Cognitive Robotics Topics:

1. Basis
   - Probability Theory
   - Bayes Filtering
2. Gaussian Filters
   - Kalman Filters
   - Extended Kalman Filter
   - Unscented Kalman Filter
   - Multi-hypothesis Tracking
3. Motion Models
   - Odometry-based Model
   - Velocity-based Model
4. Sensor Models
   - Beam-based Model
   - End-Point Model
5. Nonparametric Filters
   - Discrete Filters
   - Particle Filter
6. Mapping with known poses
   - Occupancy Grid Mapping
   - Reflection Probability Mapping
7. SLAM
   - Scan Matching
   - EkF SLAM
   - FastSLAM
   - Grid-Based FastSlam
   - FastSLAM 2.0
   - GraphSLAM
8. Motion Planning
   - Dynamic Window Approach
   - Global Path Planning with A^*
   - 5D Planning
   - Aborting A^*

1. Basis

Probability Theory

Conditional Probability:

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

Law of Total Probability:

Discrete Case:

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

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

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

Continuous Case:

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

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

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

Bayes Formula:

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

Normalization:

$$P(x|y) = \frac{P(y|x)\,P(x)}{P(y)} = \eta \cdot P(y|x)\,P(x)$$

$$\eta = \frac{1}{\sum P(y|x)P(x)}$$

Ratio:

$$\begin{aligned} \frac{p(x)}{1-p(x)} &= Y \\ p(x) &= Y - Y\,p(x) \\ p(x)\,(1+Y) &= Y \\ p(x) &= \frac{Y}{1+Y} \\ p(x) &= \frac{1}{1+\frac{1}{Y}} \end{aligned}$$

Log odds:

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

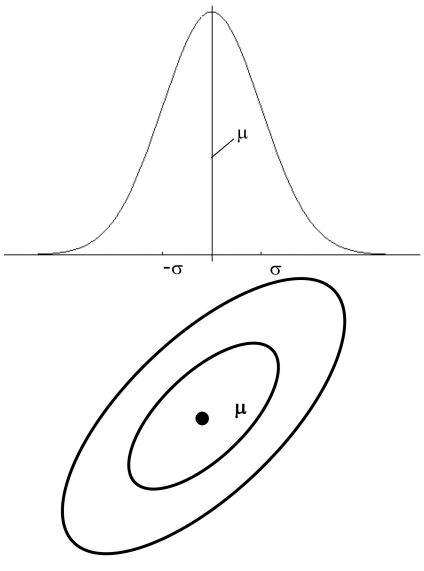
$$p(x) = \frac{1}{1+\frac{1}{\exp l(x)}}$$

Gaussians:

Univariate:

$$p(x) \sim N(\mu, \sigma^2):$$

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



Multivariate:

$$p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}):$$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

Properties

Linear Transformation:

$$\left.\begin{aligned} X &\sim N(\mu, \sigma^2) \\ Y &= aX + b \end{aligned}\right\} \Rightarrow Y \sim N(a\mu + b, a^2\sigma^2)$$

Product Rule:

$$\left.\begin{aligned} X_1 &\sim N(\mu_1, \sigma_1^2) \\ X_2 &\sim N(\mu_2, \sigma_2^2) \end{aligned}\right\} \Rightarrow p(X_1) \cdot p(X_2) \sim N\left(\frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2}\mu_1 + \frac{\sigma_1^2}{\sigma_1^2+\sigma_2^2}\mu_2,\ \frac{1}{\sigma_1^{-2}+\sigma_2^{-2}}\right)$$

Sampling

Gaussian:

$$\varepsilon_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$$

Algorithm:

1. Algorithm **sample_normal_distribution**(b):
2. return $\frac{1}{2}\sum_{i=1}^{12} rand(-b, b)$ — Application of central limit theorem; Uniform distribution

Triangle:

$$\varepsilon_{\sigma^2}(x) = \begin{cases} 0 \text{ if } |x| > \sqrt{6\sigma^2} \\ \frac{\sqrt{6\sigma^2} - |x|}{6\sigma^2} \end{cases}$$

Algorithm:

1. Algorithm **sample_triangular_distribution**(b):
2. return $\frac{\sqrt{6}}{2}[\text{rand}(-b, b) + \text{rand}(-b, b)]$

Rejection Sampling:

1. Algorithm **sample_distribution**(f, b):
2. repeat
3. $\quad x = \text{rand}(-b, b)$
4. $\quad y = \text{rand}(0, \max\{f(x) \mid x \in (-b, b)\})$
5. until $(y \leq f(x))$
6. return x

Importance Sampling:

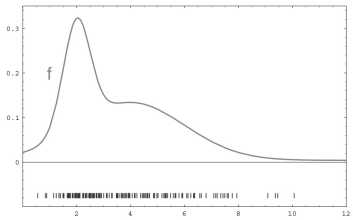
Goal: Sample from target distribution f

Steps:

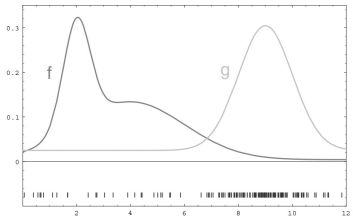
1. Sample from proposal distribution g
2. Weight samples to accord for the difference.

$$w = \frac{f(x)}{g(x)}$$

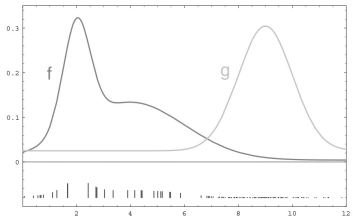
Goal:



1.



2.



Bayes Filtering

Bayes Filtering Formula:

$$Bel(x) = \eta\, P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

Correction | Prediction

Prediction:

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

Action Model

Correction:

$$Bel(x_t) = \eta\, P(z_t \mid x_t)\, \overline{Bel}(x_t)$$

Sensor Model

Derivation:

$$\begin{aligned}
Bel(x_t) &= P(x_t \mid u_1, z_1, \ldots, u_t, z_t) \\
&= \eta\, P(z_t \mid x_t, u_1, z_1, \ldots, u_t)\, P(x_t \mid u_1, z_1, \ldots, u_t) && \text{| Bayes} \\
&= \eta\, P(z_t \mid x_t)\, P(x_t \mid u_1, z_1, \ldots, u_t) && \text{| Markov} \\
&= \eta\, P(z_t \mid x_t) \int P(x_t \mid u_1, z_1, \ldots, u_t, z_t, x_{t-1})\, P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\, dx_{t-1} && \text{| Total Prob} \\
&= \eta\, P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\, Bel(x_{t-1})\, dx_{t-1} && \text{| Markov}
\end{aligned}$$

Algorithm:

1. Algorithm **Bayes_filter**(*Bel(x), d*):
2. $\eta = 0$
3. If *d* is a **perceptual** data item *z* then
4. For all *x* do
5. $Bel'(x) = P(z \mid x) Bel(x)$
6. $\eta = \eta + Bel'(x)$
7. For all *x* do
8. $Bel'(x) = \eta^{-1} Bel'(x)$
9. Else if *d* is an **action** data item *u* then
10. For all *x* do
11. $Bel'(x) = \int P(x \mid u, x')\, Bel(x')\, dx'$
12. Return *Bel'(x)*

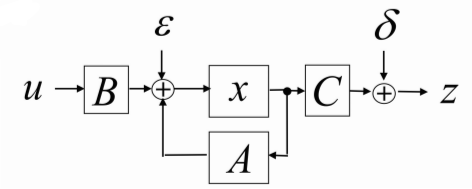
Assumptions:

1. Markov assumption
2. Static world (only actions change state)
3. Independent noise
4. Perfect model, no approximation error

2. Gaussian Filters

Kalman Filter

The Kalman Filter can be used to make predictions about the state of a linear dynamic system.



Filtering Formula:

Prediction:

$$\overline{Bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B u_t \\ \bar{\Sigma}_t = A_t \Sigma_t A_t^T + R_t \end{cases}$$

Correction:

$$Bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t (z_t - C \bar{\mu}_t) \\ \Sigma_t = (1 - K_t C_t) \bar{\Sigma}_t \end{cases}$$

$$\text{with: } K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

Components:

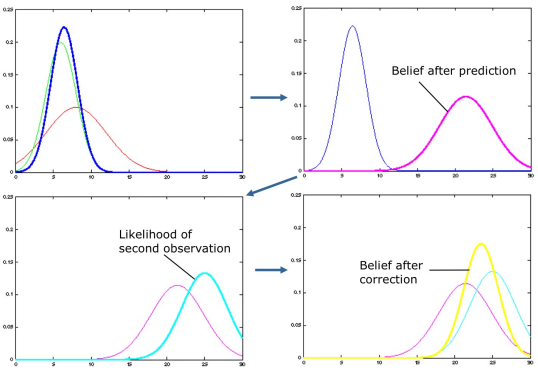
- A_t: Dynamic Matrix $(n \times n)$
- B_t: Control Matrix $(n \times 1)$
- C_t: Measurement Matrix $(k \times n)$
- ε: Dynamic error with covariance R
- δ: Measurement Noise with covariance Q
- K: Kalman Gain

Algorithm:

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2. Prediction:
3. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$ // apply motion model
4. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
5. Correction:
6. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$ // compute Kalman gain
7. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$ // compare expected with
8. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$ observed measurement
9. Return μ_t, Σ_t

Ideas:

+ highly efficient with $O(k^{2.376} + n^2)$
+ Optimal for linear Gaussian systems
- Can not deal with non-linearities



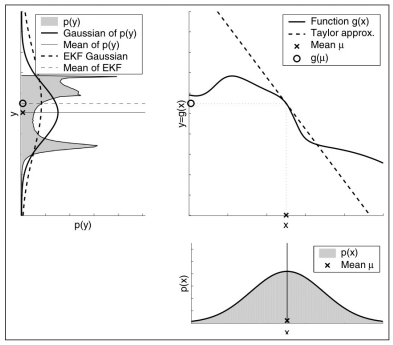
Extended Kalman Filter

The EKF extends the approach to non-linear systems by linearizing it before it is fed to the Kalman Filter

Linearization: First-order Taylor Series Expansion

Robot Motion: $x_t = g(u_t, x_{t-1})$

Measurements: $z_t = h(x_t)$

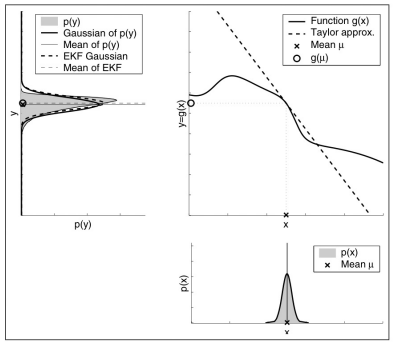


Approximation:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1})$$

$$h(x_t) \approx h(\mu_t) + G_t(x_t - \bar{\mu}_t)$$

Jacobian



Filtering Formula:

Prediction:

$$\overline{Bel}(x_t) = \begin{cases} \bar{\mu}_t = g(u_t, x_{t-1}) \\ \bar{\Sigma}_t = G_t \Sigma_t G_t^T + R_t \end{cases}$$

Correction:

$$Bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \\ \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t \end{cases}$$

with: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

Algorithm:

Prediction:

2. $G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} = \begin{pmatrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{pmatrix}$ Jacobian of g w.r.t. location

3. $V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t} = \begin{pmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial \omega_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial \omega_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta'}{\partial \omega_t} \end{pmatrix}$ Jacobian of g w.r.t. control

4. $M_t = \begin{pmatrix} (\alpha_1 |v_t| + \alpha_2 |\omega_t|)^2 & 0 \\ 0 & (\alpha_3 |v_t| + \alpha_4 |\omega_t|)^2 \end{pmatrix}$ Motion noise

5. $\bar{\mu}_t = g(u_t, \mu_{t-1})$ Predicted mean

6. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$ Predicted covariance

Correction:

` (distance, angle to landmark)

2. $\hat{z}_t = \begin{pmatrix} \sqrt{(m_x - \bar{\mu}_{t,x})^2 + (m_y - \bar{\mu}_{t,y})^2} \\ \operatorname{atan2}(m_y - \bar{\mu}_{t,y}, m_x - \bar{\mu}_{t,x}) \ominus \bar{\mu}_{t,\theta} \end{pmatrix}$ Predicted measurement mean

4. $H_t = \frac{\partial h(\bar{\mu}_t, m)}{\partial x_t} = \begin{pmatrix} \frac{\partial r_t}{\partial \bar{\mu}_{t,x}} & \frac{\partial r_t}{\partial \bar{\mu}_{t,y}} & \frac{\partial r_t}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,x}} & \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,y}} & \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,\theta}} \end{pmatrix}$ Jacobian of h w.r.t. location

5. $Q_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$ Measurement noise

6. $S_t = H_t \bar{\Sigma}_t H_t^T + Q_t$ Pred. measurement covariance

7. $K_t = \bar{\Sigma}_t H_t^T S_t^{-1}$ Kalman gain

8. $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$ Updated mean

9. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$ Updated covariance

Ideas:

+ highly efficient even when assumptions are violated
- Quality of the approximation relies on the certainty of the prior
- Diverge if non-linearities grow large
- uses linearization on a single point (far from mean)

Unscented Kalman Filter

This approach linearizes a non-linear system by approximation using multiple Sigma points.

