes are operands
(input variables or constants)

- Useful for regression problems

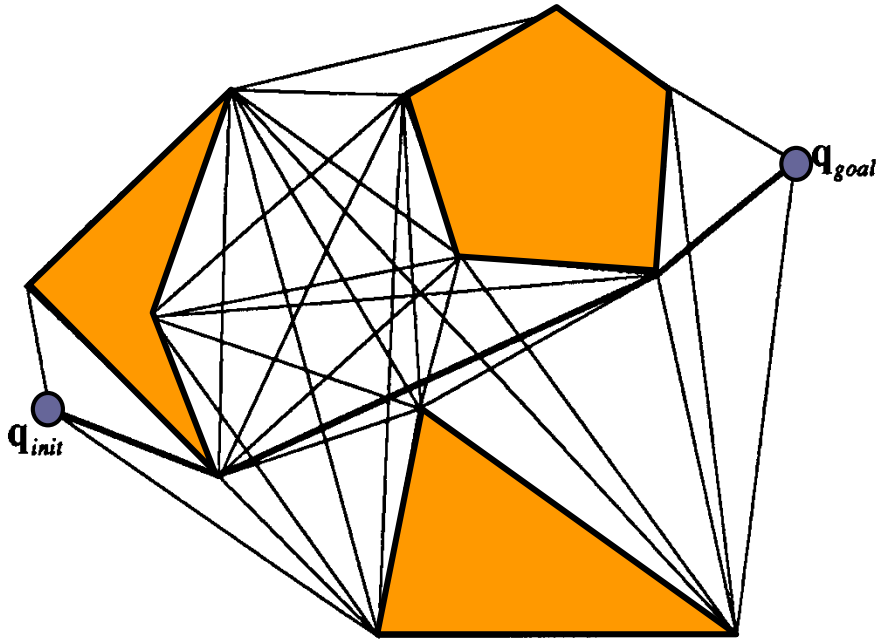
Motion Planning

A* Search

- Open list is a *priority queue*, nodes are sorted according to f(n)
- g(n) is sum of edge costs from root node to n

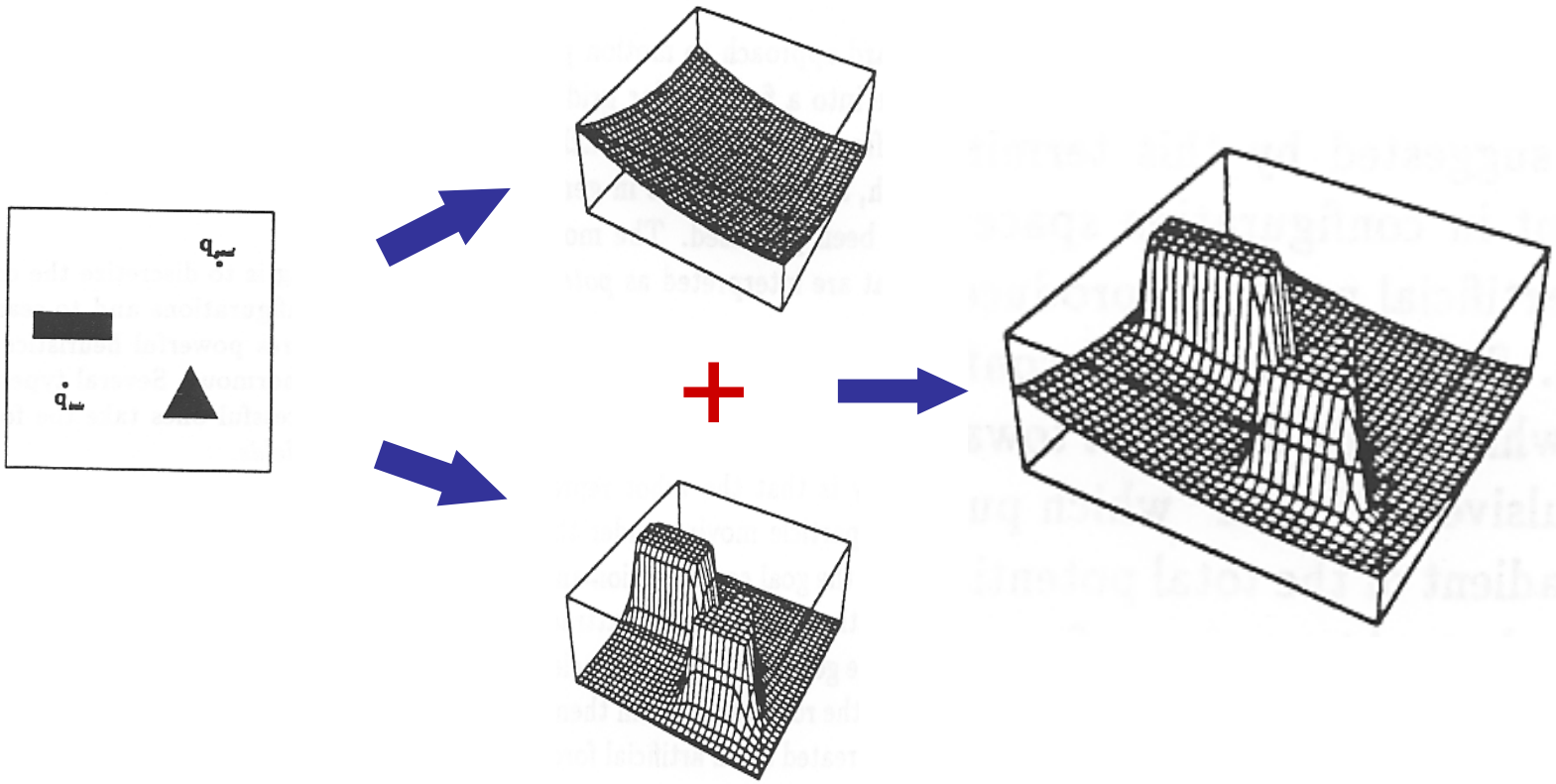


Visibility graph

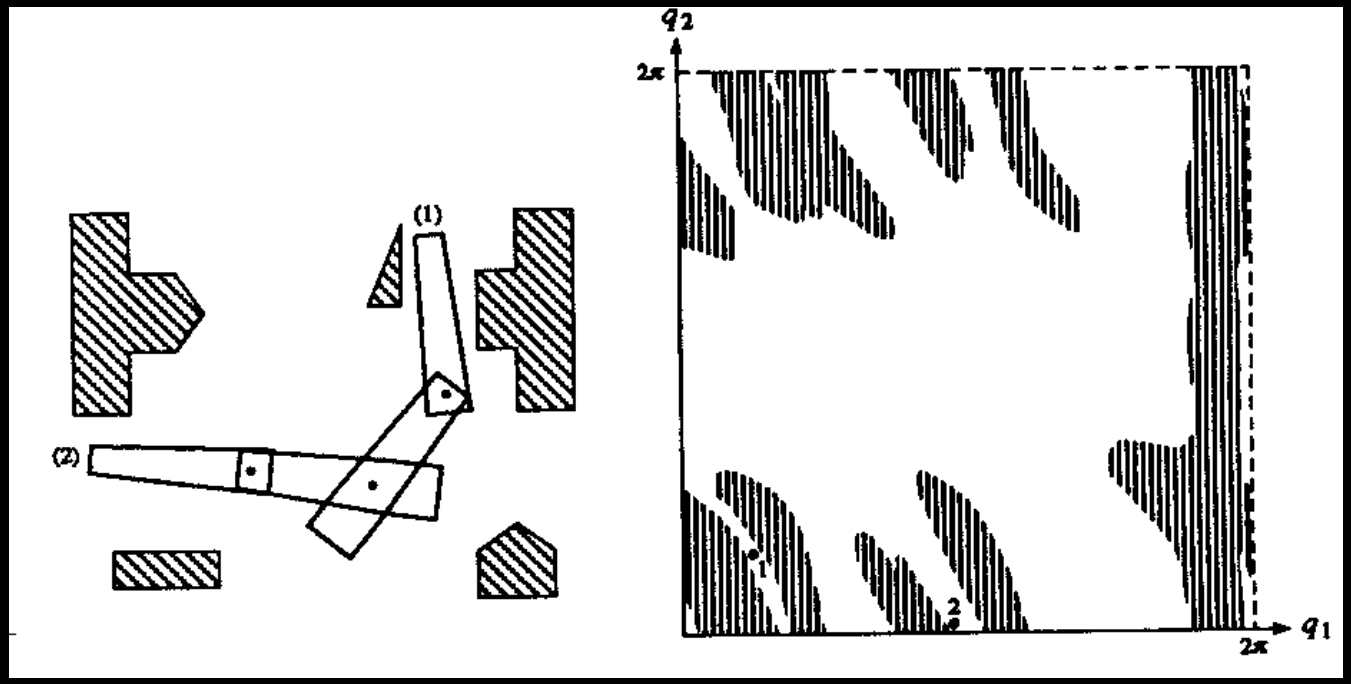


- A **visibility graph** is a graph such that
  - Nodes: q_{init}, q_{goal}, or an obstacle vertex.
  - Edges: An edge exists between nodes u and v if the line segment between u and v is an obstacle edge or it does not intersect the obstacles.

