LEC1

Traditional Robotics :

▪ Controlled environment

▪ Well understood

Flexible automation

▪ Mining, agriculture,…

▪ Logistics

▪ Household

▪ Medicine

▪ Dangerous environments

(Space, under water,

nuclear power plants, …)

▪ Toys, entertainment

I think welding also

Cognitive Robotics :

- Have cognitive functions
- Operate in dynamic real-life environments
- Exhibit a high degree of 2robustness in coping with
- unpredictable situations

-> Tour Guide Robot Minerva

-> Autonomous Vacuum Cleaners

-> Autonomous Lawn Mowers

-> Autonomous Lawn Mowers

, DARPA Grand Challenge 2005

, The Google Self-Driving Car

Autonomous Quadrotor Navigation , Stair Climbing (HRL)

, Cognitive Robot Cosero AIS Lab Uni Bonn (Sven Behnke)

Probabilistic Robotics :

▪ Explicit representation of uncertainty

▪ Using the calculus of probability theory

▪ Perception = state estimation

▪ Action = utility optimization

***->P(.)* is called probability mass function .**

$$\sum_x P(x) = 1$$

$$\int p(x)dx = 1$$

. If X and Y are independent then

P(x,y) = P(x) P(y)

▪ *P(x | y)* is the probability of *x* given *y*

P(x | y) = P(x,y) / P(y) P(x,y) = P(x | y) P(y)

▪ If X and Y are independent then

P(x | y) = P(x)

Law of Total Probability :

$$P(x) = \sum_y P(x \mid y) P(y)$$

$$p(x) = \int p(x \mid y) p(y) dy$$

Marginalization :

$$P(x) = \sum_y P(x, y)$$

$$p(x) = \int p(x, y) dy$$

Bayes' Rule :

$$P(x \mid y) = \frac{P(y \mid x) P(x)}{P(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

posterior

Conditional Independence :

When z is known , y does not tell us anything about x

$$P(x, y \mid z) = P(x \mid z) P(y \mid z)$$

- Often causal knowledge is easier to obtain (ghaleban **count frequencies!**)

- Bayes' rule allows us to use causal knowledge to compute diagnostic .

- Under the Markov assumption, recursive Bayesian updating can be used to efficiently combine evidence .

Recursive Bayesian Updating

$$= \eta_{1\ldots n} \left[\prod_{i=1\ldots n} P(z_i \mid x) \right] P(x)$$

LEC2

-> In contrast to measurements, actions generally increase the uncertainty.

Discrete case:

$$P(x \mid u) = \sum_{x'} P(x \mid u, x') P(x')$$

Continuous case:

$$P(x \mid u) = \int P(x \mid u, x') P(x') dx'$$

Bayes Filters: Framework

▪ **Sensor model** *P(z | x).*

▪ **Action model** *P(x | u, x').*

▪ **Prior** probability of the system state *P(x).*

-> *wanted :*

$$Bel(x_t) = P(x_t \mid u_1, z_1 \ldots, u_t, z_t)$$

2 IMP Markov Rules :

1) **current measurement depends only on current state .**
2) **current state depends only on previous state & action .**

Bayes Filter Interpretation :

// prediction uses motion model , while correction uses sensor model .

- Prediction

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx_{t-1}$$

- Correction

$$bel(x_t) = \eta\, p(z_t \mid x_t)\, \overline{bel}(x_t)$$

Bayes Filters come in many

Forms :

- Kalman Filters
- Particle Filters
- Hidden Markov Models
- Dynamic Bayesian Networks
- Partially Observable Markov Decision Processes (POMDPs)

Discrete Kalman Filter :

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

$$z_t = C_t x_t + \delta_t$$

A: describes how the state evolves from *t-1* to *t* without controls or noise.

B: describes how the control *ut* changes the state from *t-1* to *t*.

C: describes how to map the state *xt* to an observation *zt*.

ε t , δt ***: Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance Rt and Qt, respectively.***

// revise kalman eqs

-> K is kalman gain

Kalman Filter Summary :

- Highly efficient
- Optimal for linear Gaussian systems , but most systems are not linear .
- Data Association Problem , unimodel .

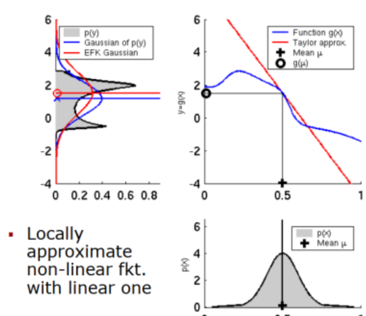
LEC3

-> EKF Linearization: First Order

Taylor Expansion

// revise EKF eqs

EKF Linearization (1)



- Approximation quality depends depends on deviation from g() in the used range .