Linearization: Recover Gaussian through Sigma points

1. **Choose Sigma points**

Sigma points:

$$\chi^0 = \mu$$

$$\chi = \mu \pm \left(\sqrt{(n+\lambda)\Sigma}\right)_i \quad \text{for } i = 1, \dots, 2n$$

$$\lambda = \alpha^2 (n + \kappa) - n$$

Weights:

Mean:

$$w_m^0 = \frac{\lambda}{n+\lambda}$$

Covariance:

$$w_c^0 = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$$

$$w_m^i = w_c^i = \frac{1}{2(n+\lambda)}$$

Parameters:

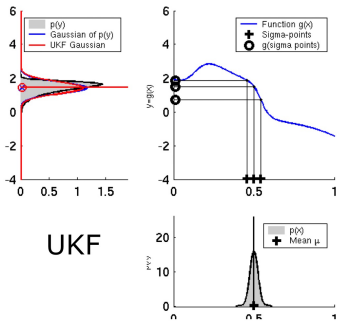
- n: number of dimensions
- α: spread of Sigma points ($0 < \alpha \ll 1$)
- κ: scaling parameter ($\kappa = 0$)
- β: prior knowledge on the distribution ($\beta = 2$ for Gaussian)

2. **Pass Sigma points through non-linear functions**

$$\psi^i = g(\chi^i)$$

3. **Recover Gaussian**

$$\mu' = \sum_{i=0}^{2n} w_m^i \psi^i \qquad \Sigma' = \sum_{i=0}^{2n} w_c^i (\psi^i - \mu')(\psi^i - \mu')^T$$



UKF

Algorithm:

Prediction:

$$M_t = \begin{pmatrix} (\alpha_1 |v_t| + \alpha_2 |\omega_t|)^2 & 0 \\ 0 & (\alpha_3 |v_t| + \alpha_4 |\omega_t|)^2 \end{pmatrix}$$ Motion noise — Depends on forward speed and rotational speed

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$ Measurement noise

$$\mu_{t-1}^a = \begin{pmatrix} \mu_{t-1}^T & (0\,0)^T & (0\,0)^T \end{pmatrix}$$ Augmented state mean — x,y,θ , motion noise, measurement noise

$$\Sigma_{t-1}^a = \begin{pmatrix} \Sigma_{t-1} & 0 & 0 \\ 0 & M_t & 0 \\ 0 & 0 & Q_t \end{pmatrix}$$ Augmented covariance 7 x 7

$$\chi_{t-1}^a = \begin{pmatrix} \mu_{t-1}^a & \mu_{t-1}^a + \gamma\sqrt{\Sigma_{t-1}^a} & \mu_{t-1}^a - \gamma\sqrt{\Sigma_{t-1}^a} \end{pmatrix}$$ 15 Sigma points

$$\bar{\chi}_t^x = g(u_t + \chi_t^u, \chi_{t-1}^x)$$ Prediction of sigma points

$$\bar{\mu}_t = \sum_{i=0}^{2L} w_m^i \chi_{i,t}^x$$ Predicted mean

$$\bar{\Sigma}_t = \sum_{i=0}^{2L} w_c^i (\chi_{i,t}^x - \bar{\mu}_t)(\chi_{i,t}^x - \bar{\mu}_t)^T$$ Predicted covariance

Correction:

$$\bar{Z}_t = h(\bar{\chi}_t^x) + \chi_t^z$$ Prediction of Measurement sigma points

$$\hat{z}_t = \sum_{i=0}^{2L} w_m^i \bar{Z}_{i,t}$$ Predicted measurement mean

$$S_t = \sum_{i=0}^{2L} w_c^i (\bar{Z}_{i,t} - \hat{z}_t)(\bar{Z}_{i,t} - \hat{z}_t)^T$$ Pred. measurement covariance

$$\Sigma_t^{x,z} = \sum_{i=0}^{2L} w_c^i (\bar{\chi}_{i,t}^x - \bar{\mu}_t)(\bar{Z}_{i,t} - \hat{z}_t)^T$$ Cross-covariance — Between state and observation

$$K_t = \Sigma_t^{x,z} S_t^{-1}$$ Kalman gain

$$\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$$ Updated mean

$$\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^T$$ Updated covariance

Advantages to EKF:

- Highly efficient: same complexity as EKF
- Better linearization: Accurate in first two terms of Taylor expansion
- Derivative free

Multi-Hypothesis Tracking

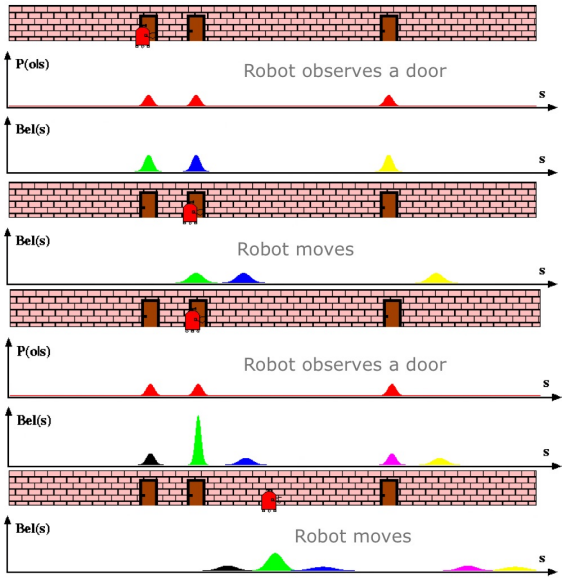
Multiple hypothesis are established and tracked by a Kalman Filter each.

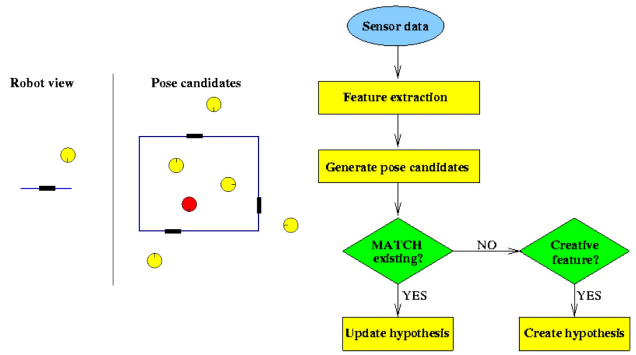
Data Association: Map observation to corresponding hypothesis

Hypothesis Management: Number of hypotheses grows fast. Thus, pruning is required

Compute probability of each hypothesis:
→Prune if the probability grows small

$$P(H_i \mid z) = \frac{P(z \mid H_i)P(H_i)}{P(z)}$$





3. Motion Models

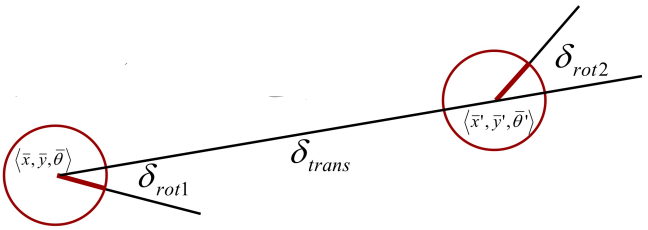
Odometry-based Model

Odometry Model

Robot Pose: $\langle x, y, \theta \rangle$

Motion: $u = \langle \delta_{rot_1}, \delta_{rot_2}, \delta_{trans} \rangle$

$$\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$$
$$\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$$
$$\delta_{rot2} = \bar{\theta}'-\bar{\theta} - \delta_{rot1}$$



Noise Model:

$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$
$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$
$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|}$$

Algorithm:

1. Algorithm **motion_model_odometry(x, x', u)**
2. $\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$
3. $\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$
4. $\delta_{rot2} = \bar{\theta}'-\bar{\theta} - \delta_{rot1}$
5. $\hat{\delta}_{trans} = \sqrt{(x'-x)^2 + (y'-y)^2}$
6. $\hat{\delta}_{rot1} = \text{atan}\,2(y'-y, x'-x) - \theta$
7. $\hat{\delta}_{rot2} = \theta'-\theta - \hat{\delta}_{rot1}$
8. $p_1 = \text{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 |\hat{\delta}_{\text{rot1}}| + \alpha_2 \hat{\delta}_{\text{trans}})$
9. $p_2 = \text{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4 (|\hat{\delta}_{\text{rot1}}| + |\hat{\delta}_{\text{rot2}}|))$
10. $p_3 = \text{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 |\hat{\delta}_{\text{rot2}}| + \alpha_2 \hat{\delta}_{\text{trans}})$
11. return $p_1 \cdot p_2 \cdot p_3$

odometry values $\mathbf{u} = (\bar{\mathbf{x}}, \bar{\mathbf{x}}')$

Poses in robot internal coordinate system

values of interest (x,x')

Algorithm: sample_motion_model(u, x):

$\mathbf{u} = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$, $\mathbf{x} = \langle x, y, \theta \rangle$ — old pose

1. $\hat{\delta}_{rot1} = \delta_{rot1} + \text{sample}(\alpha_1 |\delta_{rot1}| + \alpha_2 \delta_{trans})$
2. $\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3 \delta_{trans} + \alpha_4 (|\delta_{rot1}| + |\delta_{rot2}|))$
3. $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 |\delta_{rot2}| + \alpha_2 \delta_{trans})$
4. $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5. $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6. $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
7. Return $\langle x', y', \theta' \rangle$

sample_normal_distribution

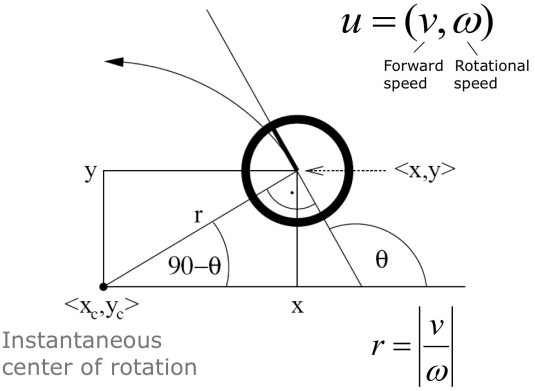
Velocity-based Model

Velocity Model

Robot Pose: $\langle x, y, \theta \rangle$

Motion: $u = (v, \omega)$

Forward speed / Rotational speed



Instantaneous center of rotation

$$r = \left| \frac{v}{\omega} \right|$$

Calculation of rotational Center:

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -r\sin\theta \\ r\cos\theta \end{pmatrix} = \begin{pmatrix} \frac{x+x'}{2} + \mu(y-y') \\ \frac{y+y'}{2} + \mu(x'-x) \end{pmatrix}$$

with $\mu = \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$

Turning Radius:

$$r = \left| \frac{v}{\omega} \right|$$

Algorithm:

Algorithm motion_model_velocity(x_t, u_t, x_{t-1}): ($u = (v, \omega)$)

$\mu = \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$

$x^* = \frac{x+x'}{2} + \mu(y-y')$ // instantaneous center of rotation

$y^* = \frac{y+y'}{2} + \mu(x'-x)$

$r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2}$ // distance to center

$\Delta\theta = \text{atan2}(y'-y^*, x'-x^*) - \text{atan2}(y-y^*, x-x^*)$

$\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$

$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$ // compute motion error (deviation from control u)

$\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{\omega}$

return $\textbf{prob}(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot \textbf{prob}(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|) \cdot \textbf{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Algorithm sample_motion_model_velocity(u_t, x_{t-1}): ($u = (v, \omega)$)

$\hat{v} = v + \textbf{sample}(\alpha_1|v| + \alpha_2|\omega|)$

$\hat{\omega} = \omega + \textbf{sample}(\alpha_3|v| + \alpha_4|\omega|)$

$\hat{\gamma} = \textbf{sample}(\alpha_5|v| + \alpha_6|\omega|)$

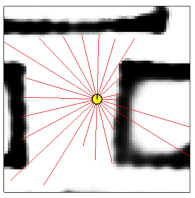
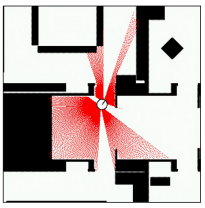
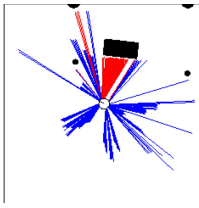
$x' = x - \frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}} sin(\theta + \hat{\omega}\Delta t)$

$y' = y + \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}} cos(\theta + \hat{\omega}\Delta t)$

$\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$

return $x_t = (x', y', \theta')^T$

4. Sensor Models

Beam-based Sensor Model

For each measurement cast a ray and compute the expected measurement to determine the likelihood of the measurement.