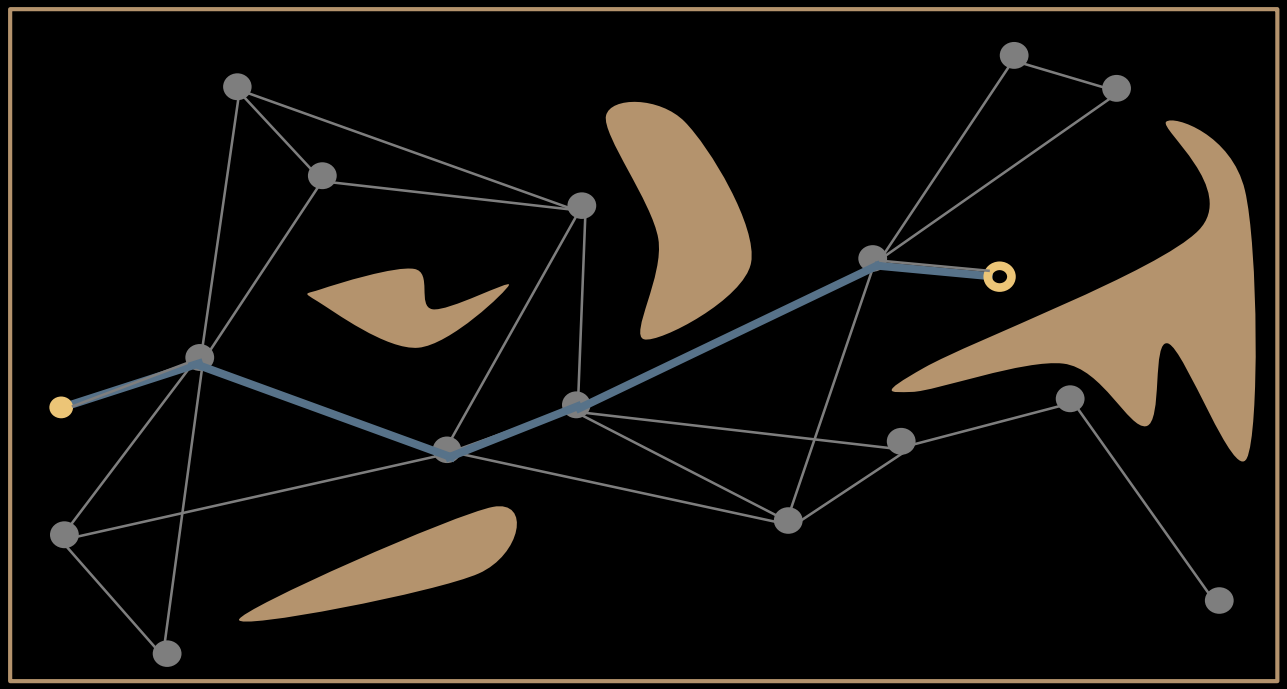
Potential Fields



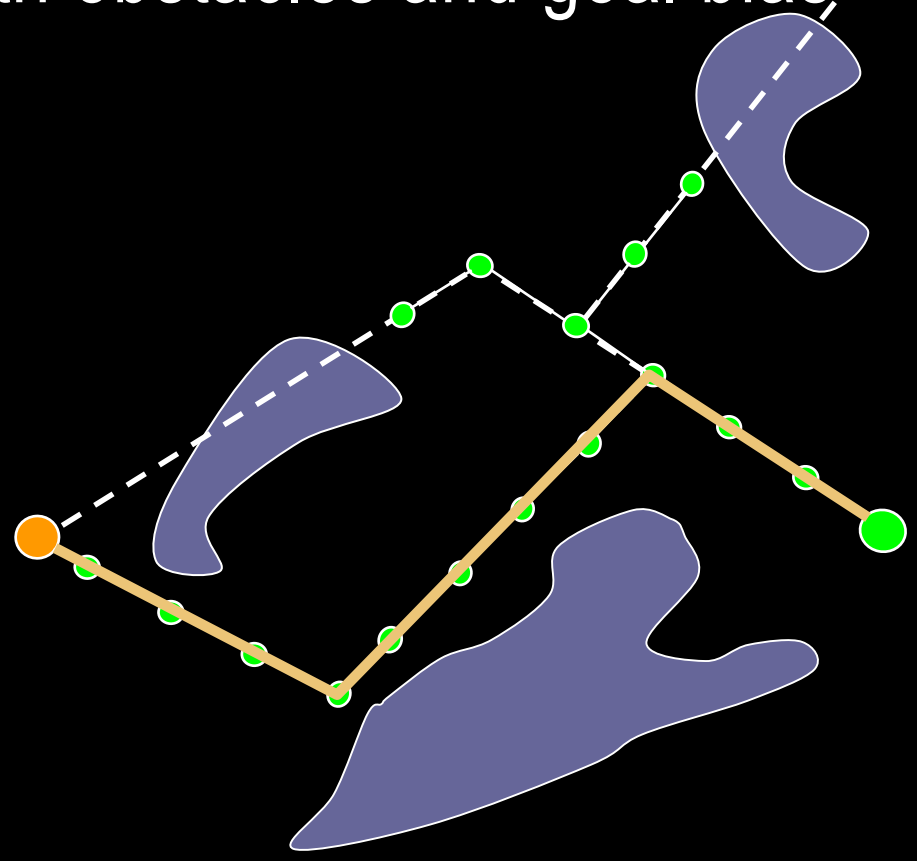
Configuration Space for Articulated Robots



