COMP 417 - Fall 2018 [Margaret Gu]

Motion-Planning Framework

Continuous representation (C space) > discretization > graph search

Grid-based planners

- PROS: works well for up to 3-4 dimensions
- CONS:
  - State-space discretization suffers from combinatorial explosion
    - If we split each dimension into N bins, there will be N nodes in the graph
    - Not good path planning for:
      - Multiple jointed robot arms
      - High dimension system

Path planning approaches:

- Roadmap
  - Represent the connectivity of free space by network of 1D curves
  - Visibility graph:
    - Can produce shortest paths in 2D C-space
    - Optimality for shortest path does not hold for higher dimensions
  - Voronoi diagram
    - Generate paths that maximizes clearance. Used in 2D c-space
  - Silhouette
    - First complete general method the applies to spaces of any dimensions and is singly exponential in # of dimensions
  - PRM (probabilistic roadmaps)
    - PROS:
    - Fixes RRT problem of single-query path planning
    - PRMs good for multi-query path planning; no need to re-plan for scratch
    - Each node is connected to its neighbours
- Cell decomposition:
  - Decompose he free space into simple cells and represent the connectivity of the free space y the adjacency graph of these cells
  - Exact cell decomposition:
    - Free space is represented by a collection of non-overlapping cells, whose union is exactly F
    - Trapezoidal and cylindrical decomposition
  - Approx cell decomposition
    - Represented by a collection of non-overlapping cells whose union is contained in F
- Potential field
  - Define a function over free space that has a global min at the foal config and follow steepest descent
- RRT:
  - Maintain a tree of reachable config from the root
  - Good for omnidirectional systems
  - Steps:
    - Sample random state
    - Find the closest state (node) already in the tree
    - Steer the closest node toward the random state
    - Uses brute force search

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

  - Uniform sampling CON: rotation components
    - Uniformly sample 3 Euler angles: nonuniform at North Pole caused by Gimbal lock: same rotation parameterized by different Euler angles
    - Uniformly sample a quaternion
    - Uniformly sample rotation matrixes.
  - Finding nearest neighbour uses brute force
    - Space partitioning
    - Locality-sensitive hashing
      - Maintain bucks, Similar points are placed on the same bucket
      - When searching consider only points that map to the same bucket
  - *RRT will eventually cover the space*
  - *Does not compute optimal path*
  - *Exhibit Voronoi bias: new node fall in free regions of Voronoi digram (cell consists of points that are closest to*

a node)
    - *Probability of RRT finding a path is higher with more iterations*
    - *Good for single query path (from A to B), but need to replan every time*

Open-loop
- Want to spin a motor at a given angular velocity. We can apply a fixed voltage to it and never check to see if it is rotating properly
- Called open loop control because there is no feedback and actual control
- If load on motor changes = output velocity changes and by feeding back some info

Control Theory
- Systematic approach to analysis and design
    - Transient response
    - Consider sampling times, control frequency
    - Taxonomy of basic controllers
- predict system response to some input (speed or oscillations)

Feedback system
- Power amp, actuator, feedback (sensor measurement), error signal and controller

Goals: regulation, tracking and optimization

Transfer function
- Relates output of a linear system to its input, how it responds to impulse

PID systematic bias
- Wheels are misaligned, car is too heavy on one side

when control variable remain at limit for long time, error increases

	Control type	Feedback	Pro/Con
Bang-bang	discrete	yes	Simple/ Discrete
Open loop	Control law	no	Simple/may be unrepeatable
Closed loop	P, I, D	yes	Continuous/ Tune Gains

When s PID insufficient>
- Large time delays
- Controller require completion time
- High-frequency oscillations
- High-frequency variations on the target state

Kinematics vs. Dynamics are physical models on how systems move (I.e what is the next state with current state)
- **Model** = function of a physical actual thing in the real world
- **Noise =** things that we don't bother modelling (ex. Friction)
- Reality, model mismatch = error/noise
- Kinematics considers models of locomotion independent of external forces and control
    - Ex. How speed of car affects the state without considering the required control commands required to generate those speeds
- Dynamics considers models of locomotion as function of control inputs and states

Dubins Vehicle: can only go forward at constant speed and you can control angular velocity
- Only left right and straight, where the instantaneous circle of rotation is the turning path
- Pitch angle determine descent rate
- Yaw angle determine turning rate

Holonomic contraint:
- Constraint on state of system
- to constrain state to line on circle (ex. Train tracks) x^2 -1 =0

Non-holonomic constraint: involves constraint on the derivatives of state (ex. Velocity)
- (cant move sideways directly Constraint n derivatives
- Dubbing car: car is constrained to move along the line of current heading

Unicycle: would u put radius to be part of a state ? No. If we can measure beforehand then no.

ControllabilityL bring system from any state to another in finite time with some controls sequence

Passive dynamics: power system not from a power supply, propelled by own weight

CCD (charge-coupled device) imaging sensors:
- Accumulates electric charge proportional to light intensity
- Each capacitor's charge is transferred to its neighbour
- Last capacitor gets amplified and output as voltage
- High quality, low noise
- CON: higher power consumption, low readout, specialized fabrication

CMOS imaging sensors:

- One amplifier per pixel
- PRO: low power, fast readout, easier to fabricate cheaper
- CON: higher noise, bad for low-light sensitivity

Gyroscopes
- A shining wheel with its mass concentrated on the outside, so force is needed to rotate
- Mesure rate and direction of rotation
- Measure angular velocity: w = w + bias + noise

Global shutter: captures all pixels (faster shutter)
Rolling shutter: does not capture all pixels
Shutter = allow light to hit imaging sensor
Shutter speed = exposure time

Camera lenses
- Lens determine: image distortion, focus and sharpness
- Lens characteristics: focal length, aperture, depth of field

Pinhole Camera Model
- Approx how 3D point projects to pixel

1. Perspective projection: [x,y] = pi(X,Y,Z): we lose depth
2. Lens distortion: [x*,y*]=D(x,y)
3. Estimating parameters of lens distortion: [x*,y*]=D(x,y)

Non-pinhole cameras with thin lens can model blur
Infrared camera: cant use underwater

DC (direct current) motor
They turn continuously at high RPM (revolutions per minute) when voltage is applied. Used in quadrotors and planes, model cars etc.

Servo motor
Usually includes: DC motor, gears, control circuit, position feedback
Precise control without free rotation (e.g. robot arms, boat rudders)
Limited turning range: 180 degrees

Stepper motor
Positioning feedback and no positioning errors.
Rotates by a predefined step angle.
Requires external control circuit.
Precise control without free rotation.
Constant holding torque without powering the motor (good for robot arms or weight-carrying systems).

RGBD camera
- Active sensing
- Projector emits infrared light in the scene
- Infrared sensor reads the infrared light
- Deformation of the expected pattern allows computation of the depth
- CONS: bad for outdoors, range is limited
- PROS: cheap and real-time depth, good for skeleton tracking, commonly used in robotics

2D LIDAR
- Produces a scan of 2D points and intensities
- Point is for laser reference
- Intensity is related to the material and how it reflects light
- LIDAR BAD FOR GLASS
- Motors move the laser beam and then return scan (around 1000 pts)

3D LIDAR
- Produce point cloud with 3D points and intensities
- Time of light of laser: time it takes for the beam to get back to scanner
- 2D BAD FOR POOR WEATHER
- Used in self-driving cars for obstacle-detection

RADAR
- Use electromagnetic energy
- May use time-of-flight with sonar and lidar for phase detection and frequency modulation
- PRO: good in bad weather

IMU - Inertial Measurement Unit
- Combines Gyroscopes, accelerometers, magnetometers to *estimate orientation with reduced drift*
- IMU necessary for autopilot
- Good for real time
- **Magnetometers**
  - Pros: as a compass for absolute heading
  - Cons: careful calibration, place away from moving parts
- **Accelerometers**
  - Measure linear acceleration relative to freefall (measured in g)
  - Affected by noise and bias, gravity director, IMU body frame, fixed world frame
  - CONS: noisy to get position bc of double integration, errors grow quadratically

GPS
- Satellites transmits coarse, navigation frame(PRN), precision code
- Receivers: know PRN code of satellite (status), takes 4+ satellites and compute latitude and longitude

Hall effect sensor:

- Measures variations in voltage response to magnetic field
- Measure rate of rotation of wheels

Rotary Encoder
- Convert angle of motor
- Good to know where different shafts are relative to each other

Pulse Width Modulation: creating continuer behaviour when applying discrete amounts of voltage
- Turn motor on/off until average voltage is at target (ex. Dimming LEDs)

Occupancy grid: Cell is either explored or unexplored, shows probability of occupation
- **Pros:** O(1) lookup and delete, supports image operations
- **Cons:** doesn't scale well in higher dimensions

Quad tree: each node represents a square. If the node is fully empty of fully occupied = it has no children
- Partially occupied = 4 children
- Division eventual stops on some min square size

Octrees
- Partial occupied = 8 children
- **CONS: not balanced trees, worst case O(n) and sensitive to small changes in location of obstacles**

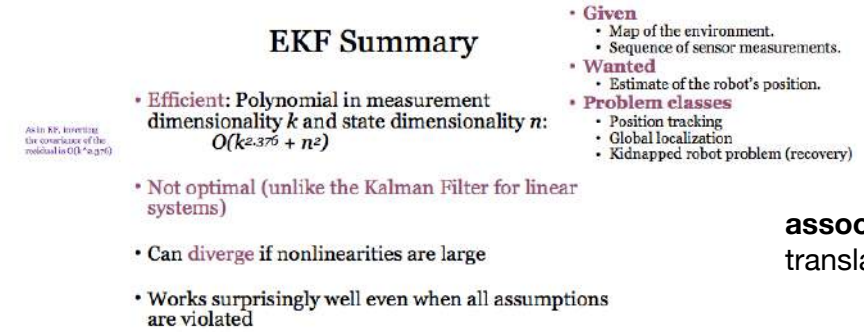
Signed distance function
- Defined over any point in 3D space

Pointclouds
- Pros: can make local changes without affecting the whole thing, can align clouds, nearest neighbour queries are easy with locality-sensitive hashing

Topometric maps
- Pros: allows us to combine accurate local maps into a global, inconsistent maps that just provides us enough navigation info
  - Edges represent rotations, translations and topological connectivity of local maps

Global location problem:
- use a decision tree and want to minimize height help of optimally decide what to do in ambiguous situation
- ***Determining pose using a min length path through a known map***
- MDL: NP-hard problem with abstract decision tree
  - Greedy algorithm, visit neighbours to know where I am (will get exponentially worse)
  - **Overlap polygon:** For x hypothetical locations, make x copies of the environment.
    - Put copies onto of each other and hypothetical locations will coincide
- Algorithm: NO BETTER ALGORITHM
  - Computer hypothetical locations
  - Compute overlay polygon
  - Visibility cell decomposition
  - Determine pts that might discriminate hypothetical locations
  - Visit closest pt
  - Go back to start
- **CONS**: high computation cost for solution, selected observations may not be practical (cant reach small cells)
- Sampling approach: compute intersection of overlay, and select based on info and distance
  - Better average performance
- Improve cost with Monte Carlo selection of point

Metric Map Alignment = ICP = scan matching = registration
- given 2 maps/point clouds .... How do we find the rotation and translation to align them
- Scan alignment with known correspondences
  - If the correct correspondences are known (**data association is known)**, the correct relative rotation/ translation can be calculated in closed form.