Sensor Data: $z = \{z_1, \ldots, z_K\}$

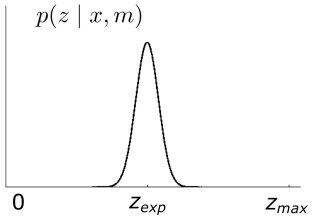
Assumption: Individual measurements are independent

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

Measurement Model

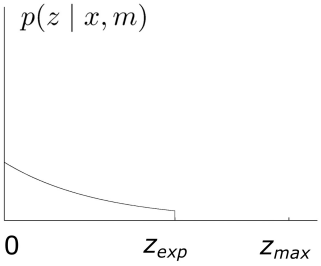
1. Beams reflected by known obstacles

$$p_{hit}(z \mid x, m) = \begin{cases} \frac{\eta}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z - z_{exp})^2}{2\sigma^2}} & \text{if } z \le z_{max} \\ 0 & \text{otherwise} \end{cases}$$

1. 

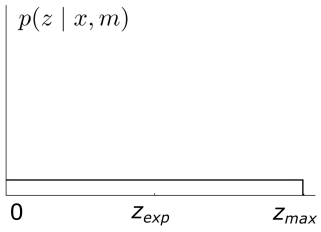
2. Beams reflected by people / objects

$$p_{unexp}(z \mid x, m) = \begin{cases} \eta\lambda e^{-\lambda z} & \text{if } z < z_{exp} \\ 0 & \text{otherwise} \end{cases}$$

2. 

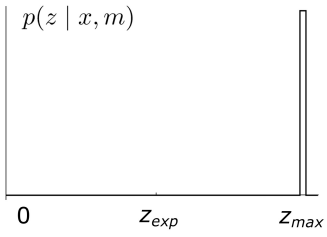
3. Random measurements

$$p_{rand}(z \mid x, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } z < z_{max} \\ 0 & \text{otherwise} \end{cases}$$

3. 

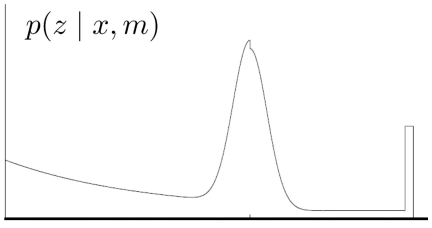
4. Maximum range measurements

$$p_{max}(z \mid x, m) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases}$$

4. 

Sensor Model:

$$p(z \mid x, m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\text{max}} \\ \alpha_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z \mid x, m) \\ p_{\text{unexp}}(z \mid x, m) \\ p_{\text{max}}(z \mid x, m) \\ p_{\text{rand}}(z \mid x, m) \end{pmatrix}$$

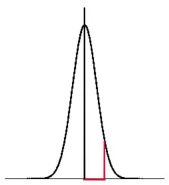


→ Parameters need to be approximated w.r.t the sensor

End-Point Model

Instead of casting complete rays, this approach solely estimates the likelihood of the end-point of the measurement. Using a Gaussian the probability of the end-point is evaluated using its distance to the closest obstacle

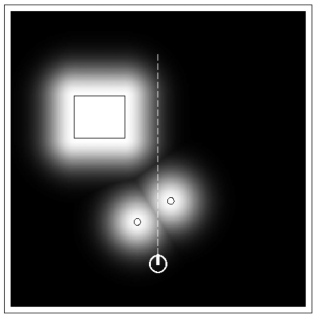
Likelihood: $Likelihood = \frac{1}{\sqrt{2\,\pi\sigma}} \exp \frac{-(d-0)^2}{2\sigma^2}$



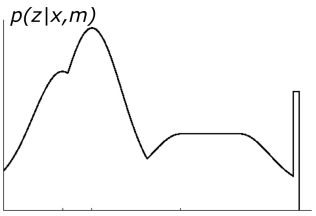
Gaussian to evaluate the distance

Measurement Model

1. Gaussian evaluating the distance to the closest obstacle
2. Uniform distribution for random measurements
3. Small uniform distribution for max. range measurements



measurement likelihood



measured length z along the indicated beam

Sensor Model:

Grid Map:

1. Build up Likelihood map
   The Likelihood map reveals the likelihood of an end-point.
2. Extract Likelihood for the end-points

Landmarks:

1. Compute Likelihood to given landmark

1. Algorithm **landmark_model**(z,x,m):
   $z = \langle i, d, \alpha \rangle, x = \langle x, y, \theta \rangle$ landmark *i*, map pose *x*
2. $\hat{d} = \sqrt{(m_x(i)-x)^2 + (m_y(i)-y)^2}$ ——— Expected distance
3. $\hat{\alpha} = \text{atan2}(m_y(i)-y, m_x(i)-x) - \theta$ ——— Expected angle
4. $p_{\text{det}} = \text{prob}(\hat{d}-d, \varepsilon_d) \cdot \text{prob}(\hat{\alpha}-\alpha, \varepsilon_\alpha)$ ——— Independence assumption, two Gaussians
5. Return p_{det}

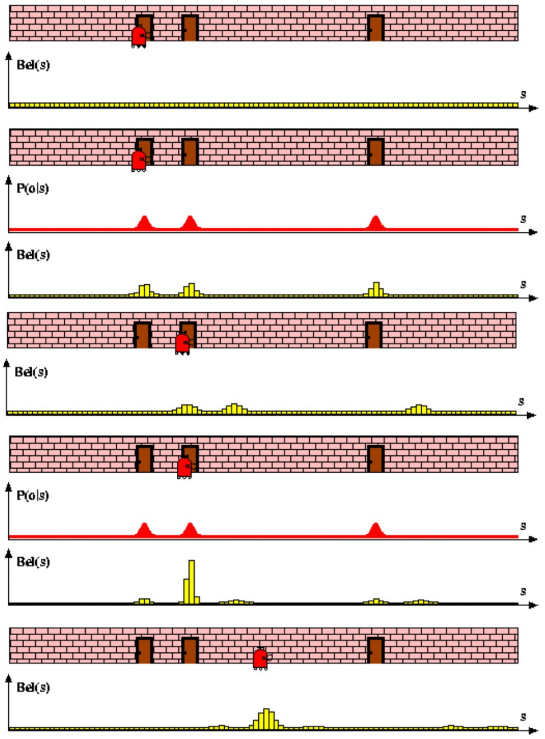
5. Non-parametric Filters

Discrete Filters (Grid Localization)

Represent multi-modal beliefs by discretizing the state space into a grid and propagate the probability for each grid cell

Algorithm:

1. Algorithm **Bayes_filter**(*Bel(x),d*):
2. $\eta=0$
3. If *d* is a **perceptual** data item *z* then
4. For all *x* do — sensor model, Ch. 5
5. $Bel'(x) = P(z \mid x) Bel(x)$
6. $\eta = \eta + Bel'(x)$
7. For all *x* do
8. $Bel'(x) = \eta^{-1} Bel'(x)$ — normalization
9. Else if *d* is an **action** data item *u* then
10. For all *x* do — sum over all **discrete** states
11. $Bel'(x) = \sum_{x'} P(x \mid u, x') \, Bel(x')$
12. Return *Bel'(x)* — motion model, Ch. 4



Computation: Extremly high memory and processing consumptions
→ only update relevant parts with a high probability
→ Update motion at lower frequency

Ideas:

+ can approximate multi-modal beliefs
+ perform global localization
- high memory and processing requirement
- accuracy depends on grid

Particle Filters (Monte Carlo Localization)

Particle Filters model a target distribution through a set of samples drawn from a proposal distribution and weighted according to its difference to the target distribution.

General Particle Filter:

1. Sample the particles using the proposal distribution
$$x_t^{[j]} \sim proposal(x_t \mid \ldots)$$
2. Compute the importance weights
$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})}$$
3. Resampling: Draw sample i with probability $w_t^{[i]}$ and repeat J times

Particle Set: $\mathcal{X} = \{\langle x^{[j]}, w^{[j]} \rangle\}_{j=1,\ldots,J}$

state hypothesis — importance weight

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

Monte Carlo Localization:

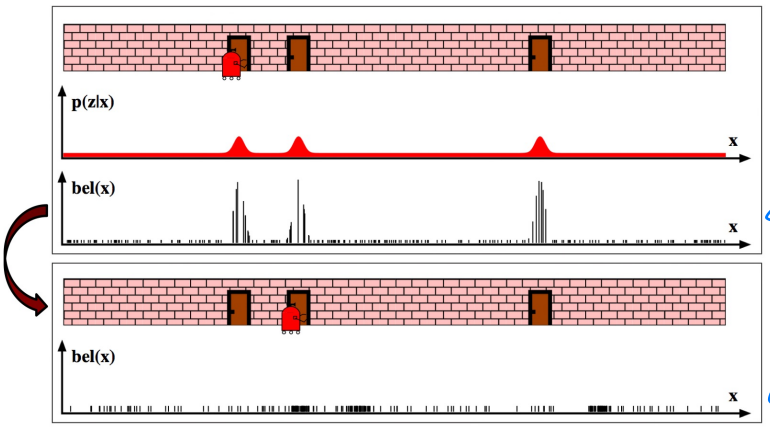
Particle Set: Each particle represents a pose hypothesis

Prediction: For each particle, sample a new pose from the motion model.
$$x_t^{[j]} \sim p(x_t \mid x_{t-1}^{[j]}, u_t)$$

Correction: Weigh samples according to sensor models
$$w_t^{[j]} \propto p(z_t \mid x_t^{[j]})$$

Derivation:
$$w_t^{[j]} = \frac{\text{target}}{\text{proposal}} = \frac{\eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})}{p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1})}$$
$$= \eta\, p(z_t \mid x_t)$$

Resampling: Resample from the set of particles with their respective probability



← Prediction

← Correction

Algorithm:

current set of particles

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do* — for all particles
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$ — prediction
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$ — correction
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do* — resampling of particles
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$ — new set of particles

Resampling:

1. Draw samples according to the probabilities
   → Distribution might be represented by a small number of particles
   → Eliminating samples with low probability makes it unable to recover from bad predictions

2. Low Variance Sampling

Low_variance_resampling$(\mathcal{X}_t, \mathcal{W}_t)$**:**

1: $\bar{\mathcal{X}}_t = \emptyset$
2: $r = \text{rand}(0; J^{-1})$ initialization
3: $c = w_t^{[1]}$ cumulative sum of weights
4: $i = 1$
5: *for* $j = 1$ *to* J *do* — J = #particles
6: $U = r + (j-1)J^{-1}$ — *step size = 1/J*
7: *while* $U > c$ — decide whether or not to take particle *i*
8: $i = i + 1$
9: $c = c + w_t^{[i]}$ — cumulative sum
10: *endwhile*
11: *add* $x_t^{[i]}$ *to* $\bar{\mathcal{X}}_t$ — particle is taken (can occur several times)
12: *endfor*
13: *return* $\bar{\mathcal{X}}_t$

6. Mapping with known Poses

General Problem:

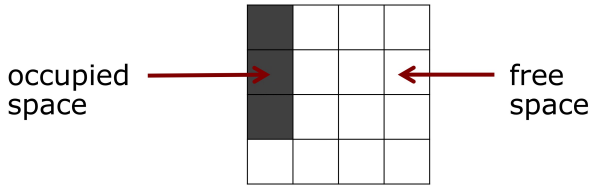
Find the most likely map given the observations and the poses of the robot:

$$d = \{x_1, z_1, x_2, z_2, \cdots, x_t, z_t\} \longrightarrow m^\star = \text{argmax}_m P(m|d)$$

Occupancy Grid Mapping

Occupancy Grid Map:

Map is represented by a grid with each cell representing if it is occupied or free.