PRM Example



RRT with obstacles and goal bias

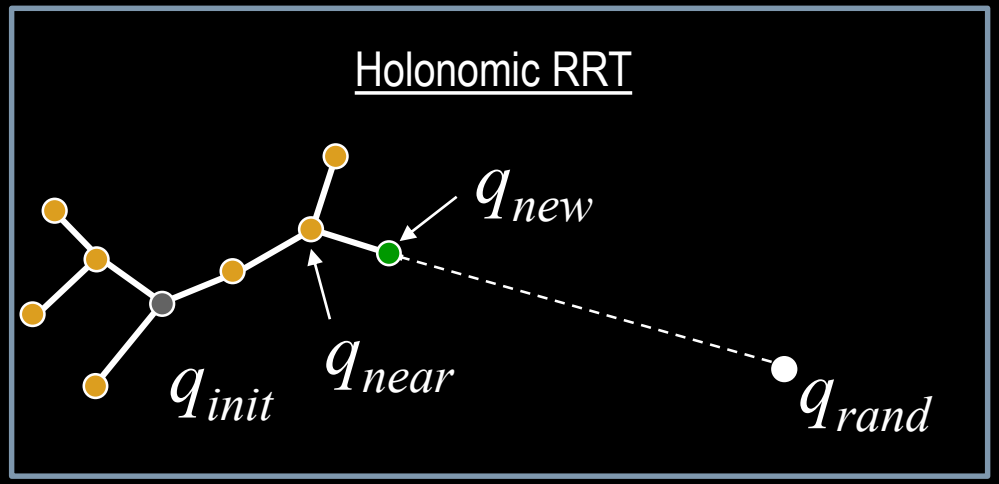


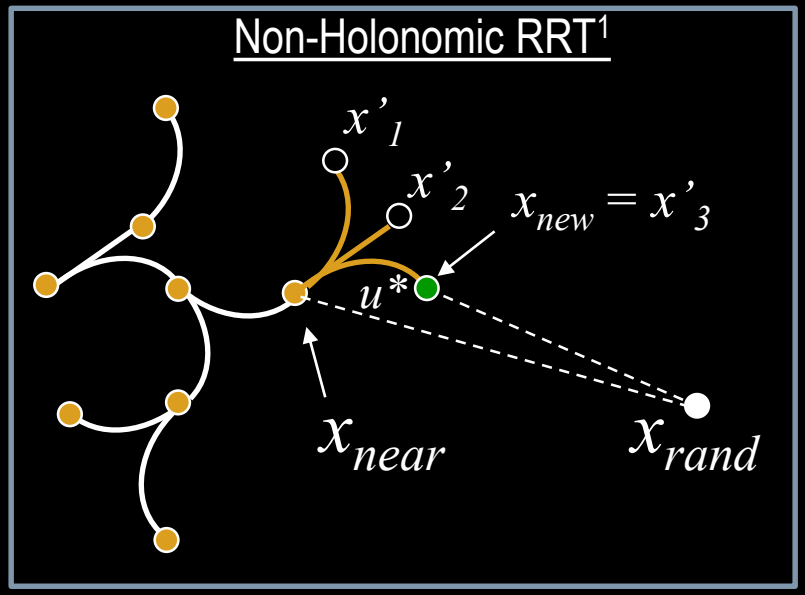
RRTs for Non-Holonomic Systems

- Apply motion primitives (i.e. simple actions) at q_{near}

$$x' = f(x,u) \text{ --- use action } u \text{ from } x \text{ to arrive at } x'$$

$$\text{chose } u_* = \arg\min_{u_i} d(f(x,u_i), x_{rand})$$



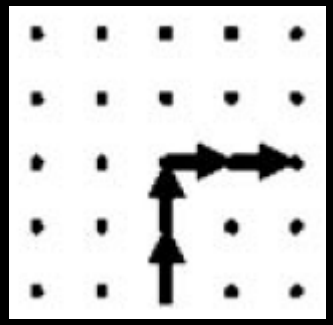


- You probably won't reach x_{rand} by doing this
  - Key point: No problem, you're still exploring!

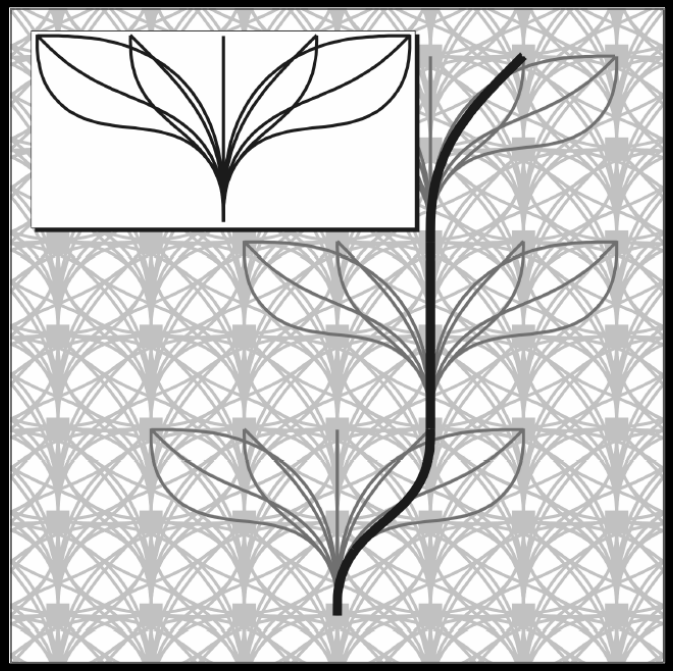
[1]often called *Kinodynamic RRT*

State Lattice for Non-holonomic planning

- Pre-compute state lattice
- Two methods to get lattice:
  - Forward: For certain systems, can sequence primitives to make lattice
  - Inverse: Discretize space, use BVP solvers to find trajectories between states
- Impose continuity constraints at graph vertices
- Search state lattice like any graph (i.e. A*)



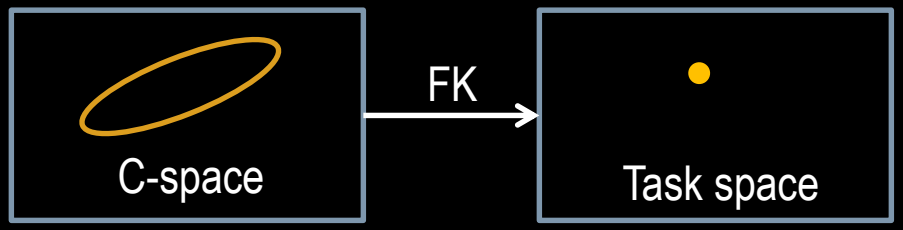
Traditional lattice yields discontinuous motion



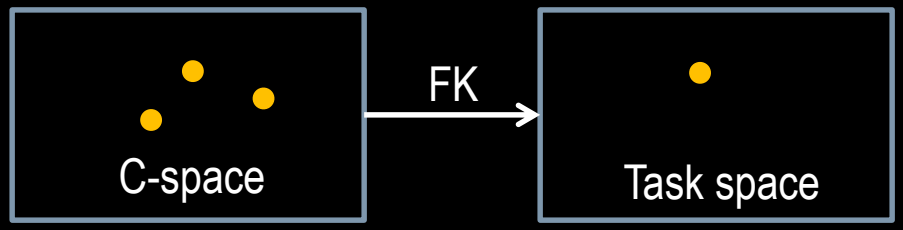
Pivtoraiko et al. 2009

C-Space and Task Space

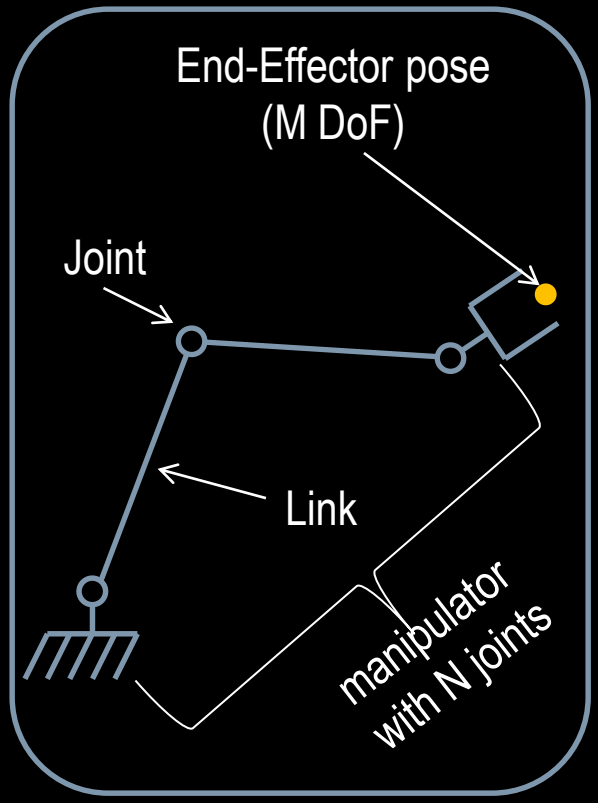
- If N > M, FK maps *a continuum* of configurations to *one* end-effector pose:



- If N = M, FK maps a *finite number* of configurations to one end-effector pose:



- If N < M, you're in trouble (may not be able to reach a target pose)



Iterative Jacobian Pseudo-Inverse Inverse Kinematics

While true

$x_{current} = FK(q_{current})$

$\dot{x} = (x_{target} - x_{current})$

error = $||\dot{x}||$

If error < threshold

return $q_{current}$

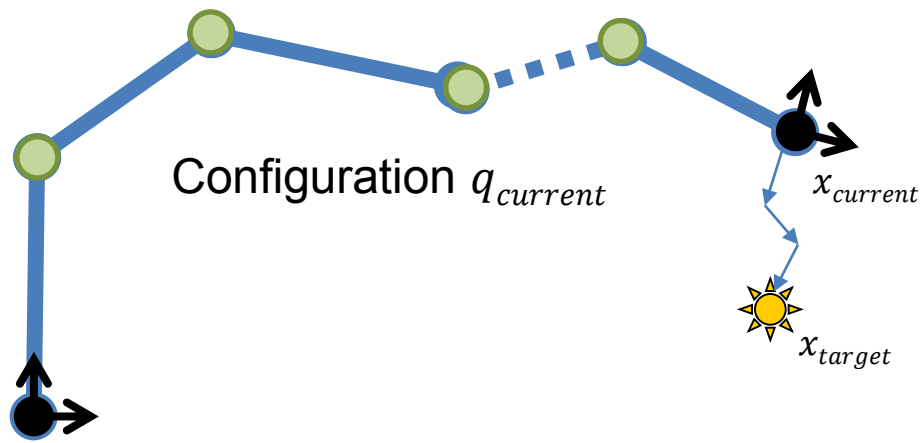
$\dot{q} = J(q)^{+}\,\dot{x}$

If($||\dot{q}|| > \alpha$)