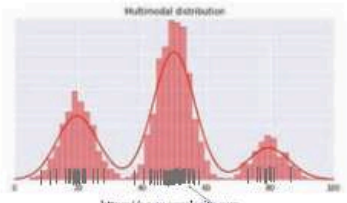
Find the 3D rotation matrix R and the 3D translation vector t that will best align the corresponding points

$$\text{error}(R,t) = \frac{1}{N}\sum_{i=1}^{N} ||p_i - (Rq_i + t)||^2$$

$$R^*, t^* = \underset{R,t}{\text{argmin}}\ \text{error}(R,t)$$

Q: How do we minimize this error?
A: Turns out it has a closed-form solution.

How can we represent multimodal distributions?



https://www.opsclarity.com

Higher density of particles means higher probability mass

Idea #1: Histograms
Advantages: the higher the number of bars the better the approximation is

Disadvantages: exponential dependence on number of dimensions

Note: this approach is called the Histogram Filter. It is useful for low-dimensional systems.

Idea #2: Function Approximation
Unclear how to do Bayes' filter updates and predictions in this case, but a mixture of Gaussians is an option.

Idea #3: Weighted Particles $\{(x^{[1]}, w^{[1]}), \ldots, (x^{[M]}, w^{[M]})\}$
Advantages: easy to predict/update by treating each particle as a separate hypothesis whose weight is updated.

Disadvantages: need enough particles to "cover" the distribution

$$bel(x_t) = p(x_t|z_{0:t}, u_{0:t-1}) = \sum_{m=1}^{M} \begin{cases} w^{[m]}/W & \text{if } x_t = x_t^{[m]} \\ 0 & \text{o.w.} \end{cases}$$

W: sum of all particles' weights

Particle Filter Algorithm

ParticleFilter($\bar{z}_t, u_{t-1}$)

$\bar{S}_t = \{\}$ $\bar{W}_t = \{\}$

for particle index $m = 1 \ldots M$

sample $x_t^{[m]} \sim p(x_t|x_{t-1}^{[m]}, u_{t-1})$

$w_t^{[m]} = p(\bar{z}_t|x_t^{[m]})$

$\bar{S}_t$.append($x_t^{[m]}$)

$\bar{W}_t$.append($w_t^{[m]}$)

Particle propagation/prediction: noise needs to be added in order to make particles differentiate from each other.

If propagation is deterministic then particles are going to collapse to a single particle after a few resampling steps.

$S_t = \{\}$

for particle index $m = 1 \ldots M$

sample particle i from $\bar{S}_t$ with probability $\propto w_t^{[i]}$

S_t.append($x_t^{[m]}$)

return S_t

Particle filter: update particle weights and distribution of particles (resampling)
Resampling: sample particles with repetition/replacement according to their updated weights

- Get rid of low weight particles

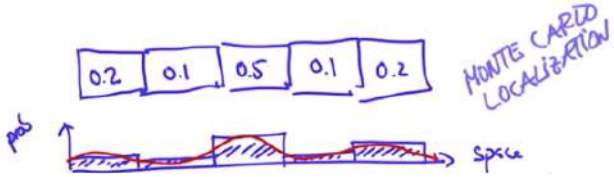
LOCALIZATION, FILTERS, CONTROLLERS [Margaret Gu]

Tracking methods: Kalman and Monte Carlo localization

- Kalman Filter: technique to estimate the state of a system
  - Estimates future locations and velocities based on data (irregardless of noise)
- Difference between Kalman and Monte
  - **Kalman** estimates a *continuous* state
    - Uni-modal distributions
  - Monte, we are forced to chop the world into discrete places (*discrete)*
    - Multi-modal distributions
- Particle filters: continuous and multimodal

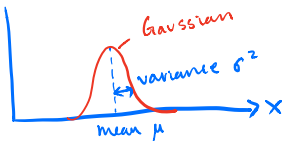
Markov model: divides the word into discrete grids

- Then assign each grid a probability
- **Histogram**: a representation of probability over spaces (like Markov)
  - Divides continuous space into finite many grid cells



In Kalman, distribution is given by **Gaussian**

- Gaussian: continuous function over the space of locations and area underneath sums to 1
  - If the space is x (x-axis), then the Gaussian is characterized by 2 parameters
    - mean (μ)
    - Width of Gaussian - variance(σ^2)
- Gaussian in 1D (parameter space is 1d)
  - Is characterized by (μ) and (σ^2)
  - Rather than estimating entire distribution of a histogram, Kalman filter is to maintain a μ and σ^2 that is our best estimate of the location of object we're trying to find



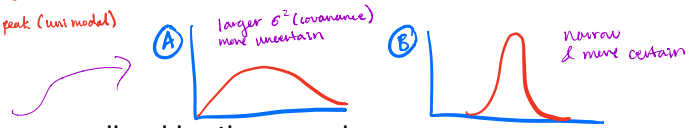
1D Gaussian (μ, σ^2)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

$(x-\mu)^2$: quadratic difference of query pt x relative to mean μ

★ if $x=\mu$, $\exp^0 = 1$

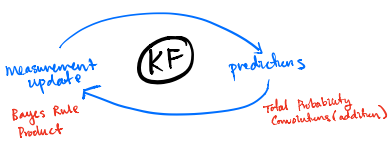
★ Gaussian all have same structure of one peak (uni modal)

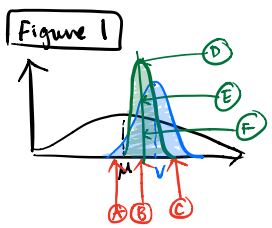


- Wide gaussian have higher variance than narrow ones
  - Why? Look at function: difference between x and y is normalized by the covariance
    - Larger the (x-u), the less the difference matters and the function is more spread out
  - Covariance is also a measure of uncertainty
    - Larger the covariance, more uncertain of actual state
- To maximize gaussian is to set x = u (I.e. peak)
- Gaussian Recap
  - Definition: f(x)
  - Unimodal and symmetrical

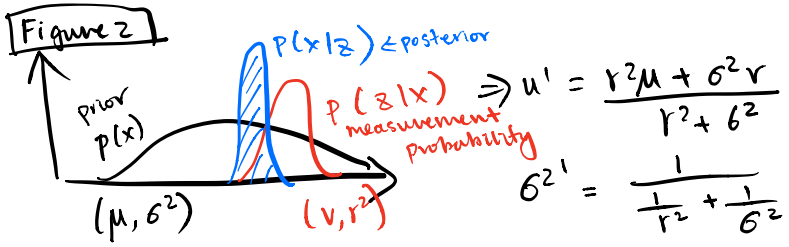
Kalman Filters(KF): represents all distributions but Gaussian

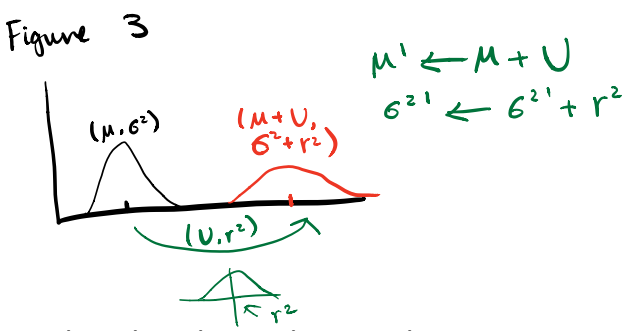
- KF iterates 2 things: measurement and motion updates
  - Measurements use products and Bayes Rule
  - Motion uses convolutions and Total Probability





- **Measurement Update**
  - Figure 1 - Example 1: suppose you're localizing another vehicle and you have a prior distribution (black with u)
    - We get a measurement (blue with v) that tells us sth about the localization of vehicle
    - Where will be the new mean? A, B or C?
      - B: between the 2 means, and will be slightly more on measurement side bc measurement was more certain (due to Kalman gain)
  - Example 2: where will be the correct posterior after the multiplication of the 2 Gaussians?
    - D: b/c more measurements gives us more certainty
      - *The new belief will be more certain either the previous belief OR the measurement*
    - Proof: Suppose we multiple 2 Gaussians as in Bayes rule: (1) prior and (2) measurement probability
      - Prior: (u, sigma^2), measurement: (v, r^2)
      - New mean (u') is the weighted sum of the old means
        - (Where u is weighted by r^2// v is weighted by sigma^2) normalized by the sum of the weight factors
      - New variance (sigma^2')
      - Prior is more uncertain with higher variance, so v is weighted more than u, so mean will be closer to v than u

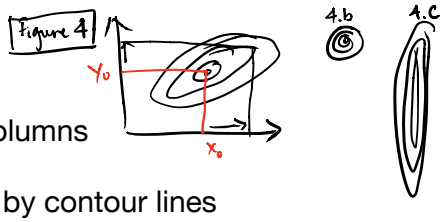




- **Motion Update**
  - Prediction, Total Probability and Addition
  - Suppose you live in a world like Figure 3, you have a current best location and uncertainty
  - If you move to the right of a certain distance, motion itself has an uncertainty (r^2)
  - Then you arrive at the prediction where you add motion to mean, and *will increase uncertainty over the initial*
    - because you have a prediction from your prior but increases uncertainty due to motion uncertainty and lack of information
  - Math: new mean(u') = old mean(u) + motion(U)
  - new variance = old variance + variance of the motion Gaussian

2D Kalman

- Suppose you have 2D space of (x,y), like camera image, or car using radar to detect location of car over time
- Kalman is able to make predictions of future locations by using velocity



High Dimension/Multivariate Gaussian Figure 4

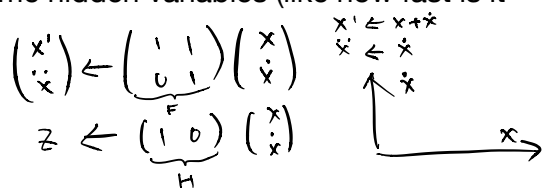
- The means(u) is a vector with 1 element for each of the dimensions
- The variance is replaced by covariance (sigma): matrix with D rows and columns
- In a 2D Gaussian, you can draw its contour lines
- Mean is (x0,y0) and covariance defines the spread of Gaussian, indicated by contour lines
- Gaussian with smaller uncertainty might look like Figure 4.b
- You can also have a small uncertainty in one dim, but huge in another dim (Figure 4.c)
- **Know location is correlated to velocity**; move faster = move further to the right (expressed by red Gaussian)