Assumptions:

1. A cell is either completely free or occupied
2. The world is static
3. The cells are independent of each other

Occupancy Probabilities

Cell is occupied: $p(m_i) = 1$
Cell is free: $p(m_i) = 0$
No knowledge: $p(m_i) = 0.5$

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

Algorithm: occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$):

1: *for all cells* m_i *do*
2: *if* m_i *in perceptual field of* z_t *then*
3: $l_{t,i} = l_{t-1,i} + \text{inv\_sensor\_model}(m_i, x_t, z_t) - l_0$
4: *else*
5: $l_{t,i} = l_{t-1,i}$
6: *endif*
7: *endfor*
8: *return* $\{l_{t,i}\}$

Derivation: Find an efficient way to compute $p(m_i)$

1. Derive through Static State Binary Bayes Filter Formula

$$
\begin{aligned}
p(m_i \mid z_{1:t}, x_{1:t}) &\overset{\text{Bayes rule}}{=} \frac{p(z_t \mid m_i, z_{1:t-1}, x_{1:t})\, p(m_i \mid z_{1:t-1}, x_{1:t})}{p(z_t \mid z_{1:t-1}, x_{1:t})} \\
&\overset{\text{Markov}}{=} \frac{p(z_t \mid m_i, x_t)\, p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(z_t \mid z_{1:t-1}, x_{1:t})} \\
&\overset{\text{Bayes rule}}{=} \frac{p(m_i \mid z_t, x_t)\, p(z_t \mid x_t)\, p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(m_i \mid x_t)\, p(z_t \mid z_{1:t-1}, x_{1:t})} \\
&\overset{\text{indep.}}{=} \frac{p(m_i \mid z_t, x_t)\, p(z_t \mid x_t)\, p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(m_i)\, p(z_t \mid z_{1:t-1}, x_{1:t})}
\end{aligned}
$$

2. Do the same for $p(\neg m_i)$ to compute ratio

$$
p(\neg m_i \mid z_{1:t}, x_{1:t}) \overset{\text{the same}}{=} \frac{p(\neg m_i \mid z_t, x_t)\, p(z_t \mid x_t)\, p(\neg m_i \mid z_{1:t-1}, x_{1:t-1})}{p(\neg m_i)\, p(z_t \mid z_{1:t-1}, x_{1:t})}
$$

3. Calculate ratio

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

4. Convert to log odds ratio for computational efficiency

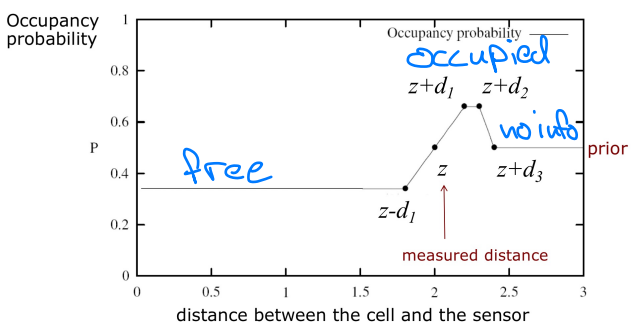
$$
\begin{aligned}
&l(m_i \mid z_{1:t}, x_{1:t}) \\
&= \underbrace{l(m_i \mid z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i \mid z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}}
\end{aligned}
$$

5. Final Mapping Formula

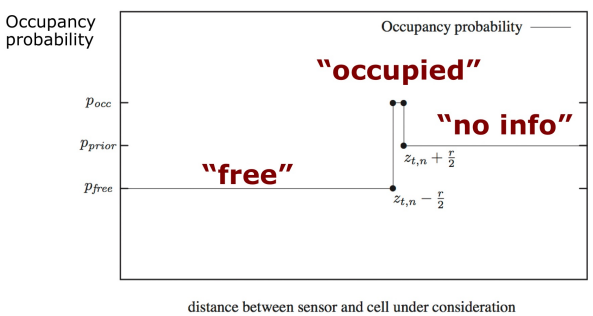
$$
l_{t,i} = \text{inv\_sensor\_model}(m_i, x_t, z_t) + l_{t-1,i} - l_0
$$

Inverse Sensor Model

Sonar:



Laser:



Reflection Probability Mapping

For each cell count hits and misses and estimate the reflection probability of given cell

Mapping:

hits(x,y): Number of cases where a beam ended in cell (x,y)
misses(x,y): Number of cases where a beam passes through cell (x,y)

For each cell compute:

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

Most Likely Map

Measurement Model: $\zeta_{t,n}=1$: Max. range reading, $\zeta_{t,n}=0$: Reflected

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

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

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

Derivation of most likely map:

$$\begin{aligned} m^\star &= \text{argmax}_m P(z_1,\cdots,z_t|m,x_1,\cdots,x_t) \\ &= \text{argmax}_m \prod_{t=1}^{T} P(z_t|m,x_t) \quad \text{since } z_t \text{ are independent and only depend on } x_t \\ &= \text{argmax}_m \sum_{t=1}^{T} \ln P(z_t|m,x_t) \\ &= \text{argmax}_m \sum_{j=1}^{J \text{ cells}} \sum_{t=1}^{T \text{ beams}} \sum_{n=1}^{N} \Big(I(f(x_t,n,z_{t,n})=j) \cdot (1-\zeta_{t,n}) \cdot \ln m_j \\ &\quad + \sum_{k=0}^{z_{t,n}-1} I(f(x_t,n,k)=j) \cdot \ln(1-m_j) \Big) \end{aligned}$$

"beam n ends in cell j"

"beam n traversed cell j"

Define

$\alpha_j = \sum_{t=1}^{T} \sum_{n=1}^{N} I(f(x_t,n,z_{t,n})=j) \cdot (1-\zeta_{t,n})$ ← number of beams ended in cell j (not max. range)

$\beta_j = \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=0}^{z_{t,n}-1} I(f(x_t,n,k)=j)$ ← number of beams passed through cell j

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

With: $\frac{\partial}{\partial m_j} = \frac{\alpha_j}{m_j} - \frac{\beta_j}{1-m_j} = 0 \qquad m_j = \frac{\alpha_j}{\alpha_j+\beta_j}$

→ Most likely map corresponds to counting how often a beam ended in each cell

7. Simultaneous Localization and Mapping

General Problem:

Estimate the robot pose while simultaneously estimate the map. Both estimations rely on each other.

Given:

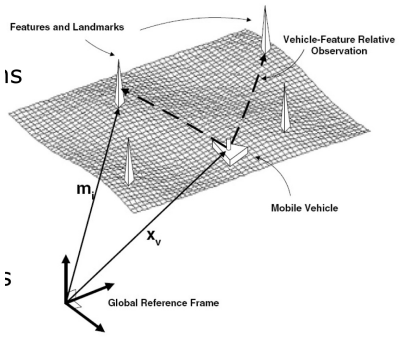
The robot's controls $u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$

Observations $z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$

Wanted:

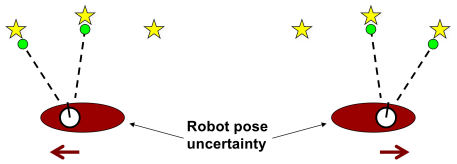
Map of the environment m

Path of the robot $x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$



Feature-based: Find landmark positions

Data Association: Map observations to the corresponding landmarks



Loop Closure

Loop closure happens when an already mapped area is recognized and the robot is able to close the path. This collapses the uncertainties. Wrong loop closure leads to divergence

Scan Matching

Incrementally match two scans or a scan against a map to locally refine the robot pose.

Maximize the likelihood:

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

- x_t^*: most likely pose at previous time step
- z_t: current measurement
- x_t: robot motion
- m_{t-1}: map constructed so far

Definition: Registration

Find a transform to match multiple scans

Given: Point clouds $X = \{x_1, \ldots, x_n\}$

$P = \{p_1, \ldots, p_n\}$

Find: A transform minimizing the mean squared error:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Closed Form Solution

A closed form solution exists, if the point correspondences are known

1. $\mu_x = \frac{1}{N_x} \sum_{i=1}^{N_x} x_i$ and $\mu_p = \frac{1}{N_p} \sum_{i=1}^{N_p} p_i$ $\longrightarrow$ $X' = \{x_i - \mu_x\} = \{x_i'\}$, $P' = \{p_i - \mu_p\} = \{p_i'\}$

2. $W = \sum_{i=1}^{N_p} x_i' p_i'^T = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T$

   $U, V \in \mathbb{R}^{3\times3}$ are unitary

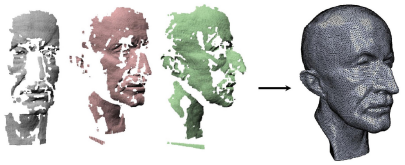
   $\sigma_1 \geq \sigma_2 \geq \sigma_3$ are the singular values of W

3. **Rotation:** $R = UV^T$

   Translation: $t = \mu_x - R\mu_p$

Iterative Closest Point Algorithm

ICP registers a point cloud obtained from a Lidar scan or RGB-D camera against a second scan. Iteratively, correspondences in the point clouds are established and aligned.



set of raw scans — reconstructed model

Algorithm: Find transform $T=(R,t)$ to transform $P \longrightarrow X$

0: WHILE (not finished)
1: Select points in the point sets
2: IF (termination criterion)
3: Return R, t
4: Establish correspondences between the points in X and P
5: Find R, t that minimizes: $E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$

x_i and p_i are corresponding points

6: Transform P

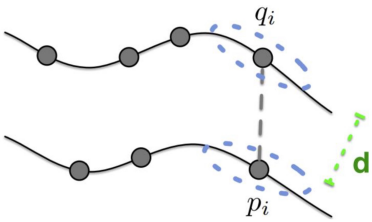
Generalized ICP:

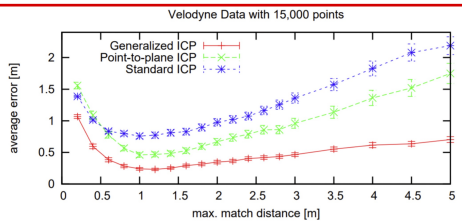
Error metrics includes uncertainty of correspondences in the alignment process

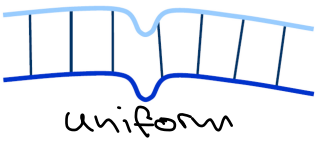
$$E_{\text{Generalized-ICP}}(\boldsymbol{T}) = \sum_{k=1}^{N} \boldsymbol{d}_k^{(\boldsymbol{T})^T} \left(\boldsymbol{\Sigma}_k^Q + \boldsymbol{T}\boldsymbol{\Sigma}_k^P \boldsymbol{T}^T\right)^{-1} \boldsymbol{d}_k^{(\boldsymbol{T})}$$

Transformation — Local covariances

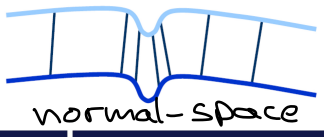
$\boldsymbol{d}_k^{(\boldsymbol{T})} = \boldsymbol{q}_k - \boldsymbol{T}\boldsymbol{p}_k$ are the point-to-point distances







uniform

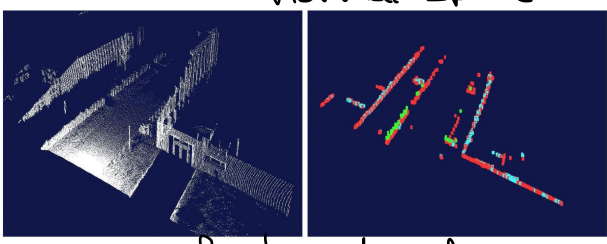


normal-space

Variations:

Step 0: Selecting source points

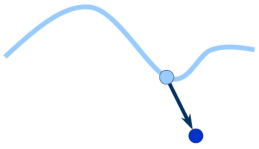
1. All points
2. Uniform sub-sampling
3. Random sampling
4. Feature-based sampling
5. Normal-space sampling
   → Normals are distributed uniformly



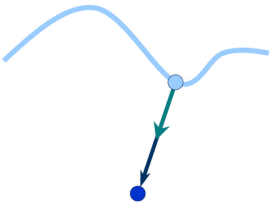
feature-based

Step 5: Weighting the correspondences
→ Weight correspondences to have more or less influence on the transform

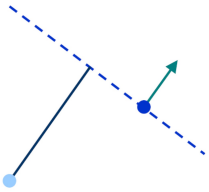
Step 4: Establishing correspondences



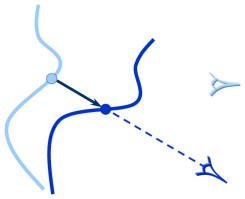
1. **Closest-Point Matching**
   - → Match to the closest point
   - → stable but slow



2. **Normal Shooting**
   - → Project along normal and select intersecting point
   - → Better but unreliable for noisy or complex data



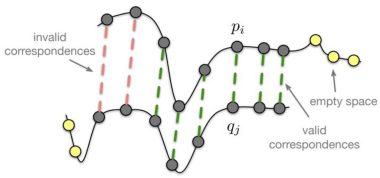
3. **Point-to-Plane Matching**
   - → Map a point to a plane spanned by the other point set
   - → needs normals and connectivity between points



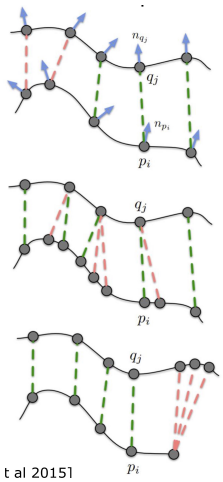
4. **Projection**
   - → project point according to the view-point
   - → worse alignments but faster data association

5. **Closest Compatible Point**
   - → Improves the previous two variations by considering the compatibility w.r.t. normals, features etc.