$\dot{q} = \alpha(\dot{q} / ||\dot{q}||)$

$q_{current} = q_{current} + \dot{q}$

end



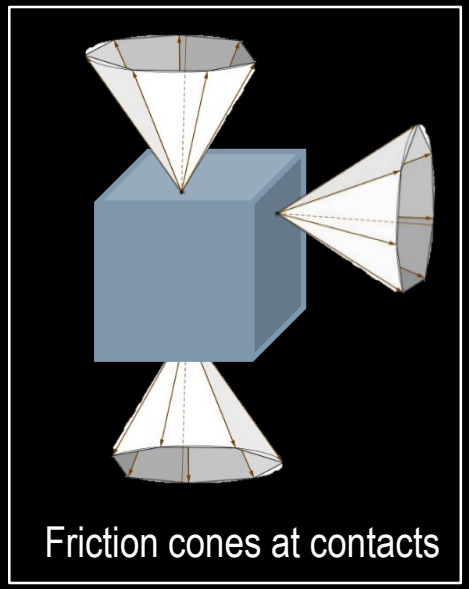
- This is a local method, it will get stuck in local minima (i.e. joint limits)!!!
- α is the step size
- Numerical error handling not shown
- A correction matrix has to be applied to the angular velocity components to map them into the target frame (not shown)

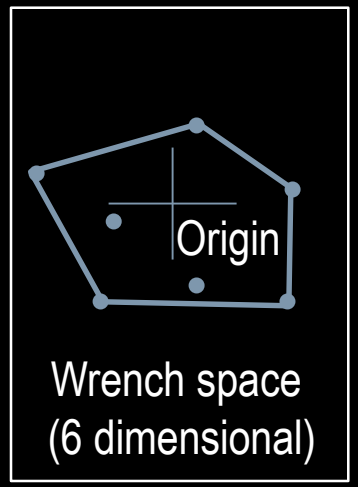
Testing for Force Closure Grasps

- Many algorithms exist to test for force closure, here is one:

Input: Contact locations
Output: Is the grasp in Force-Closure? (Yes or No)

1. Approximate the friction cone at each contact with a set of wrenches
2. Combine wrenches from all cones into a set of points *S* in wrench space
3. Compute the *convex hull* of *S*
4. If the origin is inside the convex hull, return YES. If not, return NO.



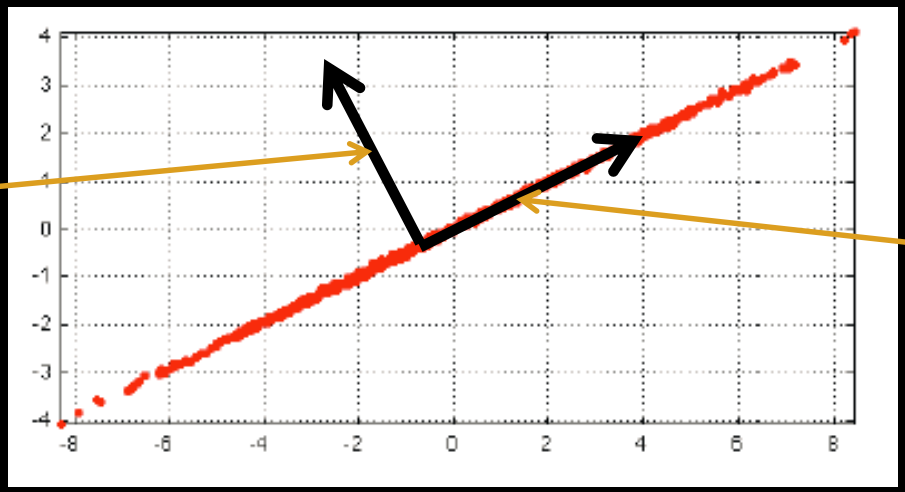


Point Cloud Processing

The main idea of PCA

- We care about the variance of the data
- High-variance implies high importance
  - Why does this make sense?

Data does not vary much in this direction



Data varies a lot in this direction

- But how do we compute the high-variance directions?

SVD

- SVD decomposes any matrix M into the following form:

M	=	U	*	Σ	*	V^T
m x n	=	m x m		m x n		n x n

Columns of U are eigenvectors of MM^T

$$\Sigma \equiv \begin{bmatrix} \sigma_{\tilde{1}} & & & & & \\ & \ddots & & & 0 & \\ & & \sigma_{\tilde{r}} & & & \\ & & & 0 & & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

Columns of V are eigenvectors of M^TM

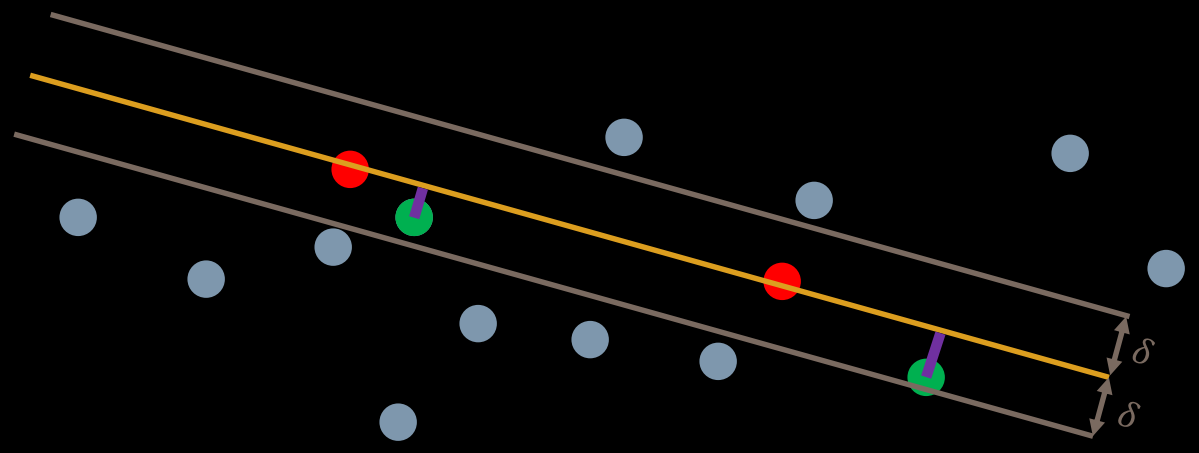
σs are $\sqrt{\text{eigenvalues}}$ of M^TM and MM^T in decreasing order of magnitude

PCA algorithm to remove rotation *and reduce dimension*

1. Given a dataset X
2. Compute the mean of X, μ
3. $X = X - \mu$ (subtract μ from every point in X)
4. Compute the covariance of X, $Q = \frac{XX^T}{(n-1)}$
5. SVD$(Q) = U\Sigma V^T$
   - Each column of V is a principle component
6. Compute the variance of each principle component: $s = \mathrm{diag}(\Sigma)^2$
7. Remove all columns of V whose corresponding entry in s is less than some small threshold, call this new matrix V_s
8. Compute $X_{new} = V_s^T X$

(Note: this may cause reflection)

RANSAC Line fitting example



Pick R (hypothetical inliers)
Fit Model to R
Find C (consensus set)
Compute Error of Model on $C \cup R$

Point set registration: Known Correspondences

- In summary, to solve

$$P = \{p_1, p_2, \dots p_n\}$$
$$Q = \{q_1, q_2, \dots, q_n\}$$

$$\underset{R \in SO(3), t \in \boldsymbol{R}^3}{\operatorname{argmin}} \sum_{i=1}^{n} \|(Rp_i + t) - q_i\|^2$$