§ **Problem classes**

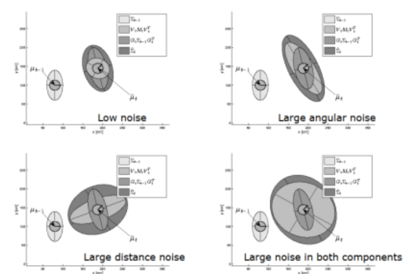
§ Position tracking (initial pose known)

§ Global localization (initial pose unknown)

§ Kidnapped robot problem (recovery)

Zoom

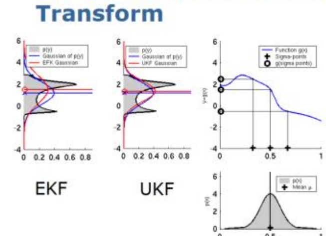
EKF Prediction Step



EKF Summary :

- Highly efficient
- Not optimal!
- Can diverge if nonlinearities are large!
- Works surprisingly well even when all assumptions are violated!

Linearization via Unscented Transform



- Represent belief by Sigma-points

// sigma points -> important points for me .

UKF Summary :

§ Highly efficient: Same complexity as EKF, with a constant factor slower in typical practical applications

§ Better linearization than EKF: Accurate in first two terms of Taylor expansion (EKF only first term)

§ Derivative-free: No Jacobians needed

§ Still not optimal!

LEC4

- Robot motion is inherently uncertain

- The error accumulates

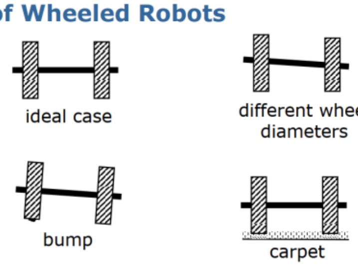
Typical Motion Models 2 :

§ Odometry-based: **when systems are equipped with wheel/joint encoders .**

-> Example Wheel Encoders , light noise will cause a problem .

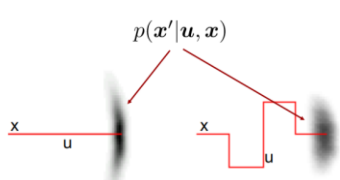
§ Velocity-based (dead reckoning): **Calculate the new pose based on the velocities and the time elapsed**

Some Reasons for Motion Errors of Wheeled Robots



// revise motion model eqs odom

-> the banana shaape is a result of the translation and rotation gaussian noises



§ Sampling from a normal distribution

$$\frac{1}{2}\sum_{i=1}^{12} rand(-b,b)$$

§ Sampling from a triangular distribution

$$\frac{\sqrt{6}}{2}\left[\mathrm{rand}(-b,b)+\mathrm{rand}(-b,b)\right]$$

§ Sampling from arbitary distribution

Rejection Sampling

§ Sample *x* from a uniform distribution from [-*b*,*b*]

§ Sample *c* from [*0, max f*]

§ if *f* (*x*) > *c* keep the sample

otherwise reject the sample

-> odom based is better than velocity based

LEC5

Sensors for Mobile Robots

Proprioceptive sensors:

§ Accelerometers – get a

§ Gyroscopes – get omega

§ Compasses or magnetumeter

Typical proximity sensors:

§ Sonars

§ Laser range-finders – use light beams

Visual sensors:

§ (Stereo) Cameras

§ Structured light (RGBD cameras)

Infrastructure-based sensors: GPS, WLAN

Beam-Based Sensor Model

- Assumption: The individual measurements are independent given the robot's pose:

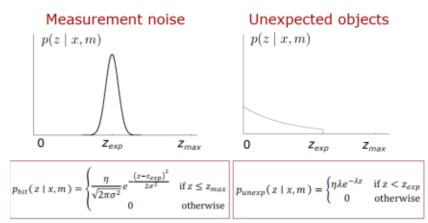
$$p(z \mid x, m) = \prod_{k=1}^{K} p(z_k \mid x, m)$$

Typical Measurement Errors 4

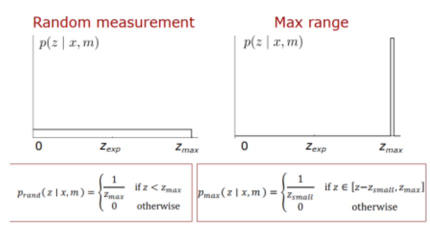
1. Beams reflected by known obstacles
2. Beams reflected by people / objects
3. Random measurements
4. Maximum range Measurements

-> Sonar has a problem called crosstalk

Beam-Based Proximity Model

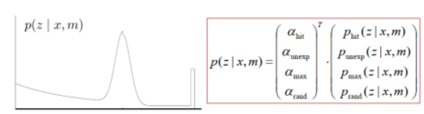


Beam-Based Proximity Model



//weighted sum

Resulting Mixture Density



Approximation

- Maximize log likelihood of the data

$$p(z \mid z_{\exp})$$

- Search space of n-1 parameters
- Hill climbing

- Gradient descent
- Genetic algorithms

Summary Beam-Based Model

§ **Assumes independence between beams**

§ Problem: Overconfident estimates

§ **Models physical causes for measurements**

§ Mixture of densities for these causes

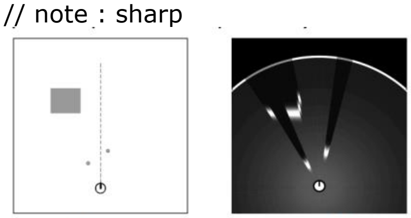
§ Assumes independence between causes

- Different models should be learned for different angles at which the sensor beam hits the obstacle (sonar)
- Determine expected distances by ray casting -> less memo , but slow
- Expected distances can be pre-computed -> more memo but fast