Kalman Review

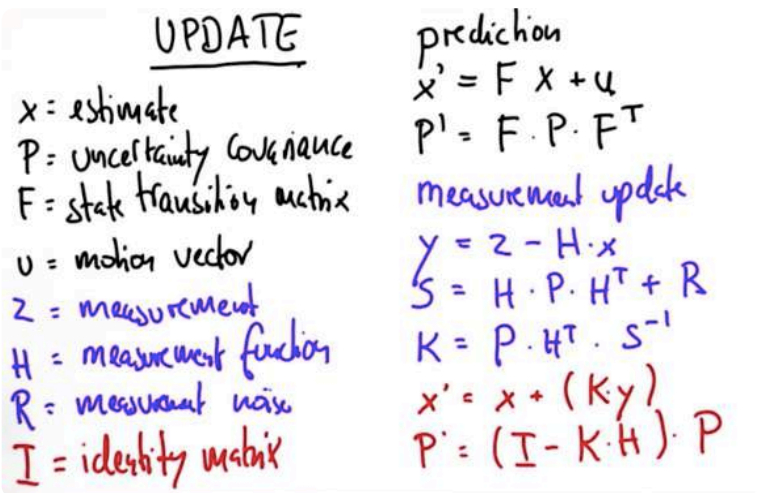
- The variable of a Kalman filter are called **states**
  - Separate into 2 subsets: observables(momentary location) and hidden (determined by velocity)
- The subsets interact; from multiple observations we can estimate some hidden variables (like how fast is it moving)

Design FK

- Need a state transition function (matrix) and measurement function
- New location(x') = old location(x) + velocity ($\dot{x}$)



- **Update**
  - There's a prediction step where I take best estimate x, multiply with state transition matrix F and add whatever motion (U) I know
    - **Prediction:** x'= Fx + u
  - I also have a covariance P that characterizes my uncertainty, and update as follows
  - There's a **measurement update** step (z)
    - Compare measurements with predictions
    - H is the measurement function that has the state of measurements
    - Y is the error
    - Error is mapped into a matrix S(obtained by projecting the system uncertainty into the measurement space using the H, + matrix R (measurement noise)
    - Then mapped into variable K (**Kalman gain)**
    - **Finally update estimate and uncertainty**



	Kalman Filter	Extended Kalman Filter	Particle Filter
Dynamics model	Linear	Nonlinear	Nonlinear
Sensor model	Linear	Nonlinear	Nonlinear
Noise	Gaussian (Unimodal)	Gaussian (Unimodal)	Multimodal

Pros and Cons of Filters

Histogram filters are at a big disadvantage because it scales exponentially bc any grid that is defined over k dimensions would have exp many grid cells....not good for high dim

- Approximate bc world is discrete

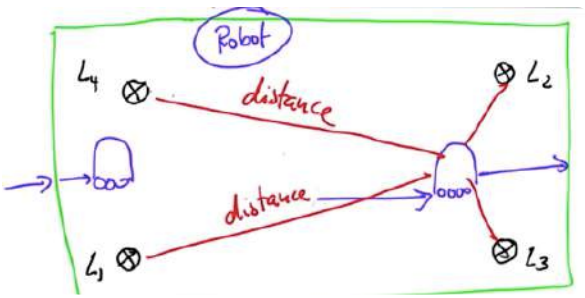
Kalman filters are better for higher dimensional spaces

- Approximate but only exact for linear systems
  - But world is nonlinear

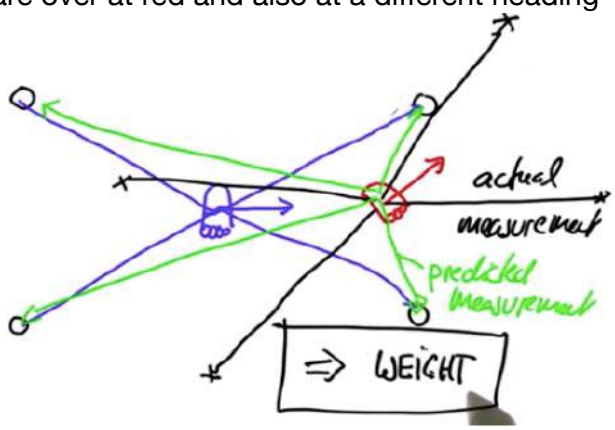
QUIZ	state space	belief	efficiency	in robotics
Class 1 Histogram Filters	✗ Discrete ○ Continuous	○ unimodal ✗ multimodal	○ quadratic ✗ exponential	○ exact ✗ approximate
Class 2 Kalman Filters	○ Discrete ✗ Continuous	✗ unimodal ○ multimodal	✗ quadratic ○ exponential	○ exact ✗ approximate
Class 3 Particle Filters	Continuous	multimodal	?	approximate

Particle filters uses a shit ton of dots as a discrete guess where the robot might be

- Structured as an X and Y coord, and a heading direction (these 3 values = 1 guess)
- Filter = shit ton of guesses
- Using particle filters finds options that survive; *particles more consistent with measurements are more likely to survive*
  - So places with high probability will collect more particles and be more representative of the robot's posterior belief
- Example robot
  - Can turn and move straight after turn, and can sense distance to 4 designated landmarks (L1, L2, L3, L4)
  - The distances comprise the measurement vector of the robot
  - Robot exists in a cyclic world



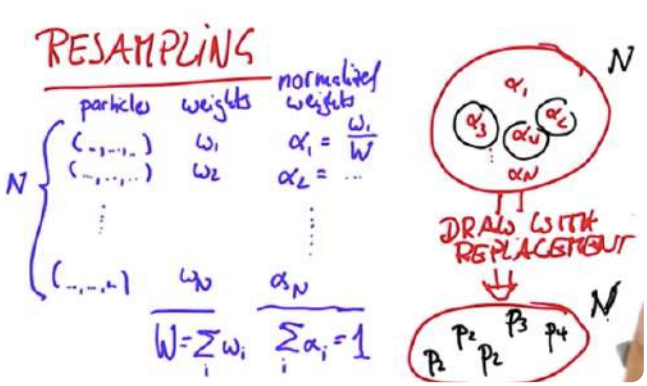
- Each dot/guess is a vector that contains an x-coord, y-coord, and a heading direction(angel at which the robot points relative to x-axis)
- Suppose robot sits where it measures exact distance to the landmarks
  - There will be some measurement noise, as add Gaussian with 0 mean
    - 0 mean = chance of being too short or too long and probability is governed by Gaussian
  - This process gives us a measurement vector of 4 values (distance to landmarks)
  - Now consider a particle that guesses the robot coordinates are over at red and also at a different heading direction
    - We can take the measurement vector (black lines) and apply to the new particle (red)
      - it should've looked like green lines, so it makes the specific location of that particle(red) really likely
  - *The closer our particle to the correct position, more likely will be the seat of measurements given that position*
  - The mismatch of actual and predicted measurement leads to **importance weight** (how important is the particle)...larger weight = more important



- With many particles and a specific measurement, each particle will have a different weight
  - The probability of survival of particle is proportional to their weight
  - So after **resampling** (randomly drawing N new particles form old ones with replacement in proportion tot he importance weight)
  - So we only need to implement 2 methods: (1) setting importance weights and (2) resampling

Resampling definition

- Given N particles (each with 3 values: X, Y, heading direction) with weights
- Resampling puts all the particles and normalized weights into a big bag and it draws with replacement N new particles by picking each particle with probability alpha
  - We can draw multiple copies of the same articles
  - In the end, the particles with high normalized weights will occur more frequently in the new set
- HOWEVER, a particle with high weight can NOT be resampled
- Does orientation matter? Yes in the second part of particle filtering (resampling)



Review:

- Measurement update
  - Computed posterior over state given the measurement.
  - It is proportional, after normalization, of probability of the measurement given the state*P of the state itself
- Motion update
  - Compute a posterior of the distribution one time step later and that the convolution of the transition probability * my prior

measurement updates: $P(x|z) \propto P(z|x)\,P(x)$

These particles represent correct posterior — importance resample weights — set of particles (prior)

motion updates:

correct distribution after robot motion $\Rightarrow P(x') = \Sigma\, P(x'|x)\,P(x)$

samples — sample — particles

MOTION EXPLANATION:

By taking a random particle from P(X) and applying the motion model with the noise model, to generate a random particle (X'). As a result, you get a new particle set P(X') that is the correct distribution after the robot motion.

Lecture Review:

Kalman Filter:

- Linear dynamics and observation models
- Initial belief = Gaussian
- Noise variables and initial state = Gaussian and independent
- Noise variables are independent and identically distributed
- Kalman gain: how much effect will the measurement have int he posterior compared to the prediction prior
  - posterior will lean towards the mean that is more certain
- Uncertainty increases after prediction step
- New observation reduce uncertainty in posterior
- **Kalman is good to compute posterior**

Bayesian Filter: requirements for implementation

- Update equations
- Motion model = Gaussian
- Sensor model = Gaussian
- Representation for the belief function
- Initial belief state

Limitations

- Unimodal distribution
- Linear assumptions
  - Mobile robot dynamics are not linear
  - Linearization will increase state error residual
    - Can be improved with EKF

The larger the error, the smaller the effect on the final state estimate

- If process uncertainty is larger, sensor updates will dominate state updates
- If sensor uncertainty is larger, process propagation will dominate state estimate

EKF Summary

- Efficient
- Not optimal
- Can diverge if nonlinearities are large

Kalman Filter Components

(also known as: Way Too Many Variables...)

Linear discrete time dynamic system (motion model)

$$x_{t+1} = F_t x_t + B_t u_t + G_t w_t$$

State transition function — Control input function — Noise input function with covariance Q

Measurement equation (sensor model)

$$z_{t+1} = H_{t+1} x_{t+1} + n_{t+1}$$

Sensor reading — State — Sensor noise with covariance R — Sensor function

Note:Write these down & remember them!!!

Assumptions guarantee that if the prior belief before the prediction step is Gaussian

$$\begin{aligned} bel(x_t) &= p(x_t|u_{0:t-1}, z_{0:t}) \\ &= \eta\, p(z_t|x_t) \int p(x_t|u_{t-1}, x_{t-1})\, bel(x_{t-1})\, dx_{t-1} \end{aligned}$$

then the prior belief after the prediction step will be Gaussian

and the posterior belief (after the update step) will be Gaussian.