1. Compute centroids and centered vectors:

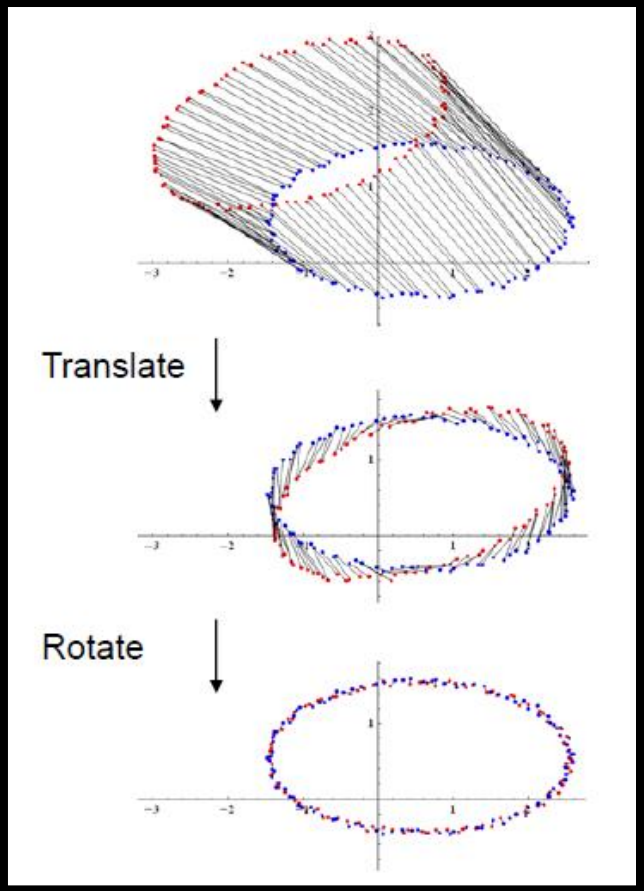
$$\bar{p} = \frac{\sum_{i=1}^{n} p_i}{n} \qquad \bar{q} = \frac{\sum_{i=1}^{n} q_i}{n} \qquad \begin{aligned} x_i &= p_i - \bar{p} \\ y_i &= q_i - \bar{q} \end{aligned}$$

2. Compute SVD of covariance matrix of centered vectors:

$$S = XY^T \qquad \mathrm{SVD}(S) = U\Sigma V^T$$

3. Compute R and t:

$$R = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(VU^T) \end{bmatrix} U^T \qquad t = \bar{q} - R\bar{p}$$



Iterative Closest Point (ICP)

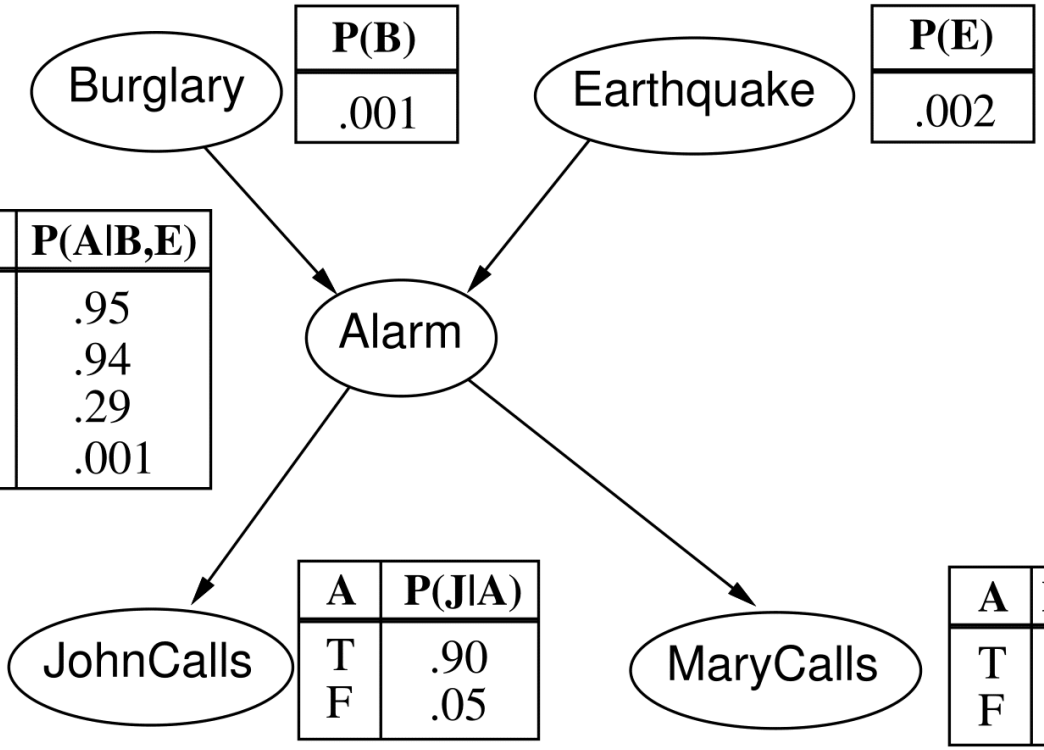
- We assumed we knew the correspondences between points
  - In practice this is rarely true
- ICP iteratively computes correspondences and registers point sets
- There are many variants of ICP, here is a simple one
- Won't always succeed!
  - Need to add a way to terminate based on
    - time
    - number of iterations
    - lack of progress
- Need to tune ϵ

```
Input: P and Q (not necessarily same size)
Output: P aligned to Q
```

```
Set P to some initial pose
While not Done
   //Compute Correspondences
   C = ∅
   For each p_i ∈ P
      find the closest q_i
      C = C ∪ {p_i, q_i}

   //Compute Transform
   //(see previous slide)
   R, t ← GetTransform(C_p, C_q)

   If Σ_{i=1}^{n} ‖(R C_{p_i} + t) − C_{q_i}‖² < ϵ
      return P

   // Update all P
   For each p_i ∈ P
      p_i = R p_i + t
```

Probabilistic Reasoning

Example contd.



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

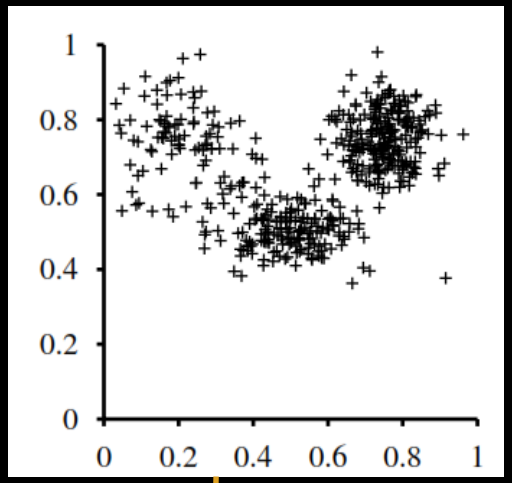
A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

Learning a Gaussian Mixture Model (GMM)

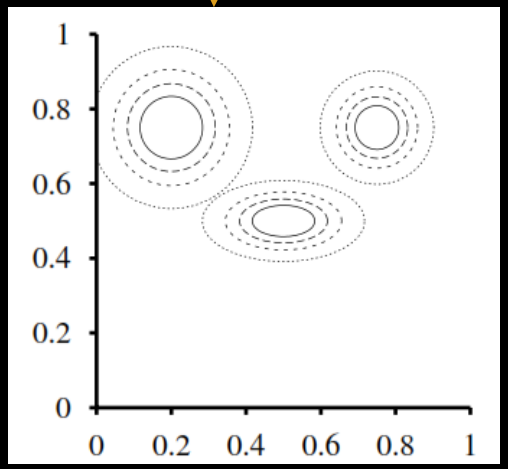
- A GMM is simply a set of Gaussians
  - The set jointly defines a probability distribution over the space

- Lets say we want to model a set of data points with a GMM
- Problem: we don't know which data point came from which Gaussian
  - i.e. we don't know how to partition the data



data

Expectation Maximization



GMM

Hidden Markov Models (HMMs)

- An HMM is a temporal probabilistic model in which the state is described by a **single discreet random variable**.