Disadvantages:

- not smooth at edges
- not very efficient (ray casting or precomputed lookup tables)

// note : sharp



Map *m* Likelihood field

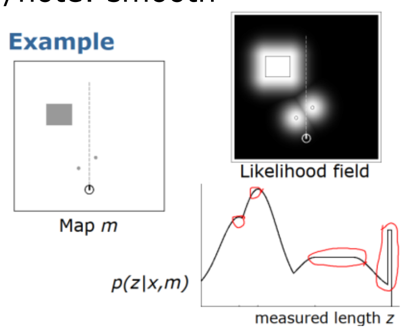
End-Point Model

- Instead of following along the beam, just check the end point of the beam
- Precompute a likelihood field (distance grid) , Precomputed independently of robot pose
- Probability is a mixture of:
- a Gaussian distribution evaluating the **distance to the closest obstacle -> new**
- a uniform distribution for random measurements
- a small uniform distribution for max range measurements

- Again, independence between different components is assumed

//note: smooth

Example



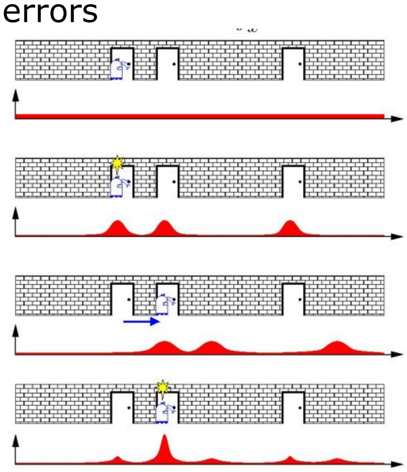
Properties End-Point Model

- Highly efficient
- Distance grid is smooth
- Ignores physical properties of beams
- Treats sensor as if it can see through walls

LEC6

Discrete Filter

- an alternative way for implementing the Bayes Filter
- Histograms for representing the density
- Can represent multi-modal beliefs and recover from localization errors



- Huge memory and processing requirements

Because To update the belief, one has to iterate over all cells of the grid , but When the belief is peaked, one wants to avoid updating irrelevant aspects of the state space .

- Accuracy depends on the resolution of the grid

Particle Filter

-> **efficiently** represent **non-Gaussian distributions**

-> Basic principle

- Set of state hypotheses ("particles")
- Survival-of-the-fittest

//note particles are state hypothesis (x, y, theta) and weights .

// note delta in sampling

Particle Set

- Set of weighted samples

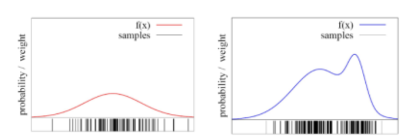
$$\mathcal{X} = \left\{ \left\langle x^{[j]}, w^{[j]} \right\rangle \right\}_{j=1,\dots,J}$$

state hypothesis importance weight

- The samples represent the posterior

$$p(x) = \sum_{j=1}^{J} w^{[j]} \, \delta_{x^{[j]}}(x)$$

- The more particles fall into a region, the higher the probability of the region



Closed Form Sampling is Only Possible for Few Distributions

- Example: Gaussian

$$x \leftarrow \frac{1}{2} \sum_{i=1}^{12} \text{rand}(-\sigma, \sigma)$$

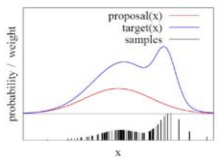
// this one is for arbitary distribution

// note w = f/pi=target/propsal

Importance Sampling Principle

- We can use a different distribution π to generate samples from f
- Account for the "differences between π and f" using a weight $\omega = f(x)/\pi(x)$
- target f
- proposal π
- Pre-condition:

$f(x) > 0 \rightarrow \pi(x) > 0$



Particle Filter

▪ Bayes filter
▪ Non-parametric
▪ Models the distribution by weighted samples
▪ **Prediction:** draw from the proposal
▪ **Correction:** weigh particles by the ratio of target and proposal
->The more samples we use, the better is the estimate!