Linear observations with Gaussian noise

$$z_t = Hx_t + u_t \quad \text{with noise } n_t \sim \mathcal{N}(0, R)$$

Linear dynamics with Gaussian noise

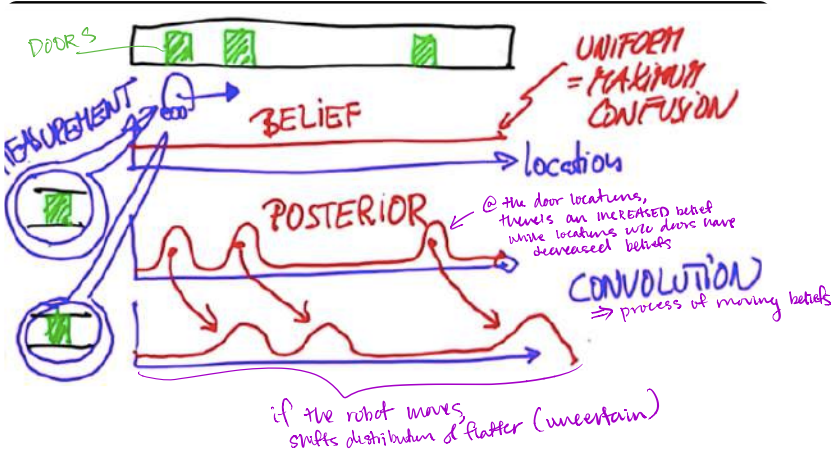
$$x_t = Ax_{t-1} + Bu_{t-1} + Gw_{t-1} \quad \text{with noise } w_{t-1} \sim \mathcal{N}(0, Q)$$

Initial belief is Gaussian

$$bel(x_0) \sim \mathcal{N}(\mu_0, \Sigma_0)$$

Localization:

- Involves a robot lost in space and model it with a function
  - Y-axis = probability for any location
  - X-axis = all places in the 1D world
  - Now we model the robot's current belief about where it might be, its confusion is a uniform function that assigns equal weight to every possible place in the world = **state of max confusion**



Product distributions must all sum to 1

P(Xi | Z)

- Posterior distribution of place Xi given measurement Z

Localization summary:

- Belief = probability
- Sense/update function = product followed by normalization
- Move = convolution

Bayes' Rule

- $P(A|B) = P(B|A)*P(A) / P(B)$
- Suppose X is my grid cell and Z is my measurement
- Then the measurement update seeks to calculate a belief over my location after seeing the measurement [p(X|Z)]
  - It take my prior distribution P(X) and multiplies in the chance to see a red or green tile for every possible location

Motion Planning

Planning: process of finding a path from a start location to a goal location (robot motion planning)

Planning Problem:

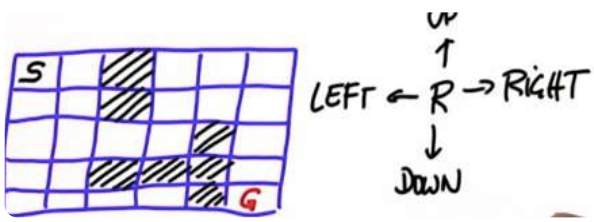
- Given:
  - a map of the world
  - Starting location
  - Goal location
  - Cost function (time it takes to drive a certain route)
- Goal:
  - Find min cost path m

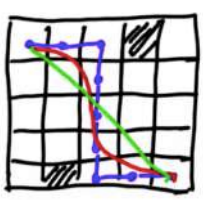
Optimal Path Finding:

- What if we punish certain actions (actions that are more cost expensive)

Search program: path planning

- Ex. Given grid with start and goal states, and 4 plausible actions
- More in audacity
  - A*, dynamic programming





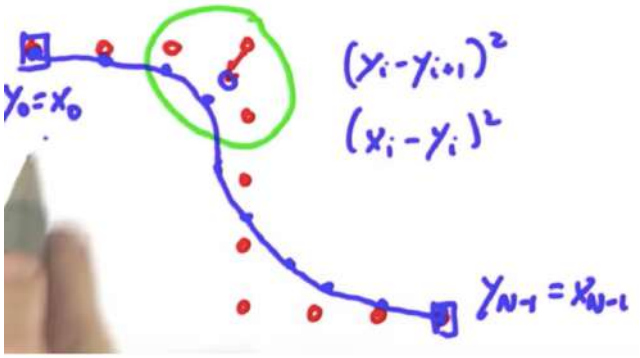
Robot Motion

- How to turn the found paths into actual motion commands
  - Generating smooth paths
  - PID Control

Generating Smooth Paths:

- Blue: expected; shortest: green; most optimal: red
  - Because of turning angles
- Smoothing Algorithm
  - Paths are sequences of points in a 2D grid
  - Call each point xi; each x is a 2D coordinate, *immaterial to the smoothing (can do it in multiple dims)*
  - Sequence goes from x0 to x(n-1)
  - **Actual smoothing algorithm**:
    - Initially create variable yi that are the same as all the xi's
      - Yi = xi
    - Optimize 2 criteria:
      - (1)Minimize the error of the its original point with the its smooth point $(x_i - y_i)^2 \rightarrow \min$
      - (2)Minimize the distance between consecutive smooth points $+\alpha (y_i - y_{i+1})^2 \rightarrow \min$
    - **If we only optimize criteria(1): you get the original path (minimization has no effect)**

- **If we only optimize criteria (2): you get no path**
  - (2) criteria ask that all y's are similar as possible
    - So if its minimized, then all the y's are all the same, which means you only get a single point and no path
- So these 2 criteria are in conflict with each other, so you do them by **weight**
  - The stronger the alpha, the smoother the path; smaller the alpha, the more we retain original points
- Ex. Suppose you were give a solution to the planning problem (red dots) and run optimization algorithm
- Consider place (green)
  - Shift corner red point into that direction, we can decrease the second error term (2)
    - Decrease both for A & B and B & C
  - But, we do this at the expense of the first error term, since we're not shif
  - ting the point away from the original x.
  - Depending on the weight of these error terms, we might arrive with as the blue dotted path
    - The new path (blue) suffers an error of the first type that we moved the points away from the original points, *but it drastically reduces the inter-point distance as in the error term (2)*
  - If the original points are not changed, take the out of the optimization algorithm
    - y0=x0 and yn-1 = xn-1



How do we optimize the 2 terms?

- Use **gradient descent**
  - for every time step, we take a small step in the direction of minimizing the (1) error
  - New expression: $y_i = y_i + \overset{0.5}{\alpha}(x_i - y_i)$
    - When we iterate, we assigned to yi recursively the old yi, but we subtract a term that's proportional to the deviation of yi to xi, weighted by a weight function alpha
  - For the second new expression: $y_i = y_i + \beta(y_{i+1} - y_i)$
    - We retain the old y variable, but e move a little bit in the direction of yi+1 and away from yi
  - An expression of an even better implementation: $y_i = y_i \overset{0.1}{-} \beta(y_{i+1} + y_{i-1} - 2y_i)$
    - Realizes that each yi occurs twice in this optimization term
    - SO we wish yi to be as close to yi-1 and simultaneously be as close as yi+1

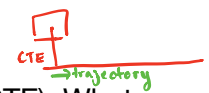
PID Control

Given scenario of a car with a steerable front axle and 2 non-steerable wheels in the back. We wish the car drove along a line. Assume car has fixed forward velocity and you are able to change the **steering angle** of the car. How do you do this? Set steering angle in proportion to **cross track error** (material distance between the vehicle and reference trajectory).

- The larger the error, the more willing to turn towards the target trajectory
- As you get closer to trajectory, steering will be slower and slower and eventually reach trajectory

P-Controller (P = proportional)

$\alpha = \tau \cdot CTE$

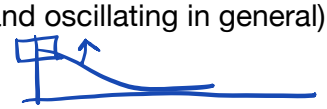


- Scenario: steer in proportion to **CTE** (I.e. steering angle is proportional by some factor of tau to the CTE). What will happen to the car? It will **overshoot**.
  - When car hits the trajectory, its wheels will be straight, but the robot itself will be oriented a bit downwards, so will be forced to overshoot
  - So the P- controller will act like figure 1
    - Slightly overshoot (which is ok) (I.e. parts below x-axis)
    - And will never converge
    - So it is **marginally stable**

Figure 1 :

PD- Control (to avoid overshooting and oscillating in general)

- $\alpha = -\tau_p CTE - \tau_d \frac{d}{dt} CTE$



$\frac{d}{dt} CTE = \frac{CTE_t - CTE_{t-1}}{\Delta t}$

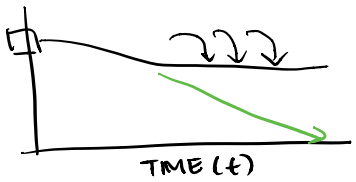
Equation means that when the car has turned enough to reduce the CTE, it won't overshoot the x -axis.

- It will notice that the error is getting smaller over time and will **counter** steer by steering up

Systematic Bias

- You assume the front wheels of your car are 100% aligned
- Bias becomes an increased cross track error in the y direction.
- Can the D-term (differential) solve this problem? NO

- So due to the bias, you steer more and more over time to compensate for this bias
- So you steer the car in the green line direction
- We need a sustained situation of large error
  - Measured by the integral/sum of the CTE over time $\sum_T CTE$



Lets make a new controller:

$$\alpha = -J_P\,CTE - J_D\,\frac{d}{dt}CTE - J_I \sum CTE$$

P proportional, D differential, I integral } PID

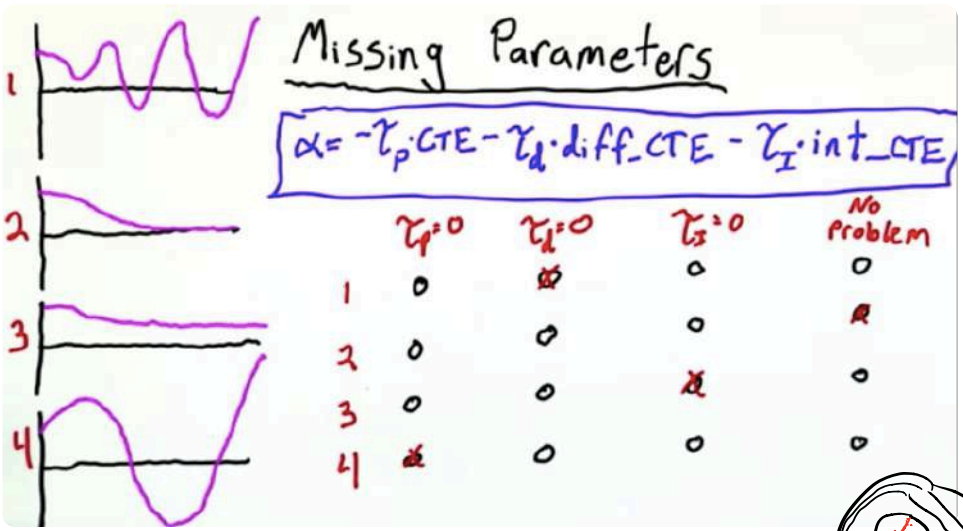
- Steering is proportional to the CTE, differential CTE and the integral/sum of all CTEs

How can we find good **control gains**? → J_P, J_D, J_I

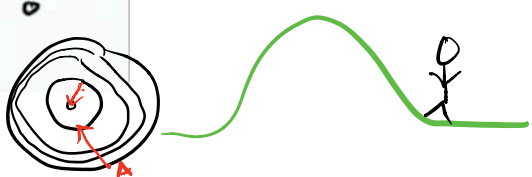
- Use Twiddle, coordinate descent

Twiddle

- Purpose is to optimize for a set of parameters
  - We need a function to return goodness (ex. Average CTE)
  - We want to minimize the average CTE




```
TWIDDLE                    run() → goodness
p = [0, 0, 0]              if err < best_err
dp = [1, 1, 1]                 best_err = err
best_err = run(p)              dp[i] *= 1.1
while sum(dp) > 0.00001    else
  for i in range(3):           p[i] -= 2*dp[i]
    p[i] += dp[i]              if err < best_err:
    err = run(p)                   best_err = err
                                   dp[i] *= 1.1
                               else:
                                   p[i] += dp[i]
                                   dp[i] *= 0.9
```



Gradient Descent

What is a gradient?