Step 4: Rejecting Correspondences



1. **Distance**
   - → sort correspondences by the error and trim lower percentile
2. **Normal Compatibility**
   - → compare normales, delete points with non-compatible normals
3. **Delete Duplicate Matches**
4. **Delete Boundary Points**



t al 2015]

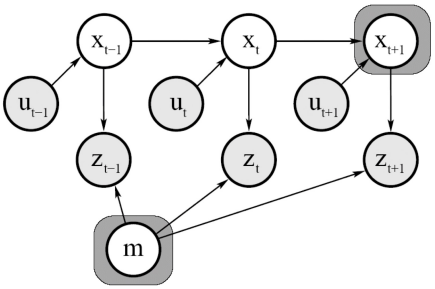
EKF SLAM

The landmarks and robot pose are included in the state and filtered simultaneously using the EKF. This allows modeling correlations between the robot pose and the landmarks.

State Representation:

$$x_t = (\underbrace{x, y, \theta}_{\text{robot's pose}}, \underbrace{m_{1,x}, m_{1,y}}_{\text{landmark 1}}, \dots, \underbrace{m_{n,x}, m_{n,y}}_{\text{landmark n}})^T$$

$$\underbrace{\begin{pmatrix} x \\ y \\ \theta \\ m_{1,x} \\ m_{1,y} \\ \vdots \\ m_{n,x} \\ m_{n,y} \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{n,x}} & \sigma_{xm_{n,y}} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{m_{n,x}} & \sigma_{m_{n,y}} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta m_{1,x}} & \sigma_{\theta m_{1,y}} & \cdots & \sigma_{\theta m_{n,x}} & \sigma_{\theta m_{n,y}} \\ \sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{\theta} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{n,x}} & \sigma_{m_{1,x}m_{n,y}} \\ \sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{\theta} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{n,x}} & \sigma_{m_{1,y}m_{n,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{n,x}x} & \sigma_{m_{n,x}y} & \sigma_{\theta} & \sigma_{m_{n,x}m_{1,x}} & \sigma_{m_{n,x}m_{1,y}} & \cdots & \sigma_{m_{n,x}m_{n,x}} & \sigma_{m_{n,x}m_{n,y}} \\ \sigma_{m_{n,y}x} & \sigma_{m_{n,y}y} & \sigma_{\theta} & \sigma_{m_{n,y}m_{1,x}} & \sigma_{m_{n,y}m_{1,y}} & \cdots & \sigma_{m_{n,y}m_{n,x}} & \sigma_{m_{n,y}m_{n,y}} \end{pmatrix}}_{\Sigma}$$



Filtering Steps:

1. **State prediction**
   → Update robot pose based on the motion model

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

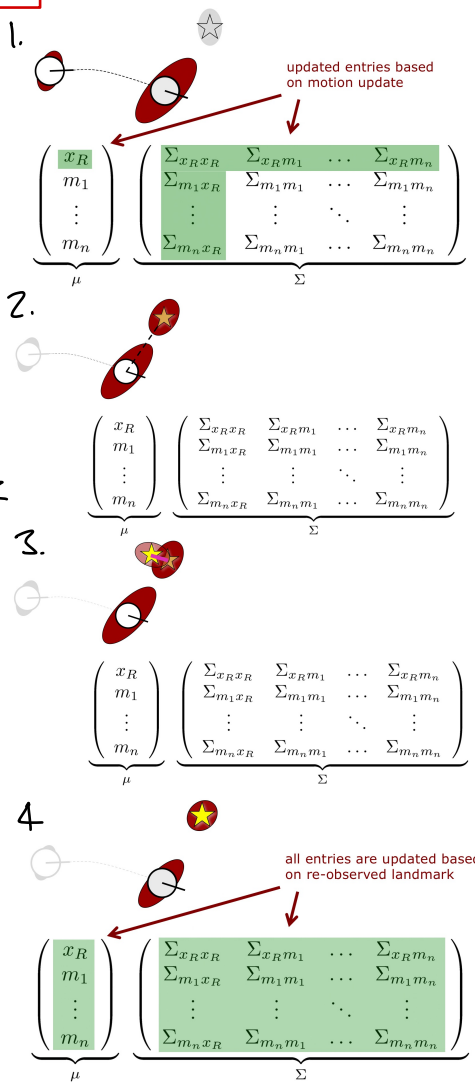
2. **Measurement prediction**
   → predict the expected measurements $\hat{z}$

3. **Data association**
   → if associations are unknown, associate each obtained measurement to a landmark using the expected measurements

4. **Update**
   → Using the data association correct all 4 states

$$K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$$
$$\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$$
$$\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$$

5. **Integration of new landmarks**
   → new landmarks are added to the state

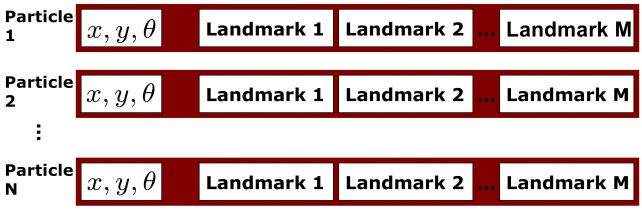


Ideas:

+ Landmarks become highly correlated
- high computational complexity for larger maps $O(n^2)$
- uni-modal

FastSLAM

Represent the belief about the map and pose using multiple particles. Each particle consists of M EKFs, one for each landmark.



Derivation: (Rao-Blackwellization)

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

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

particle filter similar to MCL

2-dimensional EKFs!

Particle Set: Pose hypothesis + M EKFs estimating the landmark positions

Filtering Steps:

1. Sample a new pose for each particle

$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$$

2. Update landmark beliefs in each particle using the EKFs
   → Solve data association for each particle

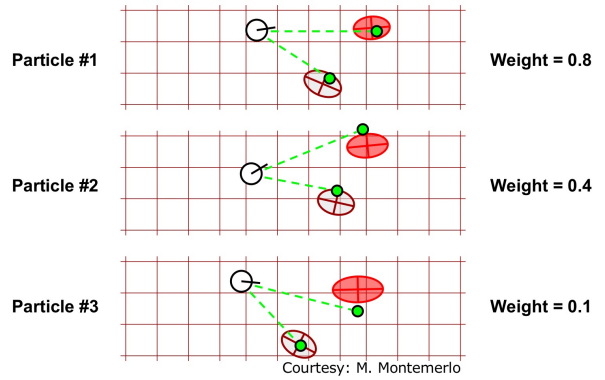
3. Compute particle weights

observation / **exp. observation**

$$w^{[k]} = |2\pi Q|^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(z_t - \hat{z}^{[k]})^T Q^{-1} (z_t - \hat{z}^{[k]})\right\}$$

measurement covariance

(includes landmark position uncertainty as well as measurement noise)



4. Resample

Ideas:

+ Data association is done for each particle, modeling ambiguities in the data association process
+ Scales well to larger maps, no high number of cross-correlation
+ More accurate than EKF, especially with noisy data

Grid-based FastSLAM

The EKFs estimating the landmarks position are replaced by a grid map. Within the particles mapping with known poses (given the particle pose) is performed

Particle Set: Pose hypothesis + Grid map

Filtering Steps:

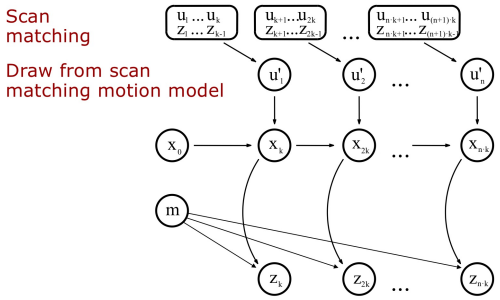
1. Perform scan matching every kth step to cope with motion noise
   → Modeling motion noise through particles is inefficient as the number of particles and maps grow large

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

current measurement

robot motion

map constructed so far



2. Sample a new pose for each particle

$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$$

3. Mapping with known poses for each particle

4. Compute particle weights
   → proportional to the likelihood of the observation model

5. Resample

FastSLAM 2.0

Adjust the proposal distribution and resampling technique to improve the estimation accuracy and reduce the number of particles to increase efficiency.

Filtering Steps:

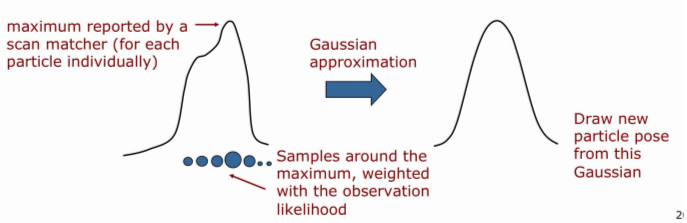
1. Perform Scan Matching
2. Sample particles with improved proposal distribution

$$x_t^{[i]} \sim p(x_t \mid x_{1:t-1}^{[i]}, u_{1:t}, z_{1:t}) \approx p(x_t|x_{t-1}^{[i]}, m^{[i]}, z_t, u_t) \simeq \mathcal{N}(\mu^{[i]}, \Sigma^{[i]})$$

$$\mu^{[i]} = \frac{1}{\eta}\sum_{j=1}^{K} x_j^{[i]}\, p(z_t|x_j^{[i]}, m^{[i]})$$

$$\Sigma^{[i]} = \frac{1}{\eta}\sum_{j=1}^{K} (x_j^{[i]} - \mu^{[i]})(x_j^{[j]} - \mu^{[i]})^T p(z_t|x_j^{[i]}, m^{[i]})$$

$x_j^{[i]}$ are the points sampled around the result of the scan matcher for particle i



3. Mapping with known poses / update belief of landmarks
4. Compute particle weights

$$\begin{aligned} w_t^{[i]} &\simeq w_{t-1}^{[i]} \int p(z_t|x_t, m^{[i]}) p(x_t|x_{t-1}^{[i]}, u_t)\, d\, x_t \\ &\simeq w_{t-1}^{[i]} c \int_{x_t \in \{x|p(z_t|x,m^{[i]})>\epsilon\}} p(z_t|x_t, m^{[i]}) dx_t \\ &\simeq w_{t-1}^{[i]} c \sum_{j=1}^{K} p(z_t|x_j^{[i]}, m^{[i]}) \end{aligned}$$

$p(z_t|x_j^{[i]}, m^{[i]})$ ← already computed during sampling

5. Selective Resampling

→ Resampling is necessary to achieve convergence
→ Might lead to important samples being deleted (Particle Depletion)

Idea: only resample if weights differ significantly

Effective Number of particles:

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

→ Measure on how well the target dis. is approximated
"Inverse variance of the normalized particle weights"

Resample, if:

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

N: Initial number of particles

Derivation of the optimal proposal distribution

$$p(x_t \mid x_{1:t-1}^{[i]}, u_{1:t}, z_{1:t})$$

Previous observations & poses are summarized in the map

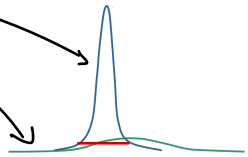
$$p(x_t \mid x_{t-1}^{[i]}, m^{[i]}, z_t, u_t) = \frac{p(z_t \mid x_t, m^{[i]})\, p(x_t \mid x_{t-1}^{[i]}, u_t)}{p(z_t \mid x_{t-1}^{[i]}, m^{[i]}, u_t)}$$

Bayes & Independance

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

Total Prob. & Independence

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



$p(z_t \mid x_t, m^{[i]})$ is typically high peaked
$p(x_t \mid x_{t-1}, u_t)$ is typically a flat distribution

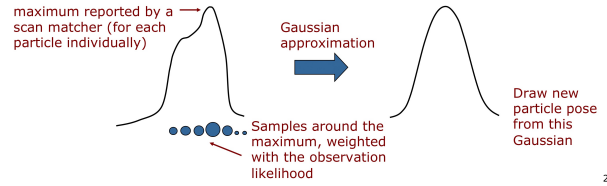
$$\Rightarrow \quad p(x_t|x_{t-1}^{[i]}, u_t)\,|_{x_t : p(z_t|x_t, m^{[i]}) > \epsilon} = c$$

$p(x_t \mid x_{t-1}, u_t)$ can be approximated by a constant

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

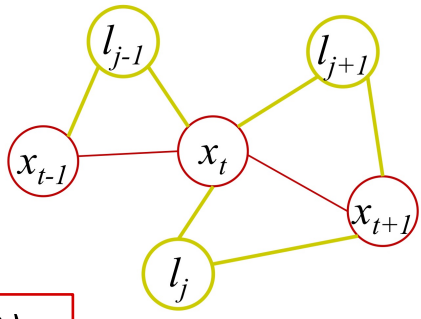
Finally: Approximate with a Gaussian and sample from Gaussian

$$p(x_t|x_{t-1}^{[i]}, m^{[i]}, z_t, u_t) \simeq \mathcal{N}(\mu^{[i]}, \Sigma^{[i]})$$



GraphSLAM

GraphSLAM models the SLAM problem as a graph. Nodes correspond to robot poses and landmark positions that are connected through edges that represent soft constraints. The goal is to maximize the probability of the robot poses and landmark positions under the given constraints.