Particle Filter Algorithm (3steps)

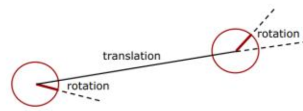
1. Sample the particles using the proposal distribution
2. Compute the importance weights
3. Resampling: Draw sample i with wi probability and repeat j times

Monte Carlo Localization (MCL)

▪ Each particle is a pose hypothesis

1. **Prediction** : For each particle, sample a new pose from the the motion model (proposal)
2. **Correction** : Weigh samples according to the observation model (target)
3. Resampling: Draw sample i with wi probability and repeat j times

Reminder: Odometry Motion Model



Decompose the motion into
- Traveled distance
- Start rotation
- End rotation

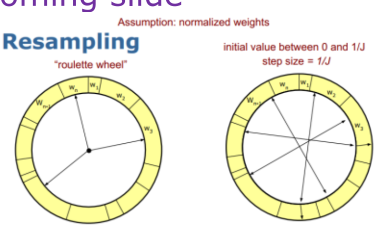
▪ For each particle, draw a new pose by sampling from three normal distributions

Resampling

-> "Replace unlikely samples by more likely ones"
-> Survival-of-the-fittest principle
-> "Trick" to avoid that many samples cover unlikely states
-> Needed as we have a limited number of Samples

// take care of the coming slide

Resampling



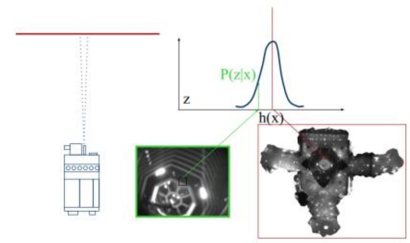
- Draw randomly between 0 and 1
- Binary search
- Repeat J times
- $O(J \log J)$

- Systematic resampling
- Low variance resampl.
- $O(J)$
- Also called "stochastic universal resampling"

34

// note : vision based localization depends on light

Vision-Based Localization



// How to deal with kidnapped robot ? (IMP)

▪ At each time step, randomly insert a fixed number of samples
▪ Alternatively, insert random samples proportional to the average likelihood of the particles

Summary – PF Localization

▪ Particles are propagated according to the motion model
▪ Particles are weighted according to the likelihood of the observation
▪ Called: Monte-Carlo localization (MCL)
▪ The art is to design appropriate motion and sensor models

LEC7

The General Problem of Mapping

- Formally, mapping involves, given the sensor data

$$d = \{u_1, z_1, u_2, z_2, \cdots, u_t, z_t\}$$

- to calculate the most likely map

$$m^* = \text{argmax}_m P(m|d)$$

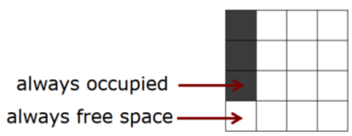
Grid Maps

§ Discretize the world into cells
§ Grid structure is rigid
§ Each cell is assumed to be occupied or free space

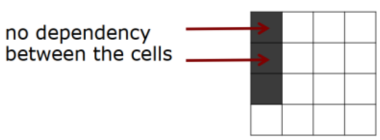
§ Non-parametric model
§ Large maps require substantial memory resources
§ Do not rely on a feature detector

Assumptions

1- The area that corresponds to a cell is either completely free or occupied .
2- The world is static



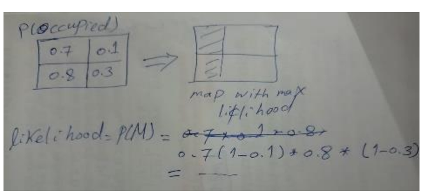
3- The cells (the random variables) are **independent** of each other



This leads to:

$$p(m) = \prod_i p(m_i)$$

map   cell



Estimating a Map From Data

$$p(m \mid z_{1:t}, x_{1:t}) =$$

$$\prod_i p(m_i \mid z_{1:t}, x_{1:t})$$

$$\frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})}$$

//highly important , zoom in

$$= \underbrace{\frac{p(m_i \mid z_t, x_t)}{1 - p(m_i \mid z_t, x_t)}}_{\text{uses } z_t} \underbrace{\frac{p(m_i \mid z_{1:t-1}, x_{1:t-1})}{1 - p(m_i \mid z_{1:t-1}, x_{1:t-1})}}_{\text{recursive term}} \underbrace{\frac{1 - p(m_i)}{p(m_i)}}_{\text{prior}}$$

//also important

$$p(m_i \mid z_{1:t}, x_{1:t}) \rightarrow p(x)$$

$$= \underbrace{\frac{p(m_i \mid z_t, x_t)}{1 - p(m_i \mid z_t, x_t)}}_{\text{uses } z_t} \underbrace{\frac{p(m_i \mid z_{1:t-1}, x_{1:t-1})}{1 - p(m_i \mid z_{1:t-1}, x_{1:t-1})}}_{\text{recursive term}} \underbrace{\frac{1 - p(m_i)}{p(m_i)}}_{\text{prior}} \rightarrow Y$$

$$p(x) = \frac{1}{1 + \frac{1}{Y}}$$

For reasons of efficiency, one performs the calculations in the log odds notation