- Image scenario of a blind hill climber, he wants to get to the top of hill in fewest steps possible. What is his method? As long as the hill is steep, he knows he can afford to take a really big step bc he is not close to the top
  - As the kill smoothes out and becomes more level, he will take smaller steps bc top is near and doesn't want to overshoot.
  - Look at hill from a top down view of isoclines, each line representing the same altitude. Closer around the outside edges and closer towards the inside (bc the hill is getting less and less steps).
  - At any point, the gradient is a vector.
    - Ex. From pt A, we will get a vector pointing in the direction of steepest descent. Length of vector depends on how steep the hill is.
- Gradient descent is like climbing into a valley and finding the **bottom** bc we want to **minimize** a function.

(1) $b = a - \gamma \nabla F(a)$

(2) $F(y_i) = \frac{\alpha}{2}(x_i - y_i)^2 + \frac{\beta}{2}(y_i - y_{i+1})^2 + \beta(y_i - y_{i-1})^2$

min w/ gradient descent

(3) $y_i' = y_i + \alpha(x_i - y_i) + \beta(y_{i+1} + y_{i-1} - 2y_i)$

(y_i': new location; y_i: old location)

- Minimization is encoded in the '-' sign
- 'a' = current position
- 'b' = next position
- Gradient term (F(a)) = direction of steepest ascent
  - Minus sign inverses to make steepest **descent**
- Gamma = waiting factor

We are trying to minimize the function of yi (2)

- with gradient descent, we should just iterative over (1) until we get to a shallow slope that we are confident are int eh bottom.
- B, our new location, becomes yi prime. Equa to our old location, yi,
- **Use in update step!**

NOV 21, 2018

pick up midterm from Travis tomorrow.
quiz from last class to before quiz 1

Unsupervised learning
↳ measurement vs. motion model.
↳ policy estimation or kinematics (move from
↳ PID control

Learning Paradigms (Supervised, Unsupervised, semi-supervised, imitation learning)

Supervised (hard bc we need answers to problems to learn / give examples to learn) ⇒ would need many labels (BAD)
↳ have access to labelled training data.
↳ learn a function f from training data such that $f(x')$ correctly predicts y' for unseen pairs (x', y') where $x' \sim p(x)$
↳ ex. which drug to give to patients dilemma.

Unsupervised (determining how to group, based on semantics)
- give unlabelled data x
↳ ex. learn ways of grouping of data. (clustering)
↳ learn distribution of data $p(x)$ (density estimation)
↳ learn to categorize articles (use histograms (group same histogram)

Semi-supervised
- many data & few label examples.

Tesla has huge dataset while customers are just driving
↳ has a lot of labelled data (supervised)

Immitation learning: mimicry

Transfer learning:
- policy learned can be applied to more domains w/ some transformation
- cons: why is this hard? ⇒ noise
↳ huge diff btw simulator vs. real world
↳ kinematics/dynamics & images are not the same.
↳ behavior is unpredictable, must if we learn the wrong simulator.
↳ expensive

Reinforcement learning
- agent interacts w/ environ, environ respond by giving reward (punishment
- optimize action to get greate cummulative reward (maintain & longterm)

Diff btw R & Super
- diff parts of world
- things you do now affect future outcome.

COMP417
Introduction to Robotics and Intelligent Systems

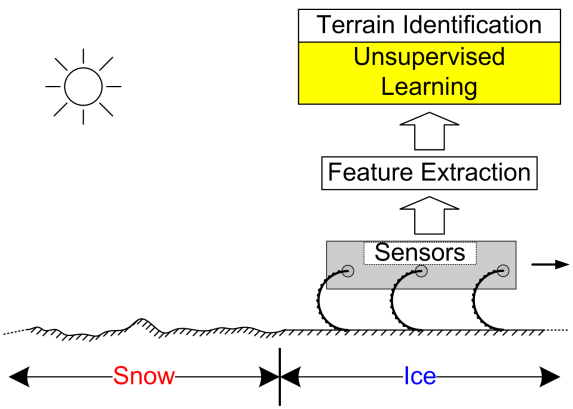
Unsupervised learning in robotics: example

McGill

MRL Mobile Robotics Lab at McGill University

Terrain-dependent walking

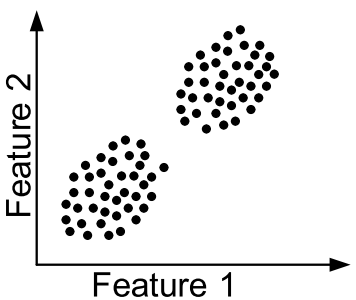
Unsupervised Localization & Terrain Identification



Giguere & Dudek, 2008

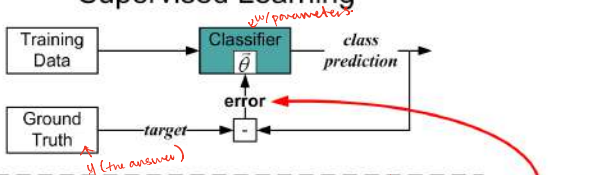
Unsupervised Case Learning of Terrains and Places

↳ then formulate classes



Classifier Training

Supervised Learning

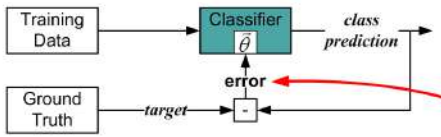


minimize

Classifier Training

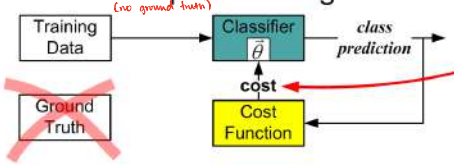
Supervised Learning

$p_c(t) = f(\vec{\theta}, \vec{x}_t)$



Our Unsupervised Algorithm

minimize



Giguere & Dudek, 2008

Proposed Algorithm

- **A classifier** $p_c(t) = p(c \mid \vec{x}_t)$

  generalizing over local distances in feature space

  $X = \{\vec{x}_1, \vec{x}_2, .. \vec{x}_T\}$

  –e.g., k-Nearest Neighbors

- **Training rule:**

$$\arg\min_{\theta} \sum_{c=1}^{N_{classes}} \frac{\sum_{t=1}^{T} [p_c(t+1) - p_c(t)]^2}{\mathrm{var}(\{p_c(1), .., p_c(T)\})^2}$$

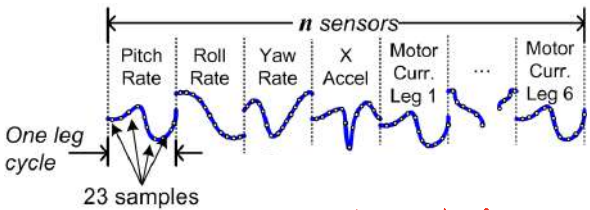
Giguere & Dudek, 2008

AQUA Robot (Based on *RHex*)

G. Dudek, M. Jenkin, C. Prahacs, A. Hogue, J. Sattar, P. Giguere, A. German, H. Liu, S. Saunderson, A. Ripsman, S. Simhon, L. A. Torres-Mendez, E. Milios, P. Zhang, I. Rekleitis. *A Visually Guided Swimming Robot*, Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1749-1754, 2005.

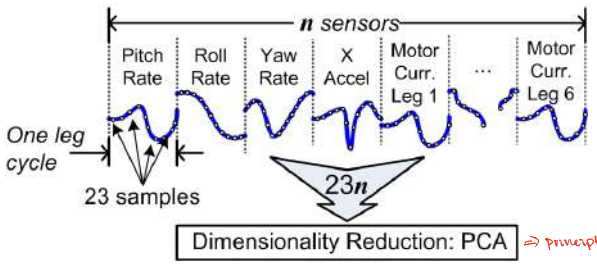
Features + Dimensionality Reduction

(2D)



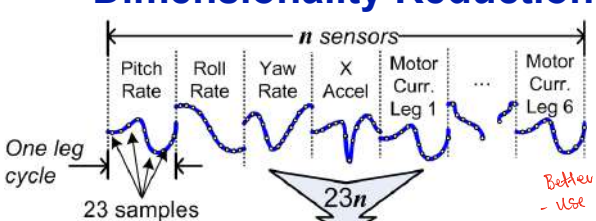
- measured what legs felt while walking on terrain
- flat terrain (low Pitch, Roll, Yaw)
- high current

Features + Dimensionality Reduction



⇒ principle component analysis is useful for face recognition

Features + Dimensionality Reduction



less variance = throw away.

BAD WAY ↳ keep x w/ mean

Want to make 2D components to 1D using PCA
↳ represent 2D data pts in 1D
↳ take an x-axis, measure vertical distance w/ data pts.
↳ average & covariance
↳ replace dimension w/ mean (only keeping one coord.)

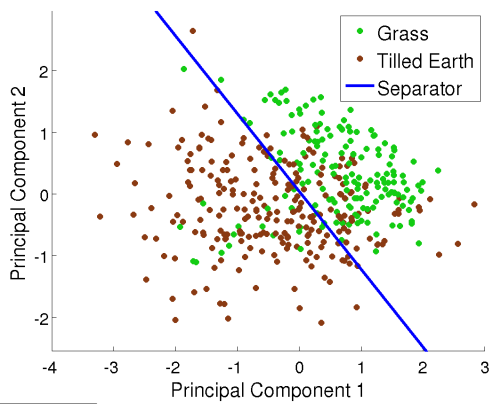
Better Way
- use y-axis
- remember y-value & take average of x.
- better ∴ less variance.

ex.

② Fit a line in data pts given parameter of one pt & equation of line.