Building up the Graph

Nodes: Robot poses X_t
Landmark positions L_i

Edges: Soft constraints

Between X_{t-1} and X_t:

$$(x_t - g(x_{t-1}, u_t))^T R_i^{-1} (x_t - g(x_{t-1}, u_t))$$

g(): motion model
R: uncertainty (covariance) of command execution

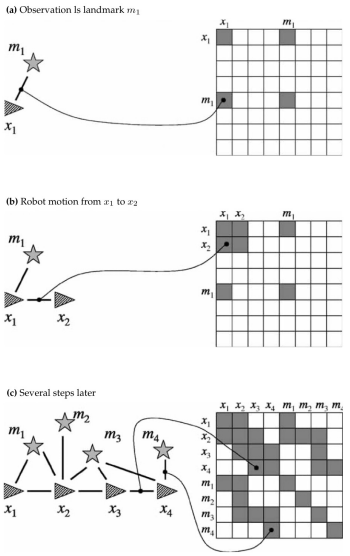
→ Uncertainty in the motion model

Between X_t and L_i:

$$(z_t(j) - h(x_t, l_j))^T Q_{tj}^{-1} (z_t(j) - h(x_t, l_j))$$

$h(x_i, l_j)$: back-projection of landmark l_j to pose x_i
Q: covariance of landmark observation

→ Uncertainty in the measurement model



Representation:

Information Matrix Ω:

$$\Omega = \sum_t \Omega_t + \sum_{ij} \Omega_{ij}$$

An entry represents a constraint (edge) between two poses or a pose and a measurement

Information Vector ξ:

$$\xi_i = \sum_t \xi_t + \sum_{ij} \xi_{ij}$$

Construction:

Linearize each constraint through th Taylor series expansion of first order and add to information matrix/ vector

For controls:

$$\Omega \longleftarrow \Omega + \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} \begin{pmatrix} -G_t & 1 \end{pmatrix}$$

$$\xi \longleftarrow \xi + \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} [g(u_t, \mu_{t-1}) - G_t \mu_{t-1}]$$

For measurements:

$$\Omega \longleftarrow \Omega + H_t^{iT} Q_t^{-1} H_t^i$$

$$\xi \longleftarrow \xi + H_t^{iT} Q_t^{-1} [z_t^i - h(\mu_t, c_t^i) + H_t^i \mu_t]$$

Optimization Problem:

Find the Maximum likelihood estimate of $\langle x_{1:n}, l_{1:m} \rangle$ by minimizing the squared error.

Maximize: $P(x_{1:n}, l_{1:m} \mid u_{1:n}, <\text{all zs}>) \propto \exp -\left(\sum_t (x_t - g(x_{t-1}, u_t))^T R_t^{-1} (x_t - g(x_{t-1}, u_t)) + \sum_{t,j} (z_t(j) - h(x_t, l_j))^T Q_{tj}^{-1} (z_t(j) - h(x_t, l_j)) \right)$

Optimization function:

$$J_{\text{GraphSLAM}} = x_0^T \, \Omega_0 \, x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T \, R_t^{-1} \, (x_t - g(u_t, x_{t-1})) + \sum_t \sum_i (z_t^i - h(y_t, c_t^i))^T \, Q_t^{-1} \, (z_t^i - h(y_t, c_t^i))$$

→ minimize negative log-likelihood

Least Squares Formulation:

$$\arg\max P(x_{1:n}, l_{1:m} \mid u_{1:n}, <\text{all zs}>) =$$

$$\arg\min \left(\sum_t (x_t - g(x_{t-1}, u_t))^T R_t^{-1} \underbrace{(x_t - g(x_{t-1}, u_t))}_{e_t} + \sum_{t,j} (z_t(j) - h(x_t, l_j))^T Q_{tj}^{-1} \underbrace{(z_t(j) - h(x_t, l_j))}_{e_{tj}} \right) =$$

$$\arg\min \left(\sum_t e_t^T \Omega_t e_t + \sum_{t,j} e_{tj}^T \Omega_{tj} e_{tj} \right)$$

e_t: Error function between the expected state x_t and the measured state $g(x_{t-1}, u_t)$

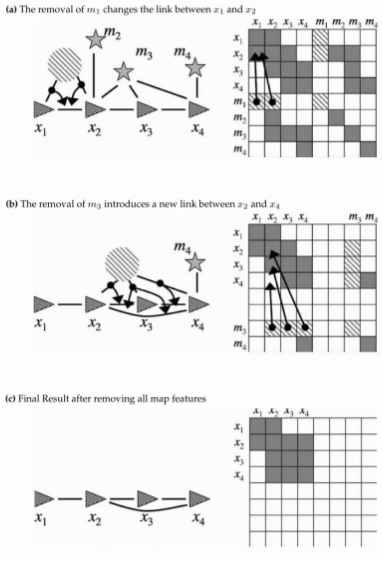
e_{tj}: Error function between the expected landmark measurement $h(x_t, l_j)$ and the real measurement $z_t(j)$

Ωs: Information matrices, i.e. inverses of Rs and Qs

Algorithm:

1. **Graph Constructions (front-end)**
   Build up nodes and edges from the controls $u_{0:t}$ and measurements $z_{1:t}$
2. **Graph Optimization (back-end)**
   The goal is to estimate poses $x_{1:n}$ and landmarks $l_{1:m}$ that maximize $P(x_{1:n}, l_{1:m} | u_{1:n}, z_{1:2})$
   Solve for: $\mu = \Omega^{-1}\xi$
   → Inversion not tractable for dense matrix with cycles

2.1 Variable Elimination



Eliminate cycles by removing iteratively all landmarks and replace them with additional constraints. The final graph consist only of the robot poses

- **Remove:** Landmark L_i and all constraints bounded to it
- **Add:** Link each node previously connected to L_i with each other and add additional constraint

Final output: Information matrix Ω'
Information vector ξ'
→ sparse matrix

2.2 Compute robot path

Solve $\mu = \Omega^{-1}\xi$ using Ω' and ξ'
→ For this step different methods might be chosen

Least Squares:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} F(\mathbf{x})$$

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_i e_i(\mathbf{x})$$

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_i \mathbf{e}_i(\mathbf{x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x})$$

$e_i(x) = x - g(x,u)$

Solution: Most probable path $x^*_{1:n}$

2.3 Recover landmark positions

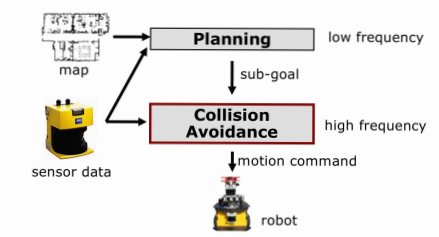
For each landmark: Build new Ω_i containing landmark and the poses retrieved from 2.3 that L_i was observed in.

Solve $\mu = \Omega^{-1}\xi$

8. Motion Planning

Motion Planning Goals

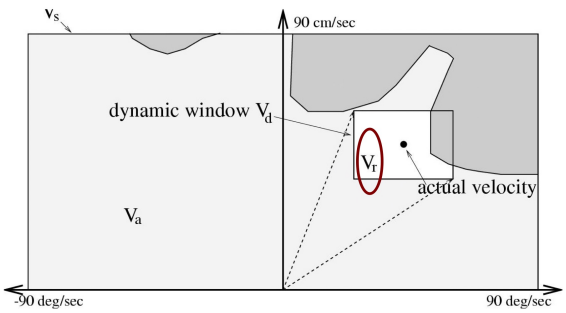
1. Reach goal location as fast as possible
2. Collision-free trajectory
3. React to unforseen objects
4. Further application-oriented objectives



Dynamic Window Approach

Generates a collision-free trajectory by using a heuristic navigation function that maximizes the velocity while keeping an admissible velocity. Each step assumes a piecewise constant velocity in a fixed time frame

Motion Commands: v: translational velocity
ω: rotational velocity



Search Space: $V_r = V_s \cap V_a \cap V_d$

V_s: Possible Velocities
V_a: Admissible Velocities
V_r: Reachable Velocities
V_d: Dynamic Window: Reachable velocities within a fixed time frames

Admissible Velocities: Velocities at which the robot is able to stop before colliding with an obstacle.

$$V_a = \{(v,\omega) \mid v \leq \sqrt{2\text{dist}(v,\omega)a_{trans}} \wedge \omega \leq \sqrt{2\text{dist}(v,\omega)a_{rot}}\}$$

dist(v,ω): Distance to the next object on the arc

Reachable Velocities: Velocities that are reachable by acceleratio within the time period t

$$V_d = \{(v,\omega) \mid v \in [v - a_{trans}t, v + a_{trans}t] \wedge \omega \in [\omega - a_{rot}t, \omega + a_{rot}t]\}$$

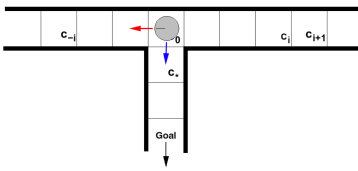
Method: 1. Discretize the search space into pairs: <v,ω>
2. Evaluate heuristic navigation function:

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$ — goal nearness

- vel: Maximize velocity
- nf: Alignment with opt. path
- Δnf: cost reduction to goal

Ideas:

+ quick and efficient collision-free trajectories
- suboptimal solutions
- no predictive planning, leading to "overshooting" when reacting to a path change



Global 5D Path Planning:

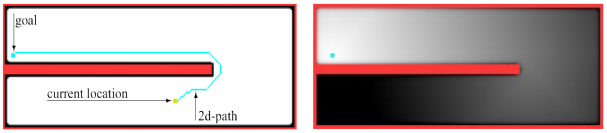
Search space: configuration space **<x,y,θ,v,ω>**
→ consider robot's kinodynamics constraints directly

Goal: Generate a sequence of steering commands

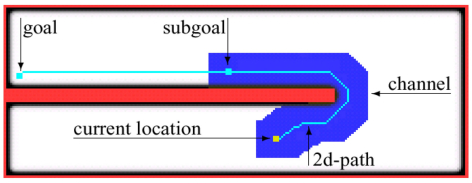
Algorithm:

1. Update the grid map based on the measurement
   → add newly detected objects, and clear free cells
   → convolve map to increase security distance

2. Use A* to find a path in the <x,y>-space
   → use a heuristic based on a deterministic value iteration in the static map



3. Determine a restricted 5D configuration space
   → Construct a channel in the 5D space around the 2D path
   → Determine a subgoal in the channel



4. Use A* to find a sequence of steering commands in the restricted configuration space
   → heuristic based on 2D value iteration to reach the subgoal

Map Convolution

Blur the map to increase size of obstacles to create more stable trajectories by keeping a secure distance to the obstacles

Gaussian Blur:

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

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

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

→ A threshold for a cell being free must be chosen

A*- Algorithm

An algorithm which uses a heuristic function to iteratively expand the state to find the shortest path. The path is optimal if the heuristic is admissable

Cost function: $f(n) = g(n) + h(n)$

$g(n)$: Actual accumulated cost from initial state to state n

$h(n)$: Estimated cost from state n to goal

Admissable heuristic: h is admissable, if $h(n) \leq h^*(n)$

Assumptions:

1. Robot pose is known
2. Path is computed on basis of an accurate map
3. Motion commands are executed correctly

Algorithm

```
0: WHILE Goal not found
1:     Etract state with minimal cost function
2:     IF, goal is reached:
3:         Return path with minimal cost
4:     Else:
5:         Expand the state with the minimum f-value
```

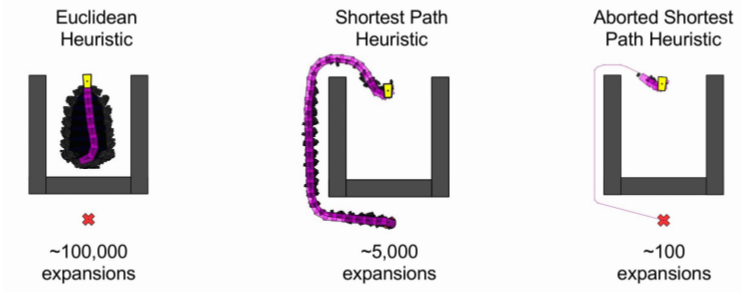
Deterministic Value Iteration

Compute the cost of the shortest path from each state to the goal (similar to Dijkstra). Use as heuristic for the A* algorithm.