//important

$$l(m_i \mid z_{1:t}, x_{1:t}) = \log \left(\frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} \right)$$

$$l(x) = \log \frac{p(x)}{1 - p(x)}$$

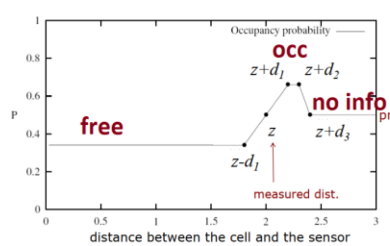
//important

$$l(m_i \mid z_{1:t}, x_{1:t}) = \underbrace{l(m_i \mid z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i \mid z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}}$$

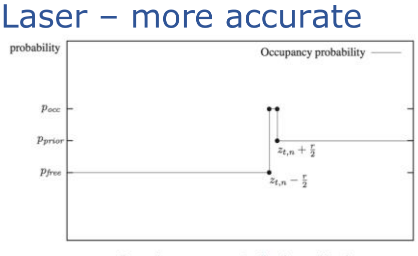
- or in short

$$l_{t,i} = \text{inv\_sensor\_model}(m_i, x_t, z_t) + l_{t-1,i} - l_0$$

Sonar



Laser – more accurate



§ Mapping with known poses is easy
§ Log odds model is fast to compute
§ No need for predefined features

Alternative: Counting Model / Reflection Probability Maps

§ For every cell count
§ hits(x,y): number of cases where a beam ended at <x,y>
§ misses(x,y): number of cases where a beam passed through <x,y>
// this bel called probability of reflection of the cell

$$Bel(m^{[xy]}) = \frac{\text{hits}(x,y)}{\text{hits}(x,y) + \text{misses}(x,y)}$$

//zoooom

- Maximum range reading: $\zeta_{t,n} = 1$
- Beam reflected by an object: $\zeta_{t,n} = 0$

$m_{f(x_t,n,z_{t,n})}$

max range: "first $z_{t,n}$-1 cells covered by the beam must be free"

$$p(z_{t,n} \mid x_t, m) = \begin{cases} \prod_{k=0}^{z_{t,n}-1} (1 - m_{f(x_t,n,k)}) & \text{if } \zeta_{t,n} = 1 \\ m_{f(x_t,n,z_{t,n})} \prod_{k=0}^{z_{t,n}-1} (1 - m_{f(x_t,n,k)}) & \zeta_{t,n} = 0 \end{cases}$$

otherwise: "last cell reflected beam, all others free"

Computing the Most Likely Map

- Compute values for m that maximize
  $m^* = \text{argmax}_m P(m \mid z_1, \cdots, z_t, x_1, \cdots, x_t)$
- Assuming a uniform prior probability for P(m), this is equivalent to maximizing:

$$\begin{aligned} m^* &= \text{argmax}_m P(z_1, \cdots, z_t \mid m, x_1, \cdots, x_t) \\ &= \text{argmax}_m \prod_{t=1}^{T} P(z_t \mid m, x_t) \quad \text{since } z_t \text{ independent and only depend on } x_t \\ &= \text{argmax}_m \sum_{t=1}^{T} \ln P(z_t \mid m, x_t) \end{aligned}$$

55

Computing the Most Likely Map

$$m^* = \text{argmax}_m \sum_{j=1}^{J} \sum_{t=1}^{T} \sum_{n=1}^{N} \Big(\underbrace{I(f(x_t, n, z_{t,n}) = j)}_{\text{"beam } n \text{ ends in cell } j\text{"}} \cdot (1 - \zeta_{t,n}) \cdot \ln m_j + \sum_{k=0}^{z_{t,n}-1} \underbrace{I(f(x_t, n, k) = j)}_{\text{"beam } n \text{ traversed cell } j\text{"}} \cdot \ln(1 - m_j) \Big)$$

// first is #hits , 2nd is #miss

$$\alpha_j = \sum_{t=1}^{T} \sum_{n=1}^{N} I(f(x_t, n, z_{t,n}) = j) \cdot (1 - \zeta_{t,n})$$

$$\beta_j = \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=0}^{z_{t,n}-1} I(f(x_t, n, k) = j)$$

$$\text{m}^* = \text{argmax}_m \sum_{j=1}^{J} \left(\alpha_j \ln m_j + \beta_j \ln(1 - m_j) \right)$$

$$m_j = \frac{\alpha_j}{\alpha_j + \beta_j}$$

Difference between Occupancy Grid Maps and Counting
§ The counting model determines how often a cell reflects a beam
§ The occupancy model represents whether or not a cell is occupied by an object
§ Although a cell might be occupied by an object, the reflection probability of this object might be very small (not sure but may be reflection maps better with glass)

LEC8

-> Mapping With Raw Odometry -> baaaad

§ Scan matching: **incrementally align two scans** or **a scan to a map**
§ Locally consistent estimates
§ But: Often scan matching is not sufficient to
build large consistent maps

$$x_t^* = \underset{x_t}{\operatorname{argmax}} \{p(z_t \mid x_t, m_{t-1})\; p(x_t \mid u_{t-1}, x_{t-1}^*)\}$$

current measurement
robot motion
map constructed so far

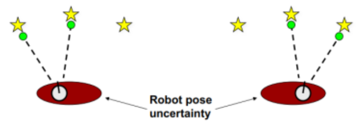
Various Different Ways to Realize Scan Matching
§ Scan-to-scan
§ Scan-to-map
§ Map-to-map
§ Iterative closest point (ICP)
§ Feature-based,…..