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of X_t is $\{1, \ldots, S\}$

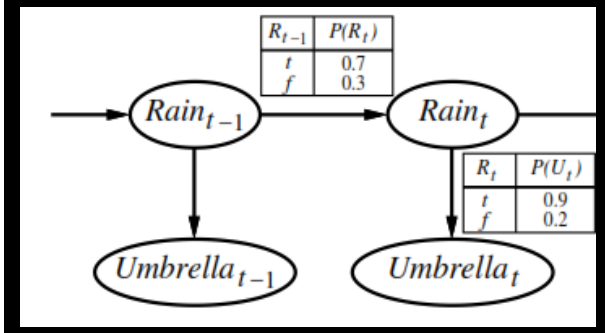
Transition matrix $\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

Sensor matrix $\mathbf{O}_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

Forward and backward messages as column vectors:

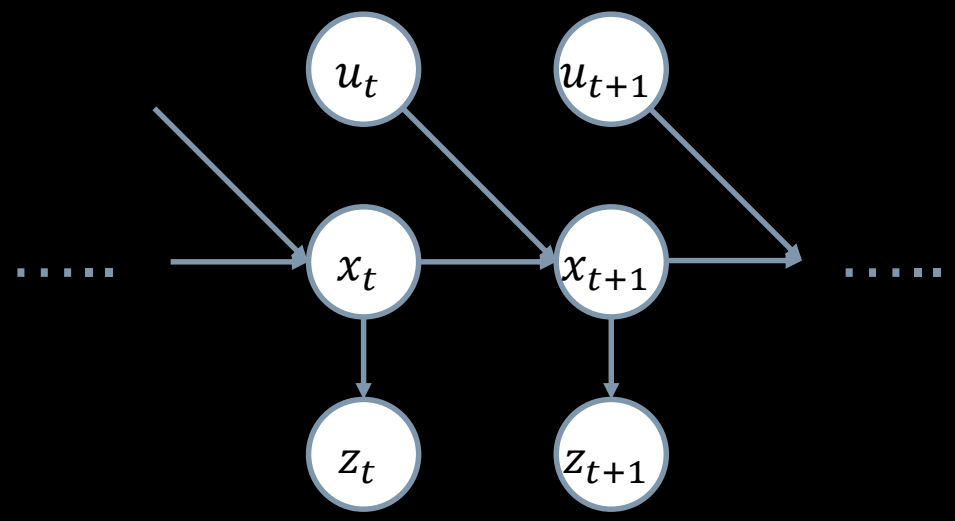
$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t}$$
$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$$

Forward-backward algorithm needs time $O(S^2 t)$ and space $O(St)$



Kalman Filters

- The real world is not discrete! Need to consider **continuous** variables



- Kalman filters are used to track state of robots, chemical plants, planets, etc.
- Key Idea: Arbitrary continuous models are intractable, so represent everything with *Gaussians*
  - Gaussian prior, linear Gaussian transition model and sensor model

Prediction-Correction Cycle for Kalman Filters

Prediction

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

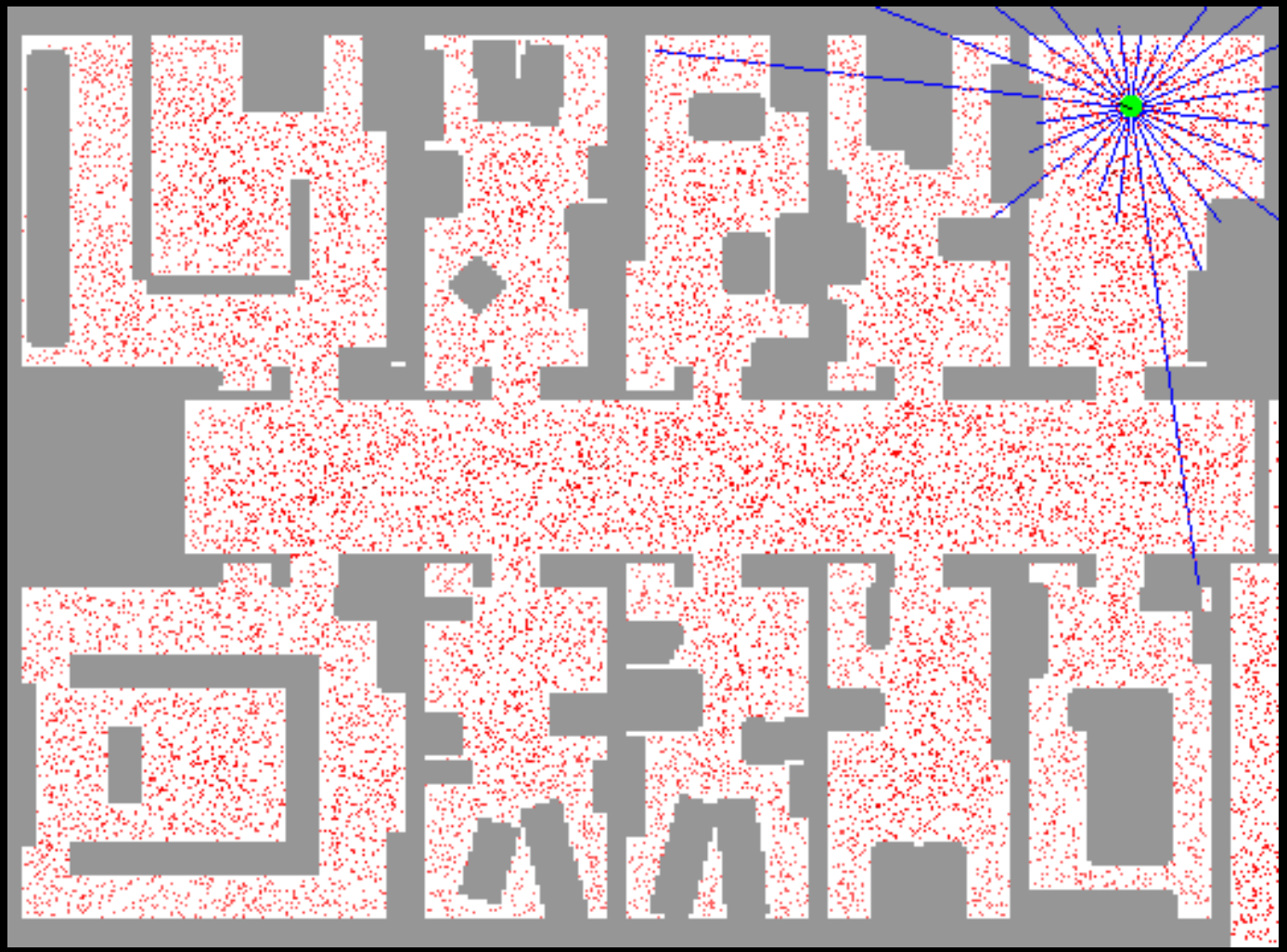
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Correction

KF Variants:

- Extended Kalman Filter
- Unscented Kalman Filter

Particle filter



Fox et al.: Mobile robot localization with 24 sonar sensors

http://www.cs.washington.edu/ai/Mobile_Robotics/mcl/

The Markov Zoo

- Markov process + partial observability = HMM
- Markov process + actions = MDP
- Markov process + partial observability + actions = HMM + actions = MDP + partial observability = **POMDP**

	full observability	*partial observability*
no actions	**Markov process**	**HMM**
actions	**MDP**	**POMDP**

Value Iteration vs. Policy Iteration for MDPs

- Value iteration update:

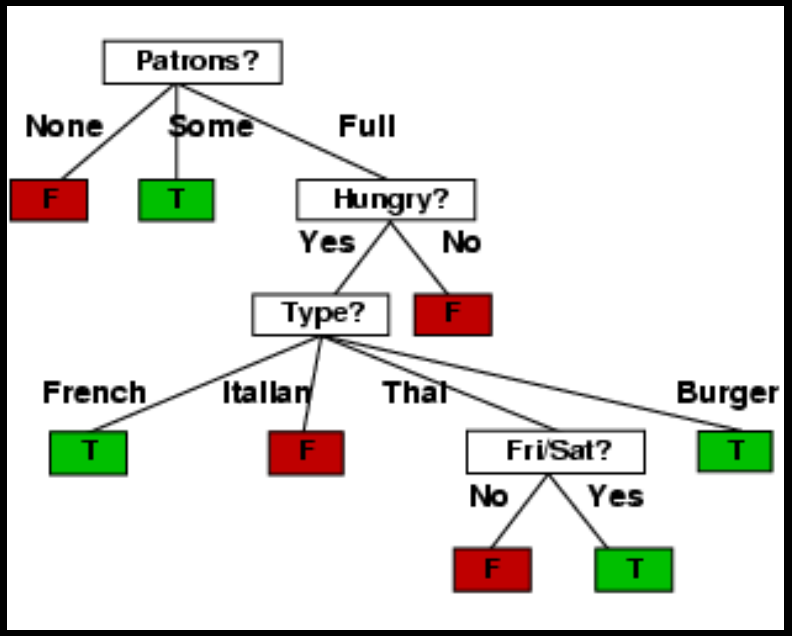
$$U_{i+1}(s) = R(s) + \gamma * \max_{a} \sum_{s'} P(s' \mid s,a)\, U_i(s')$$