Problem heuristics:

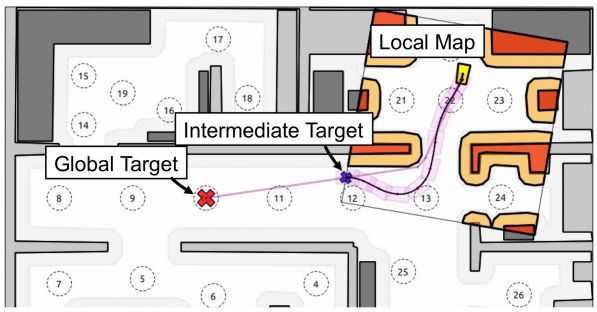
→ Some heuristics lead the A* search getting stuck in local minima

→ Heuristics might lead to a large number of expansions



Aborting A*

Algorithm that calculates the shortest collision-free path in a local map. The algorithm is real-time capable with ~30 Hz. In each step, A* finds the shortest path in the visibility graph by minimizing the execution time of the motion commands. After a time limit the search is aborted and returns a local path.



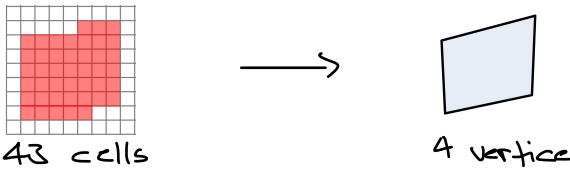
Geometric Map

Geometric Representation:

Model the map through polygons that indicate occupied spaces

Advantages:

- Smaller memory consumption
- potentially more accurate
- potentially faster processing

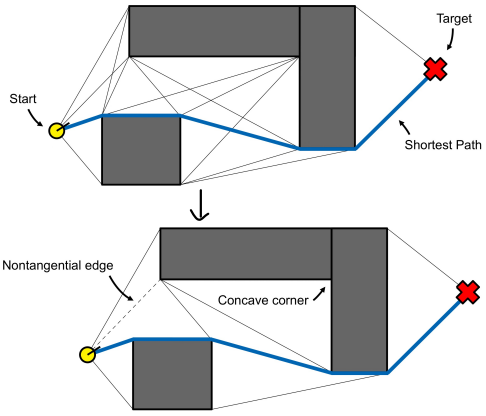


Visibility Graph:

Used as a heuristic in a geometric scene to replace the Dijkstra heuristic in a grid

Nodes: Corners

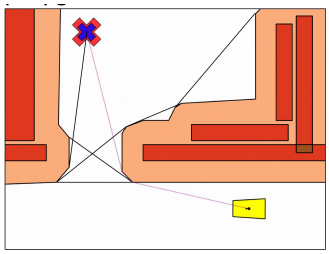
Edges: Between corners that are not separated by an obstacle



Reduced Visibility Graph:

Remove edges to concave corners and keep only edges that are tangential in both ends

Convolution: Expand Polygons



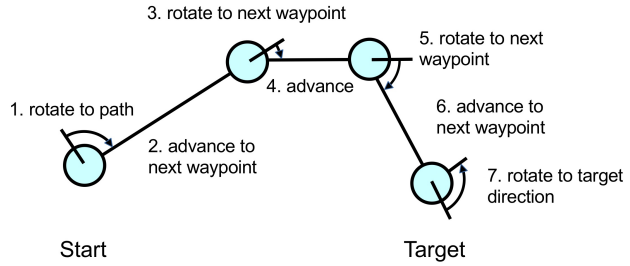
Cost function: Time required for motion

State expansion: $\langle x_1, y_1, \theta, v, \omega \rangle \rightarrow \langle x_2, y_2, \theta+t\omega, v+ta, \omega+tb \rangle$

→ x_2, y_2 are computed through an arc approximation

Path PTR Heuristic: Lay out Rotate-Translate-Rotate motions along the shortest path

→ Heuristic to estimate the time needed for the motion commands

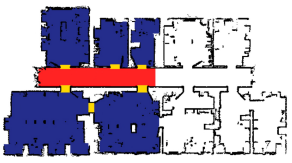


Algorithm:

```
1 push(Q, currentState)
2 while(Q is not empty and target not found)
3    bestSoFar = Q.pop()
4    if (stopWatch.elapsedTime() >= 30 ms) /* abort condition */
5       return bestSoFar
6    foreach ((t,a,b) in actionSet)
7       child.state = stateTransition(t,a,b)
8       child.g = bestSoFar.g + t
9       child.h = PathRTR(child, target)
10      child.f = child.g + child.h
11      if (!collides(child))
12         Q.push(child)
```

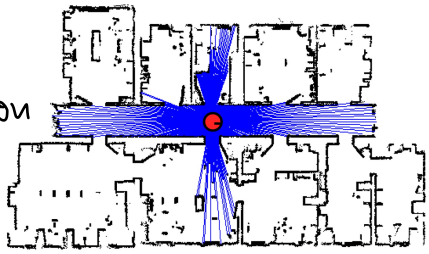
9. Perception

Place Recognition

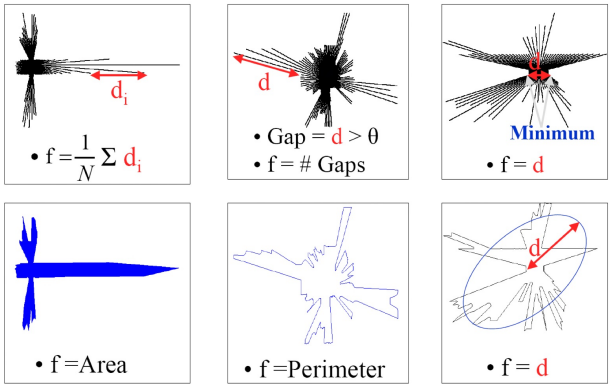


Goal: Recognize a place a robot is located in by a single observation

Representation: Rather than being represented by the single beams the observation is represented by features.



Features:



$f_{coffee\text{-}machine} = 2$

Visual Features: $f_{o \in O} : V \rightarrow N$

Number of objects visible in the image

Classification: Combine multiple weak features/classifier to a strong classifier using multiple features

Weak Classifier:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ -1 & \text{otherwise.} \end{cases}$$

Θ_j: Threshold

p_j: Direction of inequality

→ Parameters are determined by AdaBoost

AdaBoost: Combine multiple weak classifiers to a strong one. During training the examples are weighted by giving wrongly classified instances a bigger weight to boost the performance of the ensamble

Algorithm:

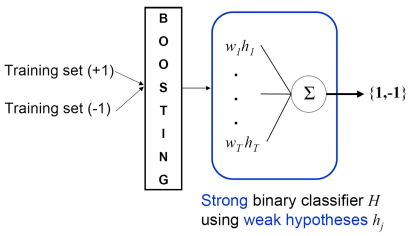
Input: set of examples $(x_1, y_1), \ldots, (x_N, y_N)$ $\forall y_i \in \{+1, -1\}$

Initialize weights $D_1(i) = \frac{1}{m}$ // all examples get same weight

For $t = 1, \ldots, T$:

1. Select h_t with less error
2. Choose weight α_t for h_t // minimize loss of classifier extended by h_t
3. Update weights: // decrease weight of correct examples // increase weight of wrong examples

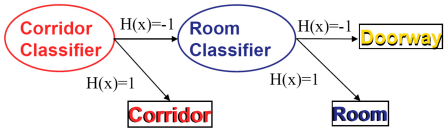
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$



Classifier:

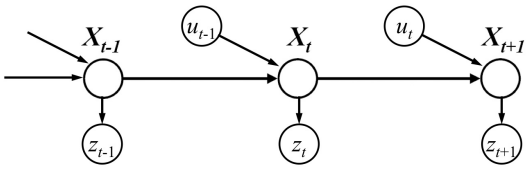
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

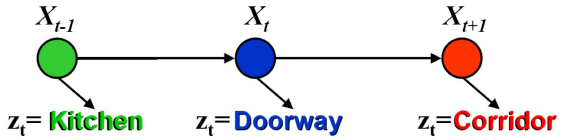
Multi-class classifier:



Probabilistic Place Recognition

Model the transition between the places through a Hidden Markov Model with a transition matrix





$$Bel(X_t) = \alpha P(z_t|X_t) \sum_{X_{t-1}} P(X_t|X_{t-1}, u_{t-1}) Bel(X_{t-1})$$

Normalization factor

Observation model

Transition model

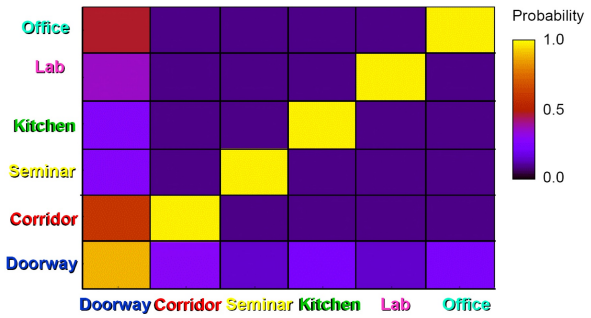
$$Bel(X_t) = \alpha P(z_t|X_t) \sum_{X_{t-1}} P(X_t|X_{t-1}, u_{t-1}) Bel(X_{t-1})$$

Normalization factor

Observation model: **AdaBoost classification rate**

Transition model: **Transition Matrix**

Transition Matrix:



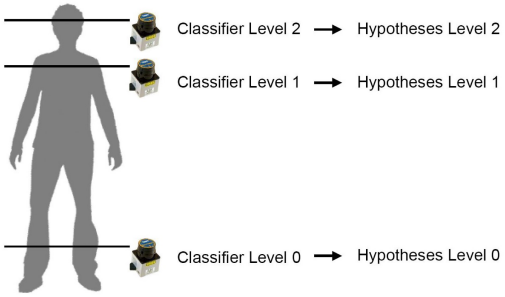
Person Perception

Detection from Laser scans. Classify segments as persons.

Classification Problem:

1. Select the **features** representing a segment
2. Select a **classifier** for the learning process

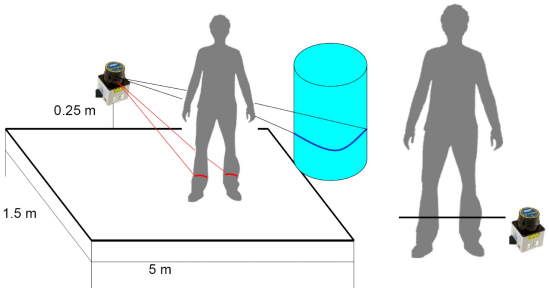
→ Classify using AdaBoost and combine different features

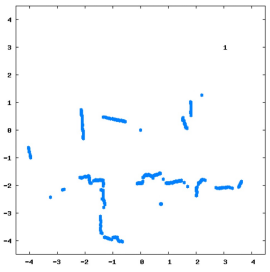


Features: Leg, body and head express different features

Level 0: Legs

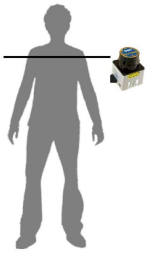
Level 0 (legs):
1. Mean angular difference
2. Standard Deviation
3. Mean Curvature
4. Standard Deviation
5. Boundary regularity
6. Mean average deviation from median
7. Boundary regularity
8. Mean Curvature
9. Radius
10. Standard Deviation

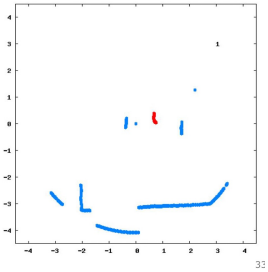




Level 1: Body

Level 1 (torso):
1. Mean angular difference
2. Linearity
3. Mean angular difference
4. Circularity
5. Linearity
6. Mean curvature
7. Number of points
8. Boundary length
9. Linearity
10. Number of points



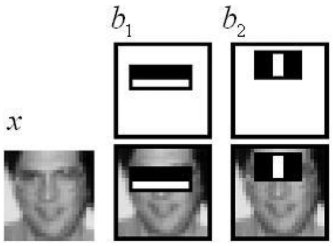


Level 2: Head

→ use face detection to detect the head

Face Detection

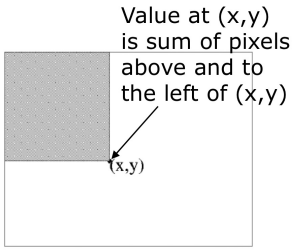
Idea: Detect relative darkness between different regions



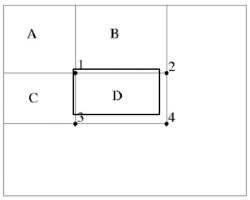
Features:

Difference of the sum of pixel values in rectangular regions
→ High number of possible features

Integral image: An image, where pixel (x,y) describes the sum of the pixels above and to the left from the original image



Integral image

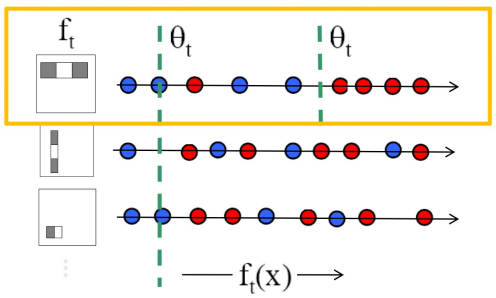


$D = 1+4-(2+3)$
$= A+(A+B+C+D)-(A+C+A+B)$
$= D$

→ Sum of pixels within a rectangular region can easily be computed from the integral image using the corner points

Classification: AdaBoost

Select single features that best separates the data

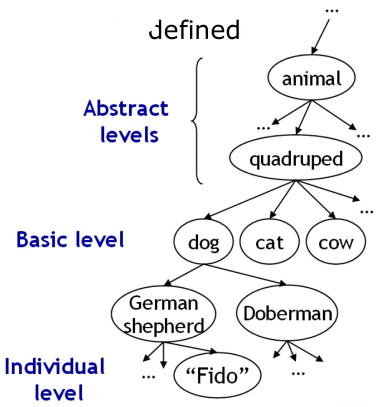


Outputs of a possible rectangle feature on faces and non-faces

Object Recognition

Levels:

1. Object classification in image
2. Object detection (position) in image
3. Object segmentation



Categories:

Categorize can be differently defined in different granularities.