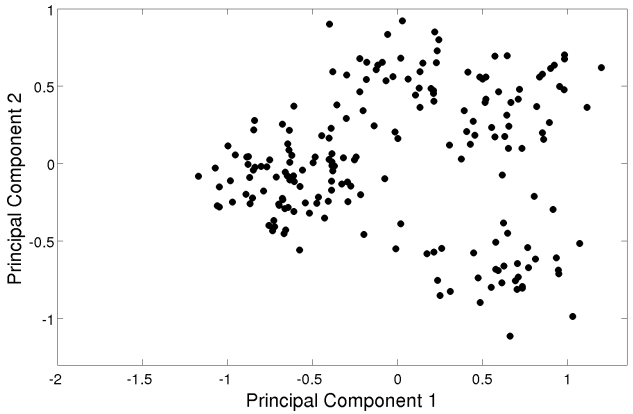
Giguere & Dudek, 2008

2 Terrains (Linear Separator)



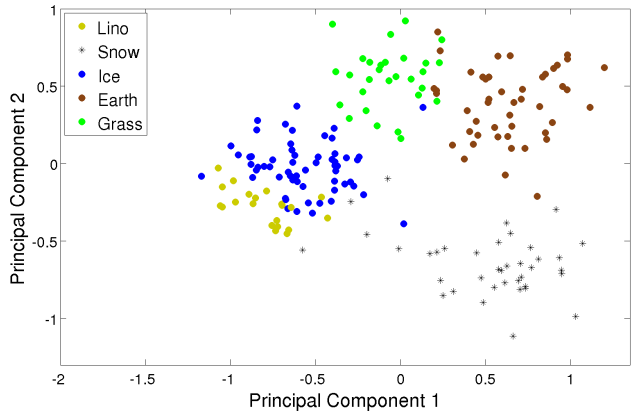
Segment Length ≈ 20

5 Terrains Data



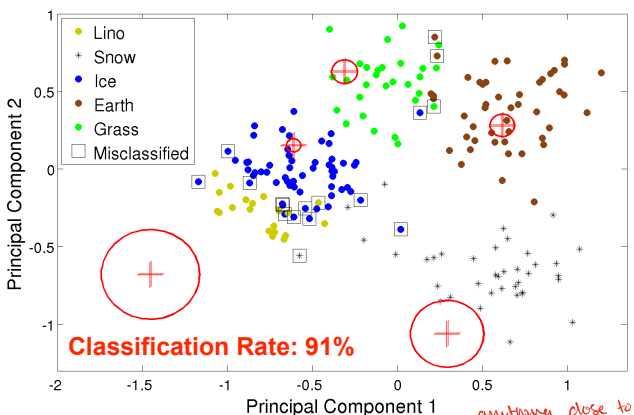
Segment Length = 6

5 Terrains Data

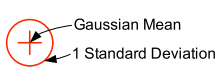


Segment Length = 6

5 Terrains (Gaussian Mix. Model)

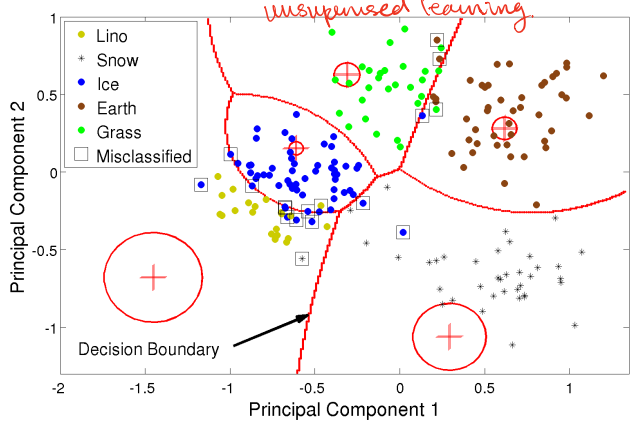


Segment Length = 6

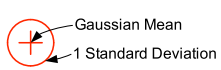


anything close to ⊕ would be classified as a town by ⓒ ⊕ for 5 terrains

5 Terrains (Gaussian Mix. Model)



Segment Length = 6

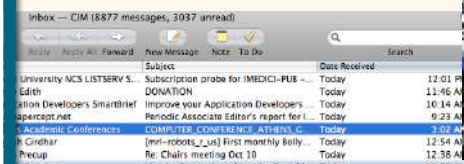


Giguere & Dudek, 2008

Outdoor evaluation

Data summarization

- Robotic systems can collect vast amounts of data.
- Where are the good bits?
- Can we find them quickly?



Navigation Summaries

(video case)

Highlights of a robot's experience (location and sensor data). (universal theory)

Did anything really odd happen?

What were the most notable moments?

Not a *complete* summary.

Delivered on-line: what's interesting *so far*.

Work with Yogesh Girdhar

Navigation Summaries

A summary of the underwater robot's "experience" based on visual highlights:
i.e. select the most interesting images.

Issue: characterize mutually distinctive images in a domain-independent manner.

Surprise & summarization

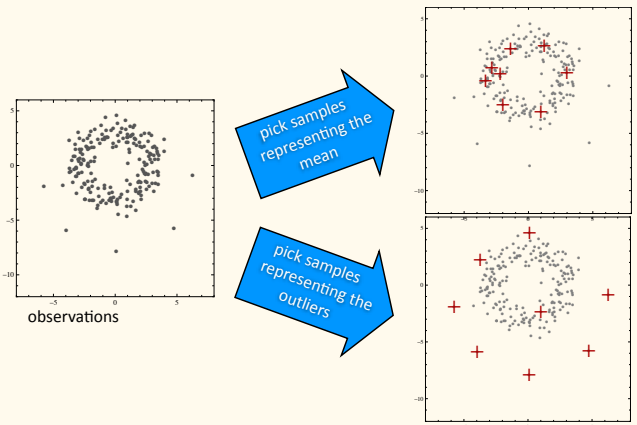
- What is a good summary?

- To be a synopsis of the data
- To minimize "surprise"
- Surprise: the extent to which a new measurement differs from anything in the summary.

"There's nothing (too) new under the sun."

(if you have a good summary)

What is a good Summary?



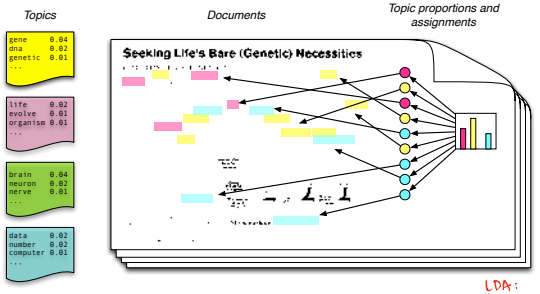
Subproblems

1. How to we measure content in images ?
2. How to we surprisingness (distinctiveness)?
3. How to we select the summary set?

Latent Dirichlet Allocation (LDA)

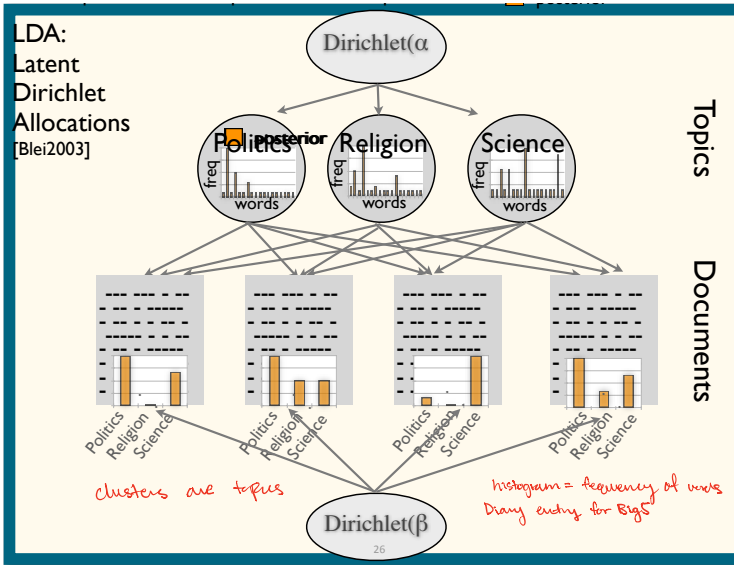
- LDA is a method for:
  - Learning a set of topic labels for a group of documents
  - Classifying documents with respect to which topics they relate to
  - Grouping documents based on topics covered.
- LDA is traditionally powerful, but slow and limited to substantial text documents.
- We are looking at making it faster (real-time), on-line and suitably for product and review clustering.

Generative model for LDA (ex of unsupervised)



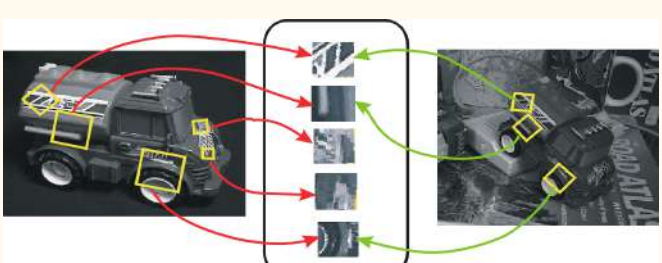
- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

LDA:
- have topics, documents, words from topics, categorize documents based on topics w/ words



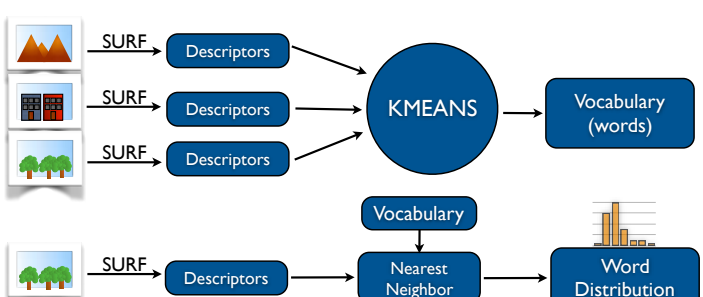
Visual words

- Detect local features that commonly occur.
- These features play the role of "words." Such a bag of words approach is a powerful representation for images.



SIFT Features

Bag of Words



- Offline : vocabulary over ***all*** images.
- Online: vocabulary over ***summary*** images. SURF is replaced by Oriented BRIEF (ORB) features.

Topics

- Directly comparing word distributions is difficult. Instead, compare the topics associated with an image.

$$P(w_i = v|M) = \sum P(w_i|z)P(z|M)$$

Word: w_i

Document (image) model: M

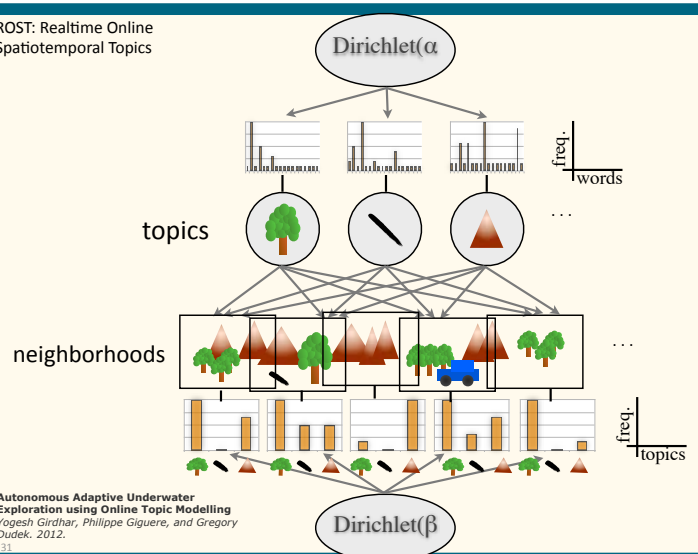
Topic label: z

- Document M: distribution θ(k) over topics
- Topic: a distribution φ(v) = P(wi = v|zi = k) over vocabulary words.

Topics

- Robot observations are generally continuous in space *and* time: corresponding semantic descriptor must also be continuous.
- Update online and in realtime.
- Must update topic labels for previous observations when computing topic labels for a new observation.
- Compute topic distributions over spatiotemporal neighborhoods, not just images.