SLAM
SLAM is hard(chicken egg), because Errors in map and pose estimates **correlated**
§ a map is needed for localization and
§ a good pose estimate is needed for mapping

SLAM Applications
§ At home: vacuum cleaner, lawn mower
§ Surveillance with unmanned air vehicles
§ Underwater: reef monitoring
§ Underground: exploration of mines
§ Space: terrain mapping

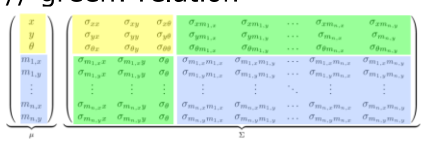
Data Association

- Picking **wrong** data associations can have **catastrophic** consequences (divergence)

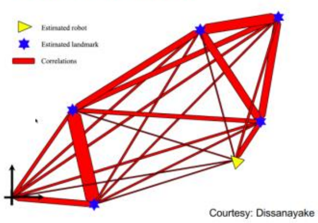


EKF SLAM
§ Estimate robot's pose and landmark locations
§ Assumption: known correspondences-> no data associations as it can not deal with it -> uni model(only one peak)

State Representation
§ Map with *n* landmarks: (3+2*n*)-dimensional Gaussian
// yellow : pose
// blue: map
// green: relation



Over time, the landmark estimates become **fully correlated**



- What if we neglected cross-correlations?

$$\Sigma_k = \begin{bmatrix} \Sigma_R & 0 & \cdots & 0 \\ 0 & \Sigma_{M_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Sigma_{M_n} \end{bmatrix} \qquad \Sigma_{RM_i} = \mathbf{0}_{3\times 2} \qquad \Sigma_{M_i M_{i+1}} = \mathbf{0}_{2\times 2}$$

- Landmark and robot uncertainties would become overly optimistic

Loop Closing/closure
§ Recognizing an already mapped area
§ Data association under
§ high ambiguity
§ possible environment symmetries
§ **Uncertainties collapse after a loop closure** (whether the closure was correct or not)
§ However, wrong loop closures lead to filter divergence

EKF SLAM Complexity

§ Cubic complexity depends only on the measurement dimensionality
§ Cost per step: dominated by the number of landmarks:

$$O(n^2)$$

§ Memory consumption:

$$O(n^2)$$

§ The EKF becomes computationally intractable for large maps!

Summary: EKF SLAM
§ The first SLAM solution
§ Can diverge if non-linearities are large (and the reality is non-linear...)
§ Can deal only with a single mode
§ Successful in medium-scale scenes
§ Data association has to be solved

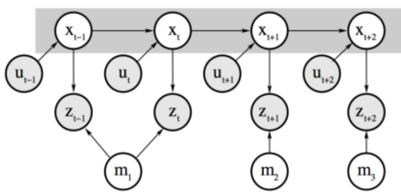
LEC9
Dimensionality Problem
§ PF are effective in low-dimensional spaces
§ The number of particles needed to represent
a posterior grows exponentially with the dimension of the state space!
Key Idea
§ Use the particle set only to model the robot's path (only poses not map)
§ For each sample, compute an individual map of landmarks

Rao-Blackwellization
§ Use factorization to exploit dependencies between variables
// poses are samples , so calculate it first then calculate map for each pose

poses map observations & movements

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = p(x_{0:t} \mid z_{1:t}, u_{1:t})\, p(m_{1:M} \mid x_{0:t}, z_{1:t})$$

path posterior map posterior



Landmark variables are all disconnected (i.e. independent) given the robot's path

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i \mid x_{0:t}, z_{1:t})$$

particle filter similar to MCL

2-dimensional EKFs!

FastSLAM

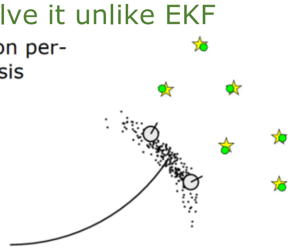
- Each landmark is represented by a 2x2 EKF
- Thus, each particle has to maintain M individual EKFs

Particle 1	x, y, θ	Landmark 1	Landmark 2	...	Landmark M
Particle 2	x, y, θ	Landmark 1	Landmark 2	...	Landmark M
⋮					
Particle N	x, y, θ	Landmark 1	Landmark 2	...	Landmark M

Key Steps of FastSLAM 1.0
1- Sample a new pose for each particle
2- Compute the particle weights
3- Update belief of observed landmarks
(EKF update rule)
4- Resample

Data Association Problem
//can solve it unlike EKF

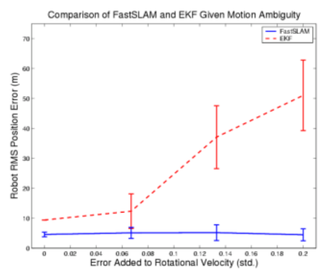
Decisions on per-particle basis



Options:
§ Pick the most probable match
§ Pick a random association weighted by the observation likelihoods
§ If the probability for an assignment is too low, generate a new landmark