- Policy iteration update:

$$U_{i+1}(s) = R(s) + \gamma * \sum_{s'} P(s' \mid s, \pi_i(s))\, U_i(s')$$

Learning

Supervised Learning: Decision Trees

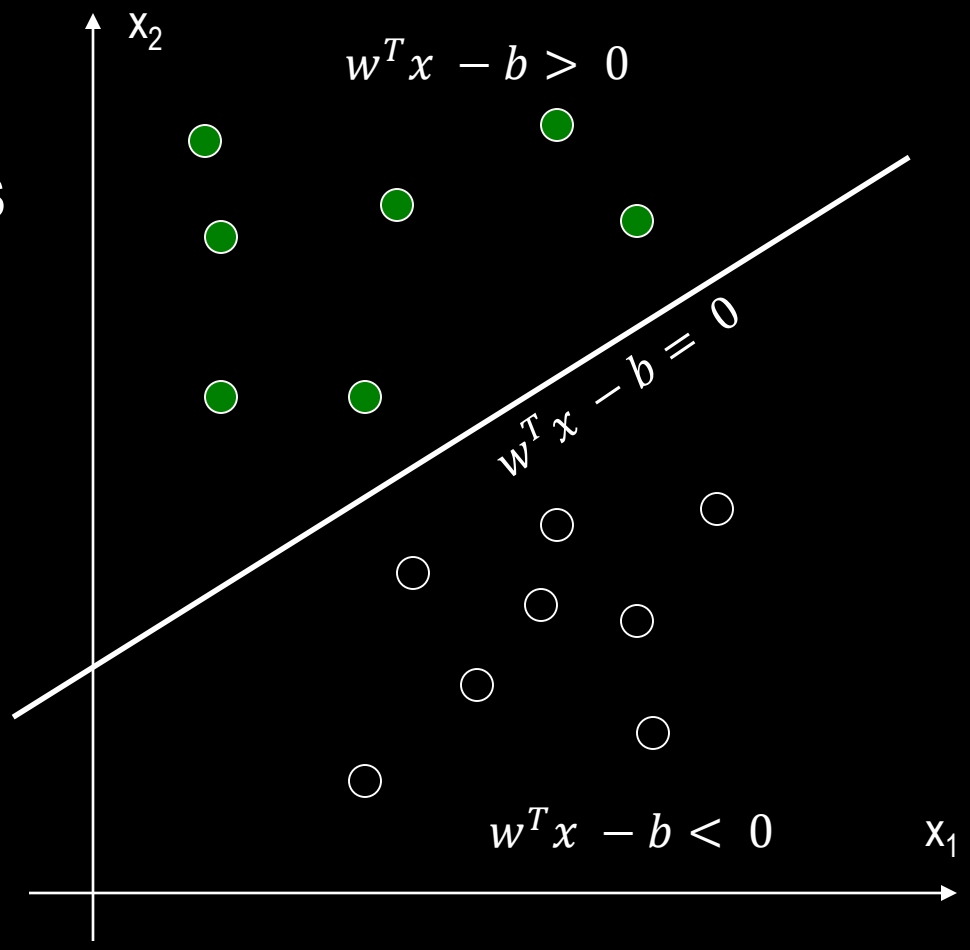


- Split on variables that give highest information gain at each level
  - Based on computing entropy of data for each branch

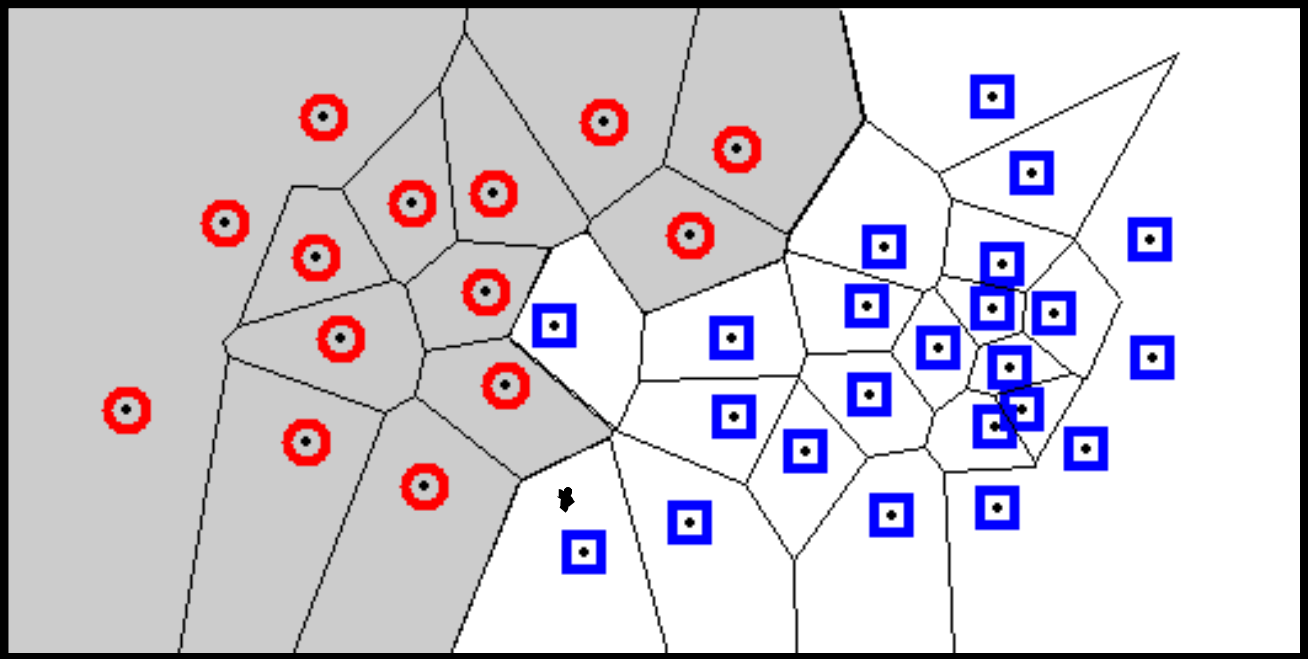
Support Vector Machines (SVM)

- A discriminant function f(x) is a function that separates two sets of data according to their labels
- SVM learns a linear function:

$$f(x) = w^T x - b$$



K-Nearest Neighbor



Decision Regions for 1-NN Classification

ANN: Backpropagation Algorithm

1. Initialize the weights to some random values (or 0)
2. For each sample (x_j, y_j) in the training set
   a. Calculate the current output of the node, h_{x_j}
   b. For each output node k, update the weights

$$\Delta_k = (h_{x_k})(1 - h_{x_k})(y_k - h_{x_k})$$

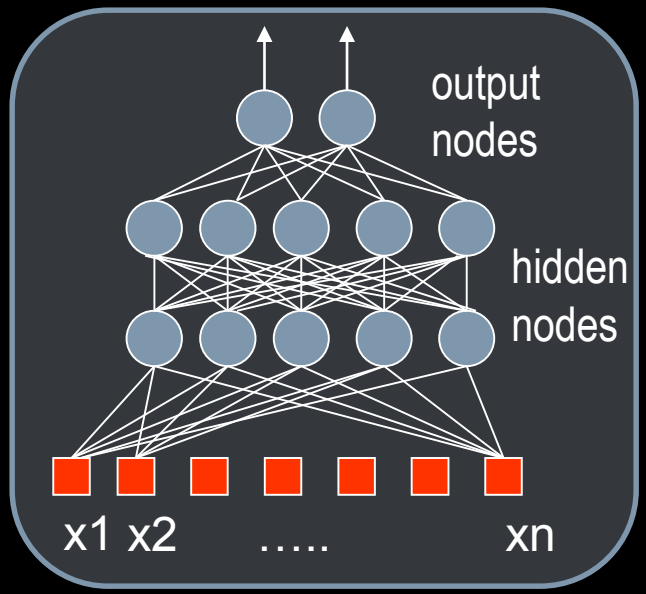
   c. For each hidden node j, update the weights