ROST: Realtime Online Spatiotemporal Topics

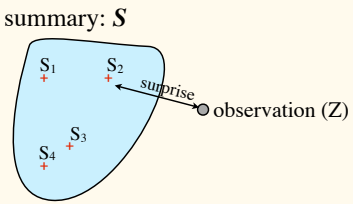


Autonomous Adaptive Underwater Exploration using Online Topic Modelling
Yogesh Girdhar, Philippe Giguere, and Gregory Dudek. 2012.

Surprise

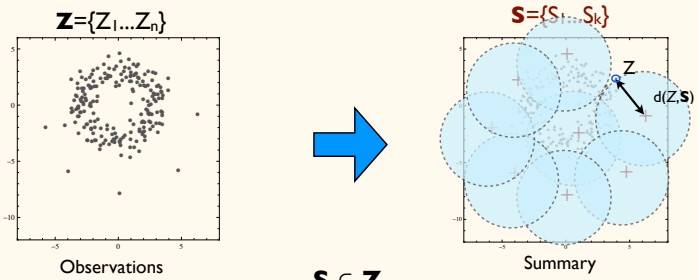
- Informally: How much a new observation differs from "what we already know."
- More formally: the difference of a new observation from the items in our summary.

Computing Observation Surprise



$$\text{surprise}(Z) = \min_i d(Z,S_i)$$

Extremum Summary: Cost Function

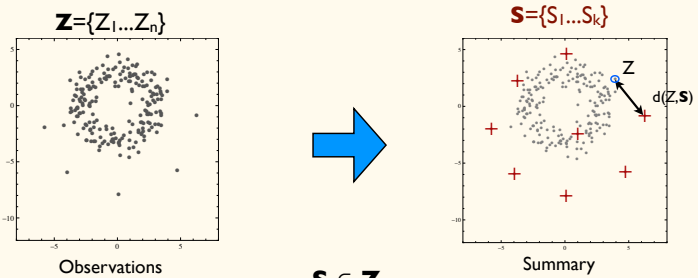


$\mathbf{S} \subseteq \mathbf{Z}$,

$\text{cost}(\mathbf{S}) = \max_i d(Z_i,\mathbf{S}) = \max_i \min_j d(Z_i,S_j)$,
find **S** which minimizes cost(**S**)

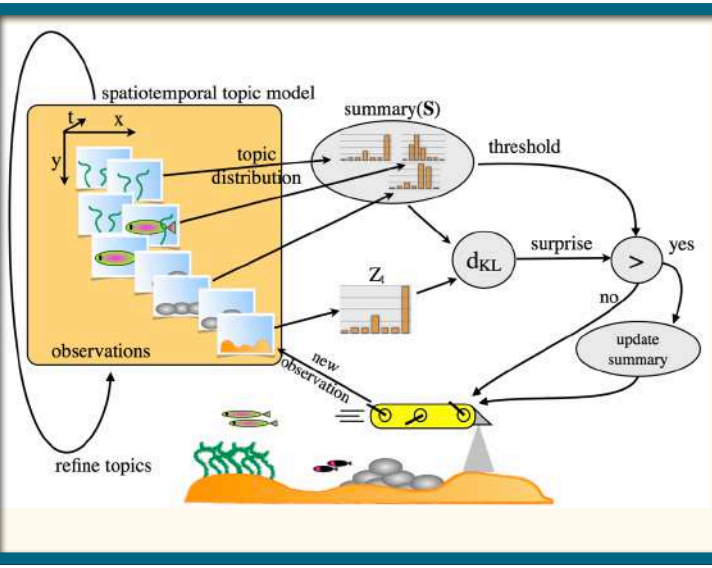
α-approximation is NP Hard, for $\alpha<2$ [Hochbaum1985]

Extremum Summary: Minimize Max Surprise



$\mathbf{S} \subseteq \mathbf{Z}$,

$\text{cost}(\mathbf{S}) = \max_i d(Z_i,\mathbf{S}) = \max_i \min_j d(Z_i,S_j)$,
find **S** which minimizes cost(**S**)

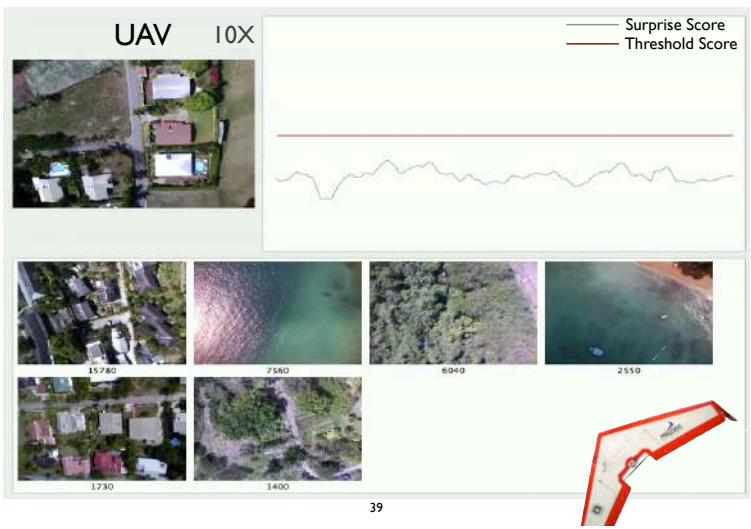


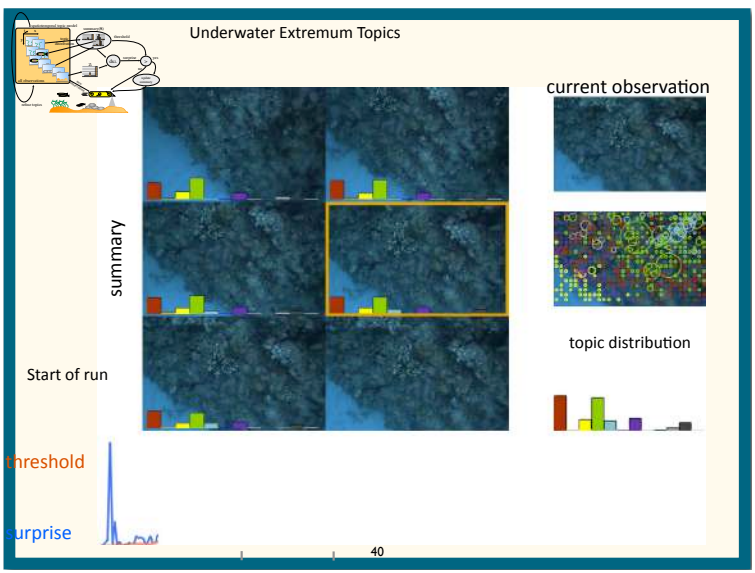
Correctness?

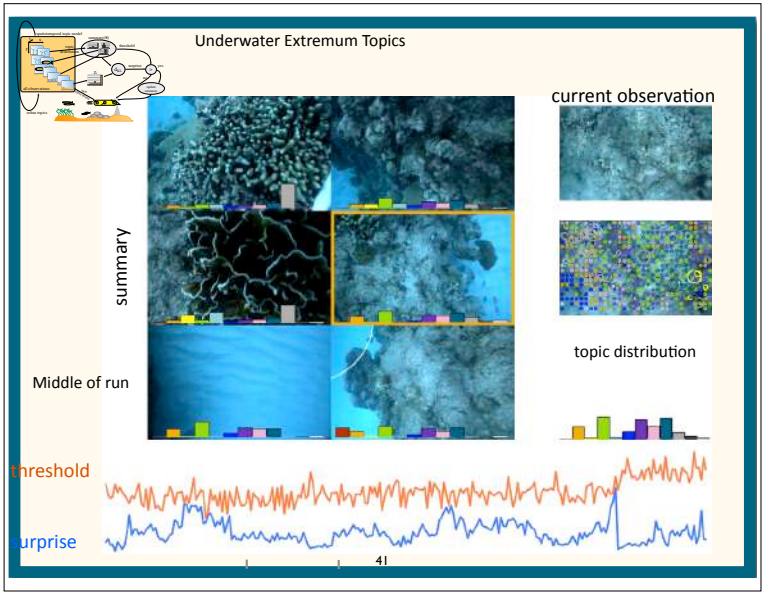
What makes a summary good?

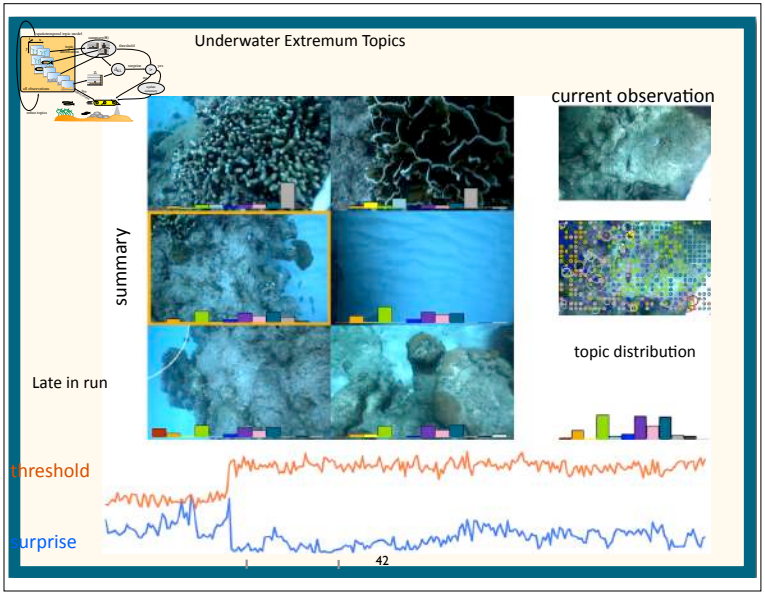
1. It "looks good"
2. It scores well in terms of objective measures?
   - Perplexity: inverse of the geometric mean per-word likelihood.
3. It scores well in objective blind human trials

Ed Ruscha, Twentysix Gasoline Stations, 1962









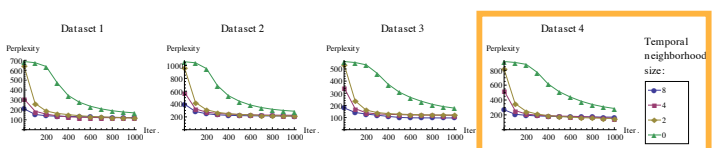
Correctness?

What makes a summary good?

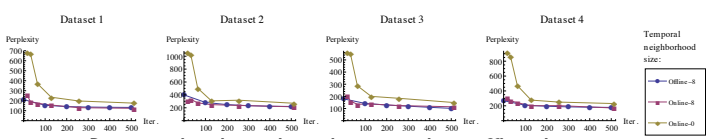
1. It "looks good"
2. It scores well in terms of objective measures?
   - Perplexity: inverse of the geometric mean per-word likelihood.
3. It scores well in objective blind human trials

Perplexity

(how well does the model describe the data)



Increasing neighborhood size results in faster convergence



Proposed online algorithm matches offline learning

Correctness?