-> EKF is Robust against motion noise

Results (w. Motion Uncertainty)



->Complexity:

N = Number of particles
M = Number of map features

$$\mathcal{O}(NM)$$

Summary: FastSLAM
§ Feature-Based SLAM with particle filter
§ Rao-Blackwellization: model the robot's path by sampling and compute the landmarks given the robot poses
§ Data association on per-particle basis
§ Robust to ambiguities in the data association
§ Scales well (1 million+ features)
§ No uncertainty about the robot pose -> each particle represent pose

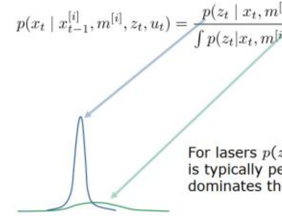
Problem
§ Too many samples are needed to sufficiently model the motion noise
§ Increasing the number of samples is difficult as each map is quite large
§ **Idea:** Improve the pose estimate **before** applying the particle filter, Pre-correct short odometry sequences using scan matching and use them as input to FastSLAM

LEC10

FastSLAM 2.0
1- Perform scan matching for each particle using its own map
2- Fit a Gaussian by sampling points around the maximum of scan matcher
3- Calculate importance weights
4- Selective Resampling

As a result:
§ More precise sampling
§ Less particles needed
§ More accurate maps

$$p(x_t \mid x_{t-1}^{[i]}, m^{[i]}, z_t, u_t) = \frac{p(z_t \mid x_t, m^{[i]})\, p(x_t \mid x_{t-1}^{[i]}, u_t)}{\int p(z_t \mid x_t, m^{[i]})\, p(x_t \mid x_{t-1}^{[i]}, u_t)\, dx_t}$$



For lasers $p(z_t \mid x_t, m^{[i]})$ is typically peaked and dominates the product

18

//leads to

$$p(x_t \mid x_{t-1}^{[i]}, m^{[i]}, z_t, u_t) \simeq \frac{p(z_t \mid x_t, m^{[i]})}{\int_{x_t \in \{x \mid p(z_t \mid x, m^{[i]}) > \epsilon\}} p(z_t \mid x_t, m^{[i]})\, dx_t}$$

Resampling
§ Resampling at each step limits the "memory"
§ Resampling is necessary to achieve convergence
§ Resampling is dangerous, since important samples might get lost ("particle depletion")
§ Resampling makes only sense if particle weights differ significantly

Selective Resampling

§ neff describes "the inverse variance of the normalized particle weights"
§ For equal weights, the sample approximation is close to the target, no resample is needed

$$n_{eff} = \frac{1}{\sum_i \left(w_t^{[i]}\right)^2}$$

$$\frac{1}{\sum_i \left(w_t^{[i]}\right)^2} \overset{?}{<} N/2$$

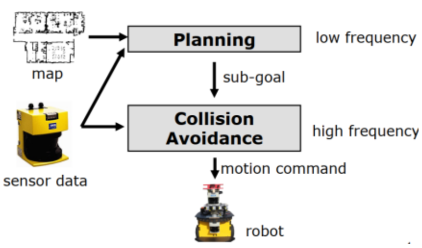
if yes , resample

Summary:
Grid-Based FastSLAM (2.0)
§ The ideas of FastSLAM can also be applied in the context of grid maps
§ Similar to scan-matching on a per-particle base
§ Selective resampling reduces the risk of particle depletion
§ Substantial reduction of the required number of particles

LEC11

Motion Planning: Requirements
§ Collision-free trajectory from the current robot pose to a goal pose
§ The robot should reach the goal location as fast as possible
§ The robot must react to unforeseen obstacles fast and reliably

Two-Layered Architecture (wheeled robots)



Dynamic Window Approach

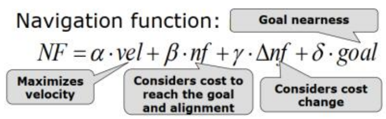
§ Determine collision-free trajectories using geometric operations (v and ω)

->A velocity is **admissible** if the robot is able to stop before colliding with an obstacle
-> Velocities that are **reachable** by acceleration within the time period *t*

//note intersection

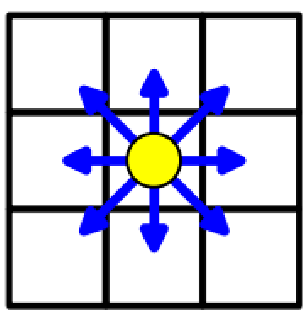
V_s = all possible velocities of the robot
V_a = obstacle free area (light grey)
V_d = velocities reachable within a certain time frame based on possible accelerations
Search space: $V_r = V_s \cap V_a \cap V_d$

§ How to choose <v,ω>?
§ Use a **heuristic navigation fuction**
§ Try to minimize travel time according to the principle: “**driving fast** in the **right direction**”