$$\Delta_j = (h_{x_j})(1 - h_{x_j}) \sum_k w_{j,k} \Delta_k$$

3. For all network weights do

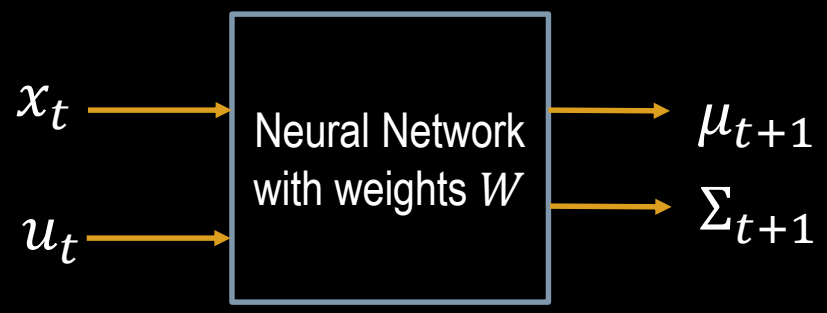
$$w_{i,j} = w_{i,j} + \alpha \Delta_j x_i$$

4. Repeat until weights converge or desired accuracy is achieved

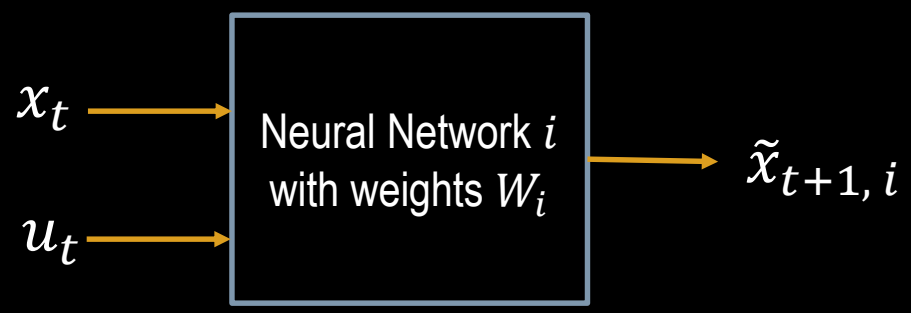


Uncertainty learning dynamics

- Predictive distributions for estimating *aleatoric* uncertainty



- Ensembles for estimating *epistemic* uncertainty



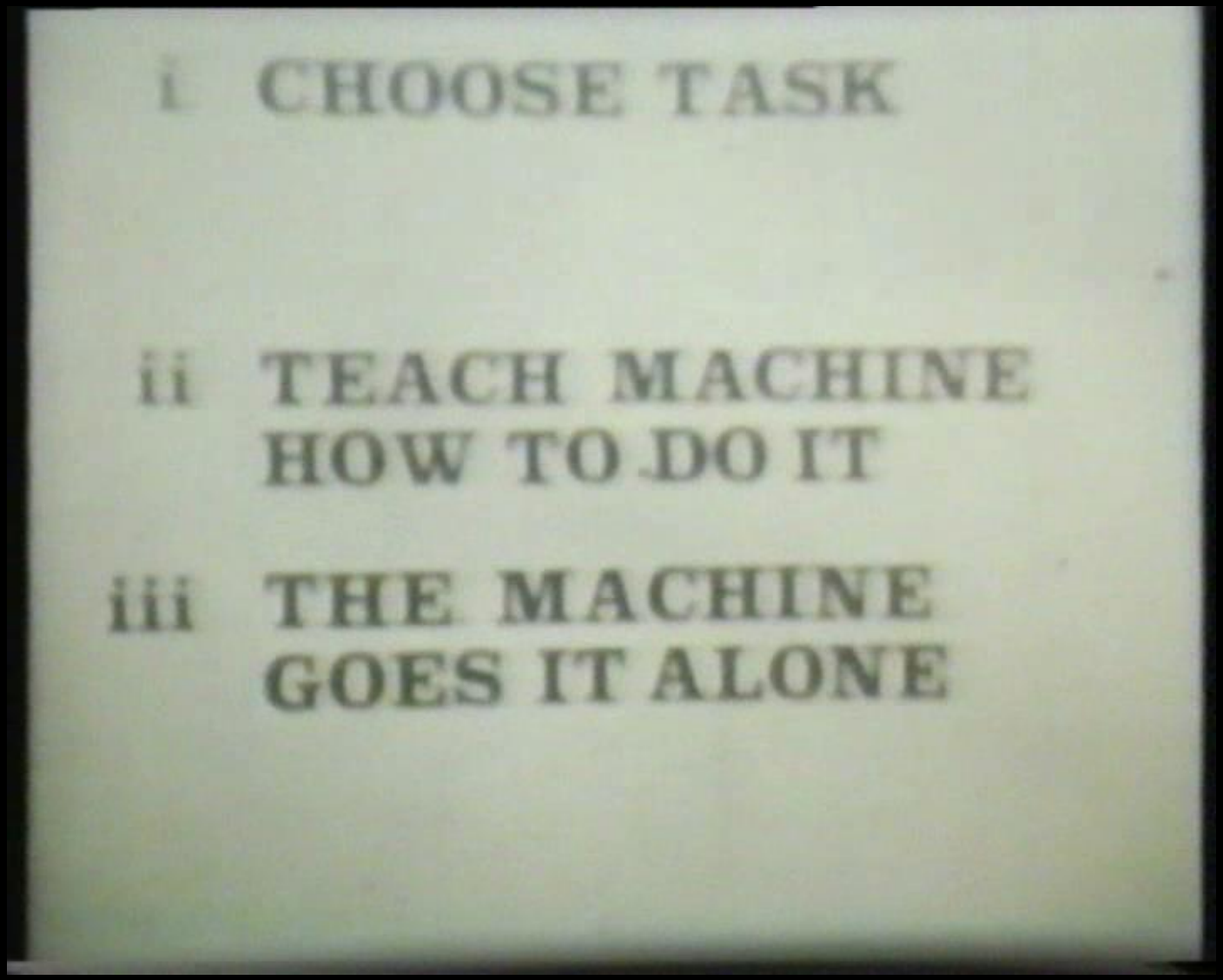
For $i = 1 \ldots K$

BREAK

Let’s play “How did they do that?”

Shakey, SRI, 1966-1972

Freddy, University of Edinburgh, 1973



Sensorless Parts Orienting, CMU, 1986

Navlab, CMU, 1986-2002

DARPA Urban Challenge, 2007

Team CMU

PR2 and ROS, Willow Garage, 2008-2013

KinectFusion, 2011

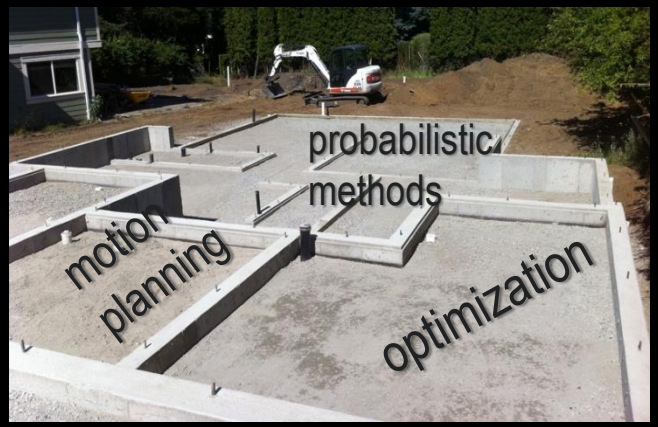
Kiva Robots, Amazon, 2014

DARPA Robotics Challenge, 2015

Team WPI-CMU

What we learned in this course

- Built a **foundation** for more advanced robotics courses/concepts



- The next step:

What to take next?

- Optimization (IOE 611)
- Artificial Intelligence (EECS 492/592)
- Motion Planning (EECS 598)
- Perception (vision in EECS 442/504/542, EECS 498: Self Driving Cars)
- Estimation and Mapping (EECS 467)
- Machine Learning (EECS 445)
- Kinematics/Dynamics (EECS 398)

- Consider getting involved in research!

Thank you for taking this course!