What makes a summary good?

1. It "looks good"
2. It scores well in terms of objective measures?
   - Perplexity: inverse of the geometric mean per-word likelihood.
3. It scores well in objective blind human trials

Outline

- Introduction
  - –What's so great about robotics?
  - –Multi-agent robotics
- Interaction
  - –Coral reef monitoring: a multi-robot system
- Robots finding novelty on behalf of people
  - –Robots as surrogates & envoys
    - Discovery and data summarization
- Illustrative video
- Conclusion

Seeking novelty

- Moderate speed as a function of novelty [Girhdar, Giguere, Dudek, ISER 2012]
  - Requires multiplicity of gaits to permit full range of speeds, including hovering.
- Moderate trajectory itself to cover more interesting terrain. [Girhdar,, Dudek, ICRA 2014]
  - Model of novelty within subregions of the image.

Seeking novelty

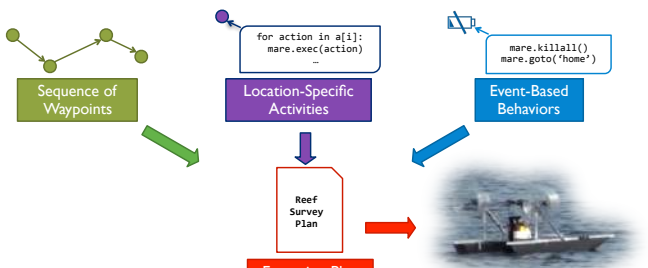
Couple exploratory navigation to novelty

Why Dirichlet Priors?

- Allows to place priors on sparseness of the distributions: word-topic, and topic-neighborhood histogram.
- Sparse topic-neighborhood matrix => encoding of neighborhoods is sparse.
- Sparse word-topic matrix => topics are independent, and represent different visual objects in the scene.
- Faster learning.

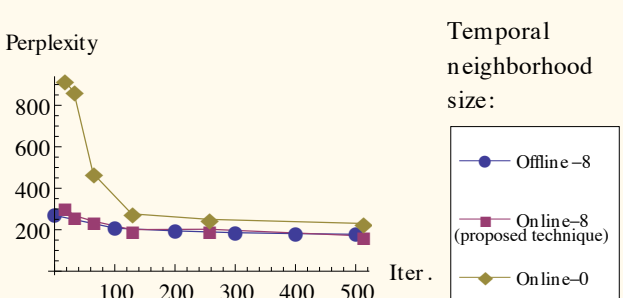
Graphical State Space Programming (GSSP)

- Prioritized-concurrent behavior management framework
- Hybrid textual-graphical programming environment



J. Li, A. Xu, G. Dudek. **Graphical State Space Programming: A Visual Programming Paradigm for Robot Task Specification.** *in Proc. of the Int. Conf. on Robotics and Automation (ICRA 2011)*, pp. 4846-4853, Shanghai, China. May 2011.

Online vs Offline learning



Proposed online algorithm matches offline performance

Given a location and time $(\mathbf{x}_i, t_i)$, we use the following generative model for the corresponding observed word w_i:

1. word distribution for each topic k: $\phi_k \sim \text{Dirichlet}(\beta)$,
2. neighborhood for an observation at $(\mathbf{x}_i, t_i)$: $G(\mathbf{x}, t) \sim$ uniformly from all neighborhoods which contain $(\mathbf{x}_i, t_i)$,
3. topic distribution the neighborhood $G(\mathbf{x}, t)$: $\theta_{G(\mathbf{x},t)} \sim \text{Dirichlet}(\alpha)$,
4. topic label for location $(\mathbf{x}_i, t_i)$: $z_i \sim \text{Discrete}(\theta_{G(\mathbf{x},t)})$,
5. word observed at location $(\mathbf{x}_i, t_i)$: $w_i \sim \text{Discrete}(\phi_{z_i})$,

where $x \sim Y$ implies that random variable x is sampled from distribution Y.

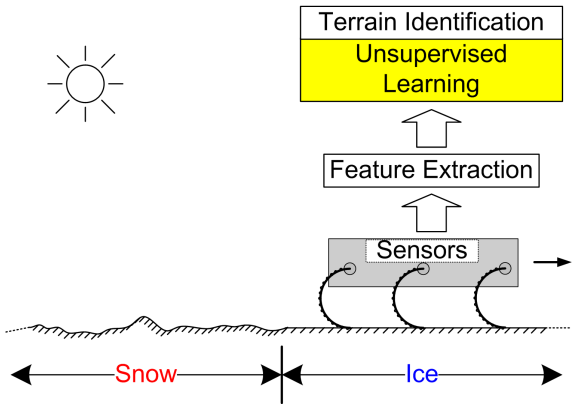
Using the Gibbs sampler proposed by Griffiths et al.[7], we can define the posterior topic distribution of a neighborhood $G(\mathbf{x}, t)$:

$$\mathbb{P}(z_i = k | \mathbf{z}_{-i}, w_i = w, \mathbf{w}_{-i}, G(\mathbf{x}, t)) \propto \frac{n^w_{k,-i} + \beta}{\sum_{w=1}^{V}(n^w_{k,-i} + \beta)} \cdot \frac{n^k_{G(\mathbf{x},t),-i} + \alpha}{\sum_{k=1}^{K}(n^k_{G(\mathbf{x},t),-i} + \alpha)}, \quad (2)$$

53

McGill

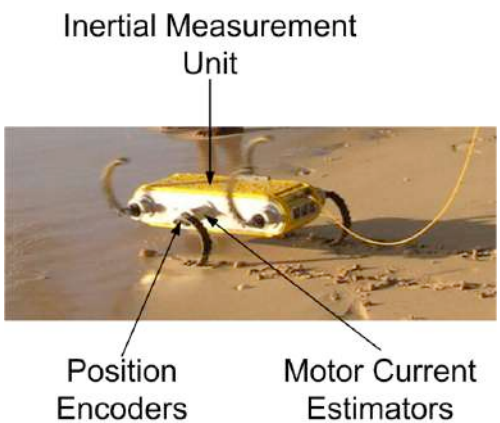
Unsupervised Terrain Identification



Giguere & Dudek, Robotics Science and Systems (RSS), 2008.
Giguere & Dudek, Robotics, Autonomous Robots, 2009.

McGill

AQUA Contact Sensors



Giguere & Dudek, Robotics Science and Systems (RSS), 2008.
Giguere & Dudek, Autonomous Robots, 2009.

Nov 23, 2018

Computing observation surprise

- Surprise(Z) = min d(Z, S)

Extremum summary: cost function

Competitive ratio

Cost function vs. Min max

Minimize max surprise

- Summarizing by averages of datasets (clusters)

How to compare histograms?

- Ex. Histogram of freq of words
    - Euclidian distances over histograms Sigma(ha-hc) ^2
        - Terrible over continuous data (due to shifting)
- Earth movers distance: how much do I have to shift to make histograms equal
- Surprising, update summary
    - Not surprising, throw image away
- What makes a summary good?
    - Looks good
    - Scored well in some object measure
    - It scores well in objective blind human trails
- If surprising, lose prior frame with the new surprising frame

How to recognize

- Obstacles
- Places
- Objects
- People
- Graps of objects
    - Two-day we look quickly at recognizing images pictures of things (under narrow assumptions)

Blue line in Space of faces, changes length for how bright image it is

Markov Decision Processes (MDP)

Why do we want rewards sooner? To be on the right track

- future rewards has more uncertainty

No policy gradient

NOV 30

Revolute rotate

Prismatic extend

Hydraulics - pumps, strength/energy is displaced at the end of vectors(fingers)

End-effectors: things at the end

Manipulators (no need to memorize)

- Cartesian PPP (3 prismatic)
- Cylindrical RPP(rotate and 2 prismatic)
- Spherical RRP (2 rotate and 1 prismatic)
- Articulated RRR (3 rotate)
- SCARA: RRP
- Hand coordinate: 2 plates squished together

Motion control methods

- Point to point control
- Continuous path control
    - Follow curves

    - controlling where the end-effector is
- Robot specification
    - Axis
    - Degree of freedom (DOF)
    - Workspace
    - Payload (load capacity) = how much it can lift
    - Precision vs. Repeatability = same place repeatedly

Forward and Inverse Kinematics
Caging (use arms to grasp; cage around object)
Approaches to grasping

- cons: you need a lot of training and data, in simulation and need **transfer learning**
-

PRM (random sample points and linking, no start or end ipts)

- Sample points have to be close, in order to join them (same notion in art)
- Especially high dim, as long ptr are close together, its doable
- Add points, connect, etc.
- Need a bound on how many pts we need depends on the gaps of our words
- Small gaps between worlds, more sample points needed
-

RRT(with beginning and ed and find optimal path)
Visibility planning: corners, well in 1D

- Connect vertices
- Optimal: gives shortest path only in 2D
    - Does not work for 3D, can go over edge, not only vertices
- Reduced visibility graph: just need around 4 objects for an object

Smoothing planned paths: taking random points and make smooth

- Cut out intermediate edges and vertices (

Artificial potential fields -Potential energy and forces

- Cons: Almost never give u shortest path, can oscillate
- Pros: always in free space (good for planning methods), **its local (you dont have to know the whole environment, just nearby**)

PID: integral divide by the size of window
ROS: OPEN CV
REMEMBER HILDEBRAND GAIT DIAGRAM
No zero momentum point walk on exam

Wagon steering: rolling and rotate towards axis

No sensor characteristics
Measuring with gyro will eventually get lost (error accumulates)
Gyroscripts: measure rotation, you get 3D orientation

Know diff kinds of lens distortion

- Lens is to mimi pinhole camera wihtout actual pinhole
    - Pinhole bad bc limits amount of light

RGBD
LIDAR 2D and @D

- Measure shape of light
- 3D: time and flight

Radar: lidar with radio

- Unlike LIDAR, doesn't work well in fog, rain snow etc

GPS: satellites, measure time and distance

- Good GPS system: know time when was sent and received, time is converted to distance
    - Why is it bad?have to know exactly when they send it and how often
        - Need a good clock to measure time delay

Proprioceptive sensors: internal measurements with internal sensors

- Ex. IMU, gyroscopes

Rotary encoder: measure how many turns
Actuators, dont need to know motors
Connection between methods

Localization

No drawing variance for Gaussian distribution
3sigma = more away and more uncertain

Bayes Filter: Derivation

- Explain a step of derivation (equation)

EKF can diverge, gets worse and worse (due to wrong estimate of state), due to incorrect linearization
SLAM: estimates of pose and landmarks and updates, and add new landmarks etc.

- covariance matrix depend on number of landmarks
- Why is it hard? Which landmark attached to which object...how do u know

Multimodal Distribution - Particle filer

- does not use Guassian
- Ideal need infinite number of particles
- Make Kaman filter slow: inverse in the matrix of the common gain
    - Inverse of n-dim matrix: n^3 time
    - bad if using SLAM
    - So use **resampling**