§ quick reaction
§ Low CPU power
§ collision-free path
§ real-world systems
§ Resulting trajectories sometimes suboptimal
§ Local minima might prevent the robot from reaching the goal location
§ DWAs might not be able to reach the goal location
§ Robot does not slow down early enough to enter the doorway

Robot Path Planning with A*
§ Find the shortest 2D path in a given static map
§ 2D grid map (=states)
§ 8-connected neighborhood (=actions)
§ Consider move cost and occupancy value



Reminder: A*
§ *g(n):* actual cost from the initial state to *n*
§ *h(n)*: estimated cost from *n* to the goal
§ *f(n) = g(n) + h(n)*: estimated cost of the cheapest solution through *n*
§ Let h*(n) be the actual cost of the optimal path from *n* to the goal
§ *h* is admissible if it holds for all *n*: *h(n) <= h*(n)*
§ A* yields the optimal path if *h* is admissible
(the straight-line distance in the Euclidean space is admissible)

Deterministic Value Iteration
§ To compute the shortest path from every state to one goal state - similar to Dijkstra
§ the optimal heuristics for A*, which is needed for re-planning, e.g., in case of nonstatic obstacles, also for localization errors

Problems when Using A* on Grid Maps
§ Moving on the shortest path often guides the robot on a trajectory close to obstacles
§ What if the robot is slightly delocalized?
§ What if commands are only inaccurately executed?

Solution: Convolution of the Grid Map
§ Convolution, e.g, with a Gaussian kernel to “blur” the map
§ Obstacles are assumed to be bigger than in reality
§ Perform an A* search on convolved map
§ As a result, the robot increases its distance to obstacles and moves on a short path

Convolution

- The convolution of an occupancy map is defined as:

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

$$P(occ_{x_0,y}) = \frac{2}{3} \cdot P(occ_{x_0,y}) + \frac{1}{3} \cdot P(occ_{x_1,y})$$

$$P(occ_{x_{n-1},y}) = \frac{1}{3} \cdot P(occ_{x_{n-2},y}) + \frac{2}{3} \cdot P(occ_{x_{n-1},y})$$

- This is done for each row and each column of the map
- Named "Gaussian blur"

5D Planning

§ **A* search in the full 5D <x,y,θ,v,ω>- configuration space**
§ Considers the robot's kinematic constraints directly
§ Considers driving time and distance to obstacles in the cost function

//constraint given in slides

$$|v_1-v_2| < a_v, \ |\omega_1-\omega_2| < a_\omega$$

Problem:
The search space is too huge to be explored within the time constraints

Solution: Restrict the full search space

Main Steps of 5D Path Planning

1. Update the grid map based on sensory input
2. Use A* to find a path in the <x,y>- space using the updated grid map
3. Determine a restricted 5D configuration space based on step 2
4. Find a trajectory by planning in this restricted <x,y,θ,v,ω>-space

Updating the Grid Map

§ Add newly detected obstacles
§ Convolve the map to increase security distance

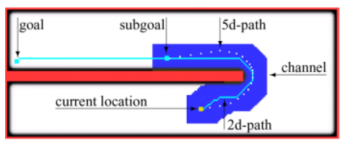
Find a Path in the 2D Space

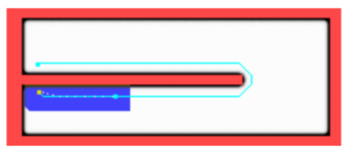
§ Use heuristics based on a deterministic value iteration within the static map

Restricting the Search Space

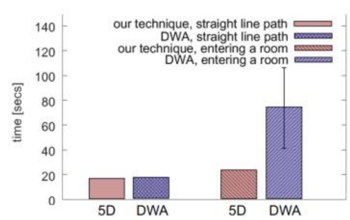
Assumption: The projection of the optimal 5D path onto the <x,y>-space lies close to the optimal 2D path
Idea: Construct a restricted search space ("channel") based on the 2D path

§ Choose a sub-goal lying on the 2D path within the channel
§ Use A* in the restricted 5D space to find a sequence of steering commands to reach the sub-goal
§ Heuristics to estimate cell costs:
deterministic 2D value iteration within the channel

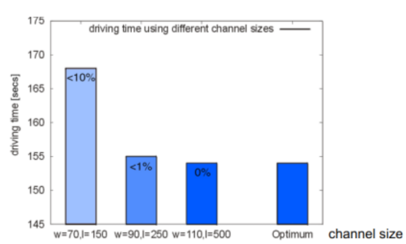




//both are the same in straight lines , while in entering a room (narrow) it differs



//channel size increases , we are close to optimal solution



Summary : path planing

§ Robust navigation requires path planning and collision avoidance
§ Collision avoidance considers the robot's kinematic constraints and generates velocities
§ Planning in the 5D space shows better results than the pure DWA in a variety of situations
§ Using the 5D approach the quality of the trajectory scales with the computational resources available -> channel size
§ Still DWA often used in practice