ETH Zürich
Chair of Software Engineering

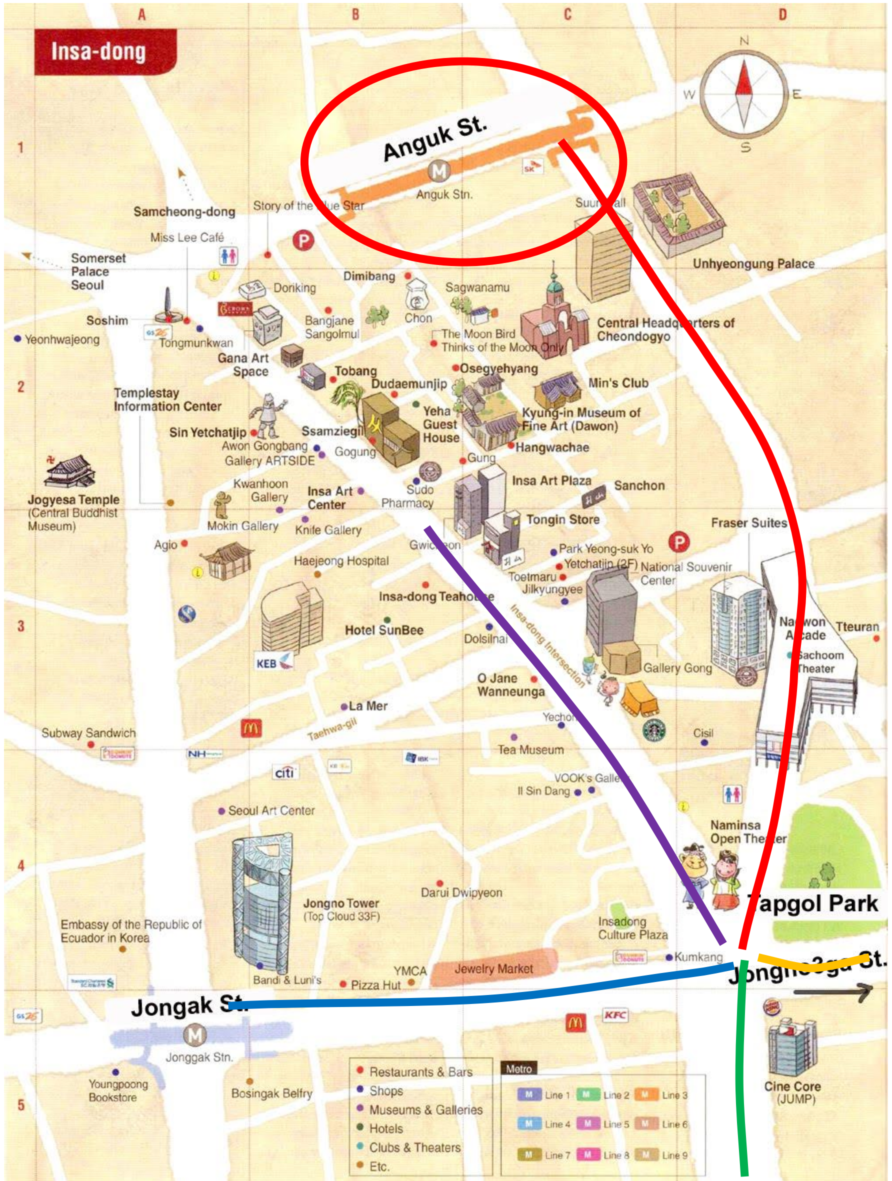
Robotics Programming Laboratory

Bertrand Meyer
Jiwon Shin

Lecture 10: Localization and mapping

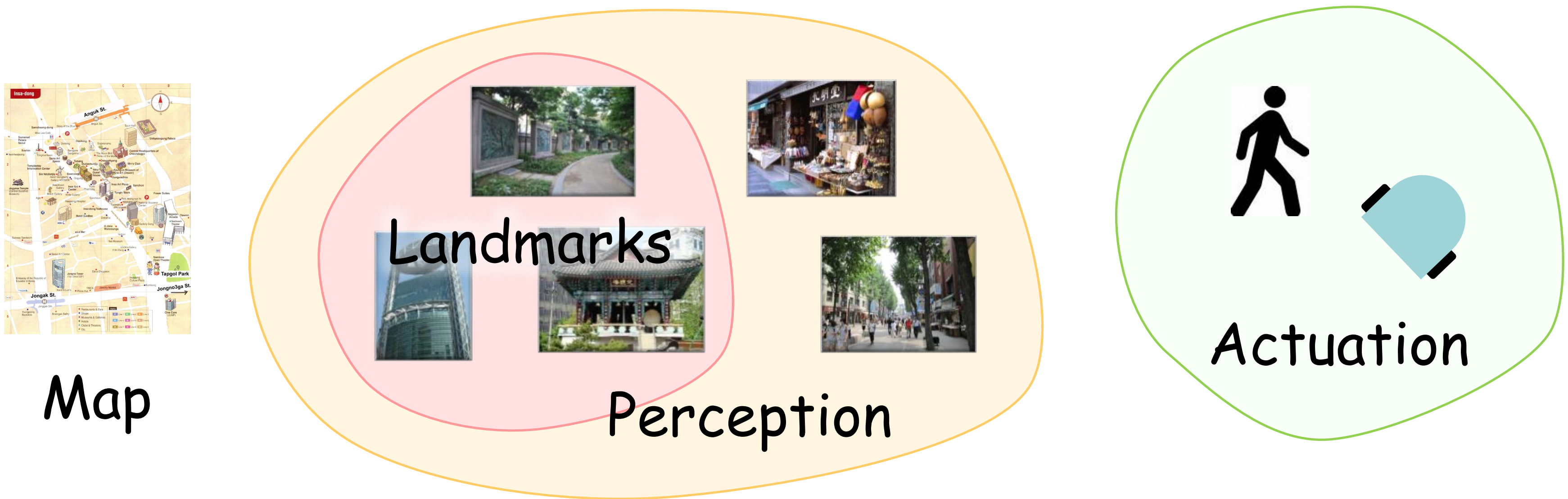
This lecture is based on “Probabilistic Robotics” by Thrun, Burgard, and Fox (2005).

Where am I?


Insa-dong
A
B
C
D
1
2
3
4
5
N
E
S
W
Anguk St.
Anguk Stn.
Samcheong-dong
Miss Lee Café
Somerset Palace Seoul
Dimibang
Donking
Sagwanamu
Unhyeongung Palace
Soshim
Yeonhwajeong
Tongmunkwan
Bangjane Sangolmul
Chon
The Moon Bird Thinks of the Moon Only
Central Headquarters of Cheondogyo
Gana Art Space
Tobang
Dudaemunjip
Osegyehyang
Min's Club
Templestay Information Center
Yeha Guest House
Kyung-in Museum of Fine Art (Dawon)
Sin Yetchatjip
Ssamziegil
Awon Gongbang
Gallery ARTSIDE
Gogung
Hangwachae
Gung
Jogyesa Temple (Central Buddhist Museum)
Kwanhoon Gallery
Insa Art Center
Sudo Pharmacy
Insa Art Plaza
Sanchon
Mokin Gallery
Knife Gallery
Tongin Store
Fraser Suites
Agio
Haejeong Hospital
Park Yeong-suk Yo
National Souvenir Center
Toetmaru
Jilkyungyee
Insa-dong Teahouse
Tteuran
Hotel SunBee
Dolsilnai
Insa-dong Intersection
Gallery Gong
KEB
O Jane Wanneunga
La Mer
Taehwa-gil
Subway Sandwich
Cisil
Tea Museum
citi
Il Sin Dang
Seoul Art Center
Naminsa Open Theater
Tapgol Park
Jongno Tower (Top Cloud 33F)
Darui Dwipyeon
Embassy of the Republic of Ecuador in Korea
Insadong Culture Plaza
Kumkang
Jewelry Market
YMCA
Bandi & Luni's
Pizza Hut
Jongak St.
Jonggak Stn.
KFC
Cine Core (JUMP)
Youngpoong Bookstore
Bosingak Belfry
Restaurants & Bars
Shops
Museums & Galleries
Hotels
Clubs & Theaters
Etc.
Metro
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9

Localization

Localization: process of locating an object in space



Dimensions of localization

Type of localization

- Local localization: initial pose is known.
- Global localization: initial pose is unknown.
- Kidnapped robot problem: the robot gets teleported to some location during the operation.

Environments

- Static: the robot is the only moving object.
- Dynamic: other objects change their configuration or location over time.

Approaches

- Passive: the localization module only observes the robot.
- Active: the localization module actively controls the robot to minimize the error and/or the cost of bad localization.

Number of robots

- Single-robot: all data are collected at a single robot platform.
- Multi-robot: communication between the robots can enhance their localization.

Probabilistic robotics

Uncertainty!