Challenges

1. Variations:

   Illumination, pose, clutter, occlusion, POV

2. Robustness:

   Detection in crowded scenes

3. Scale, Efficiency

   High number of training data, efficient inference, scalability to large amounts of classes

4. Supervision

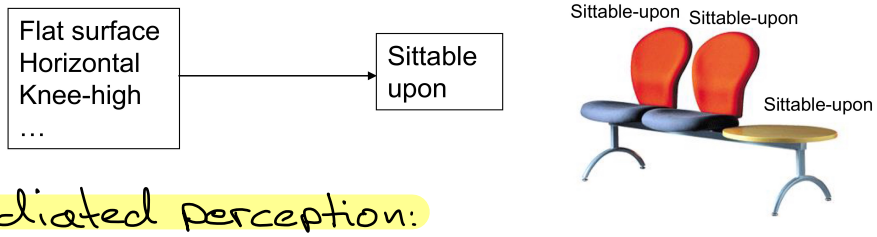
   Number of data and labels required to learn the task

101

Perception of Function

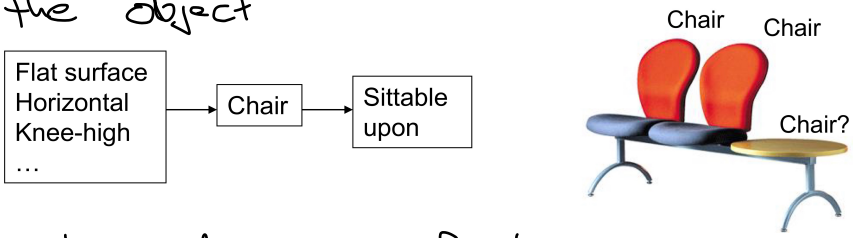
Direct perception:

Conclude function directly from the observed features



Mediated perception:

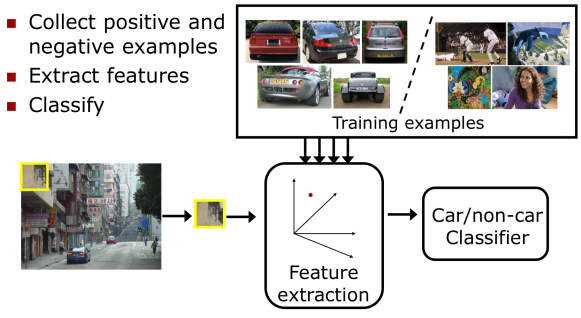
Detect an object and conclude the function from the object



Detection by Classification

Slide a window over the image and classify the window to detect an object

→ high computational complexity
→ context lost bw. windows
→ rectangular assumption often violated



Features:

Gradient-based Representations

→ consider edges, contours and oriented intensity gradients

Chamfer Matching:

Match a template against the edge image in a window

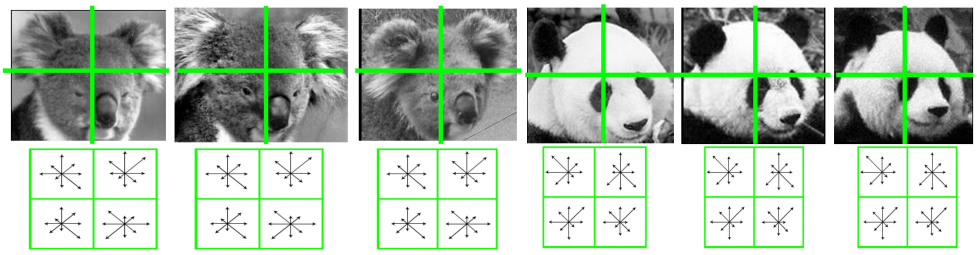
Select position with minimal distance: $D_{chamfer}(T, I) \equiv \frac{1}{|T|} \sum_{t \in T} d_I(t)$

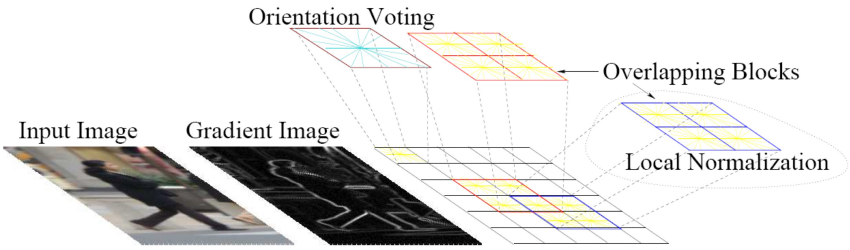
Input image | Edges detected | Distance transform | Template shape | Best match

Histograms of Oriented Gradients (HoG)

Locally summarize the distribution of gradients in a histogram

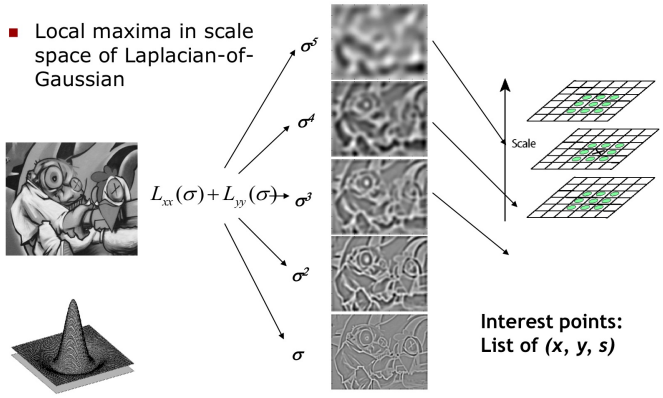
→ invariance to small shifts and rotations

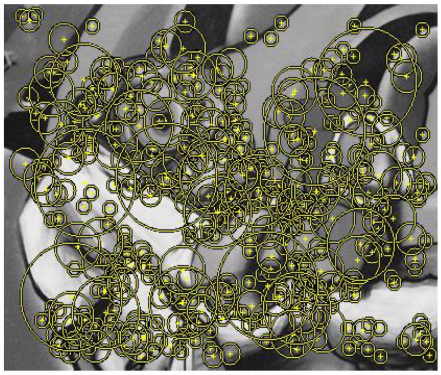




Laplacian-of-Gaussian (LoG)

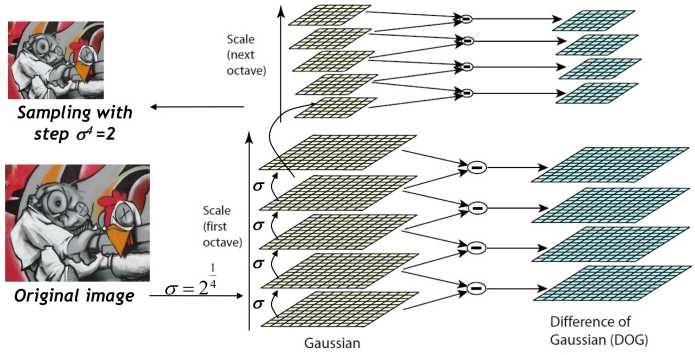
Represent the image at different scales and find interest points through local maxima





Difference-of-Gaussian (DoG)

Blur the images at different scales and calculate the difference to build a gaussian pyramid in order to approximate the LoG



Scale-invariant feature transform

1. Detect interest points using DoG

   Key points are extracted at different scales
   → invariance against scale and location

2. Determine orientation of the key point

   Describe local image patch through a histogram of orientations and extract the most dominant one
   → invariance against rotation

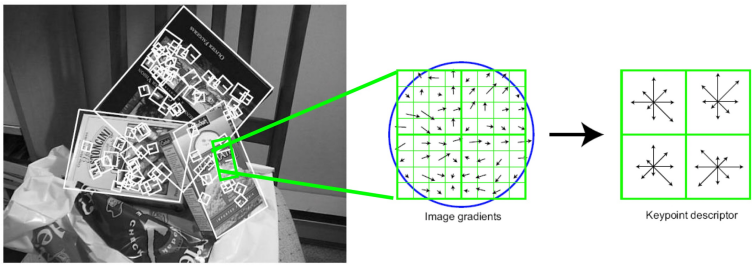
3. Generate a key-point descriptor

   3.1 Determine gradients in a 16x16 neighborhood

   3.2 Split into 4x4 subregions and create histograms of orientation using 8 bins for each region

   3.3 Key-point descriptor is a vector of 128 values
   → normalize for invariance against illumination

   → invariance against illumination and viewpoint

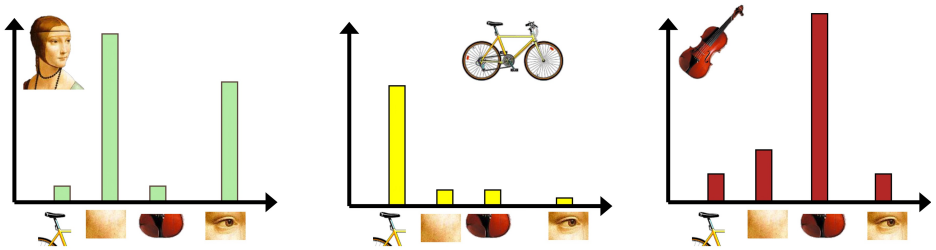


Bag-of-Words Model

Describe/identify an object through its components

Object → **Bag of 'words'**

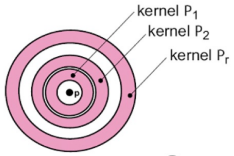
Represent an object through a histogram of visual words



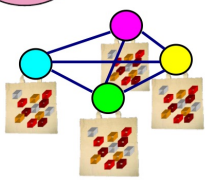
→ identification through SIFT descriptors

Adding spatial information:

Feature level
- Circular kernels (Savarese, Winn and Criminisi, CVPR 2006)



Generative models
- Model dependency of adjacent image regions (Niebles & Fei-Fei, CVPR 2007)

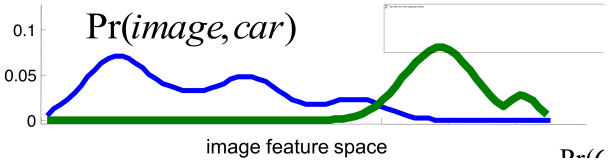


Discriminative methods
- Multiscale histogram (Lazebnik, Schmid & Ponce, 2006)

Classifier Construction

Generative Model: Model the underlying priors and joint distribution of the data. That means the learner learns the features of the data

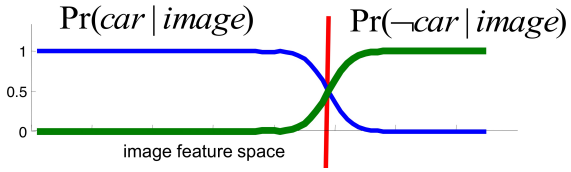
Classification: $\Pr(Class \mid image) = \frac{\Pr(image \mid Class)\Pr(Class)}{\Pr(image)}$



Ideas:

+ possibly interpretable
+ can draw samples
- models variability unimportant to classification task
- often hard to build good model with few parameters

Discriminative Model: Directly learn the posterior distribution. Thus, it learns a decision boundary to split the data.



Ideas:

+ appealing when infeasible to model data itself
+ excel in practice
- often can't provide uncertainty in predictions
- non-interpretable

Detector Performance

Confusion Matrix:

	Expected Result (ground truth)	
Computed Result (detection + classification)	true positive **Correct** result	false positive **Unexpected** result
	false negative **Missing** result	true negative **Correct absence** of result

Precision: **tp / (tp+fp)**

Recall: **tp / (tp+fn)**

F1 Score: 2 (Recall·Precision) / (Recall + Precision)