- Environment, sensor, actuation, model, algorithm
- Represent uncertainty using the calculus of probability theory

Probability theory

- X: random variable
  - Can take on discrete or continuous values
- P(X = x), P(x) : probability of the random variable X taking on a value x
- Properties of P(x)
  - $P(X = x) >= 0$
  - $\sum_x P(X = x) = 1$ or $\int_x p(X = x) = 1$

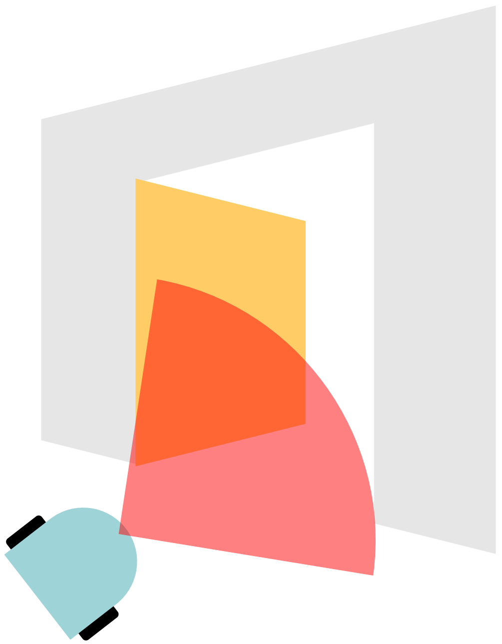
Probability

- $P(x,y)$: joint probability
  - $P(x,y) = P(x)\, P(y)$: X and Y are independent

- $P(x \mid y)$: conditional probability of x given y
  - $P(x \mid y) = p(x)$: X and Y are independent
  - $P(x,y \mid z) = P(x \mid z)\, P(y \mid z)$: conditional independence
  - $P(x \mid y) = P(x,y) / P(y)$
  - $P(x,y) = P(x \mid y)\, P(y) = P(y \mid x)\, P(x)$

- $P(x \mid y) = \dfrac{P(y \mid x)\, P(x)}{P(y)} = \dfrac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$: Bayes' rule
  - $P(y) = \sum_X P(x,y) = \sum_X P(y \mid x)\, P(x)$: Law of total probability

Bayes' rule



$$P(door=open \mid sensor=far)$$

$$= \frac{P(far \mid open)\, P(open)}{P(far)}$$

$$= \frac{P(far \mid open)\, P(open)}{P(far \mid open)\, P(open) + P(far \mid closed)\, P(closed)}$$

Bayes' filter

$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$: belief on the robot's state x_t at time t

Compute robot's state: $bel(x_t)$

- Predict where the robot should be based on the control $u_{1:t}$

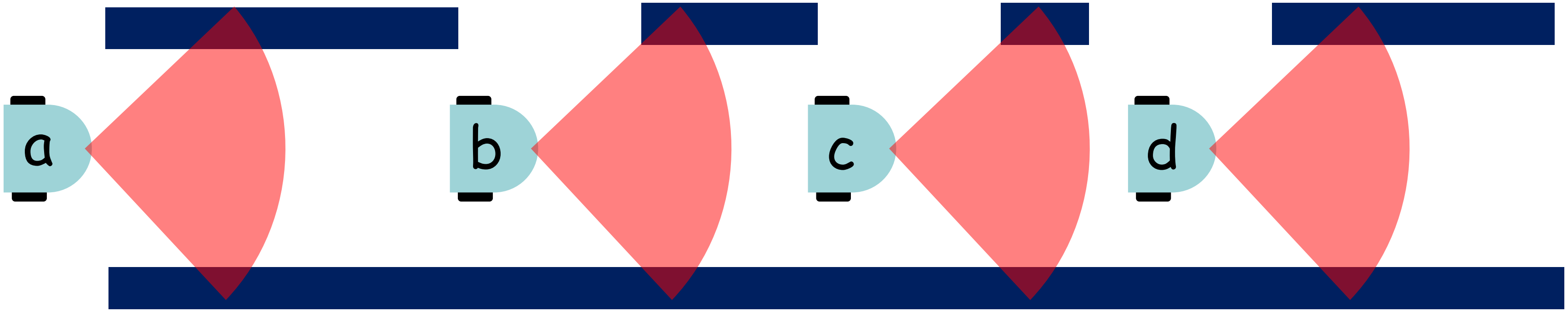
$$bel^*(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx_{t-1}$$

- Update the robot state using the measurement $z_{1:t}$

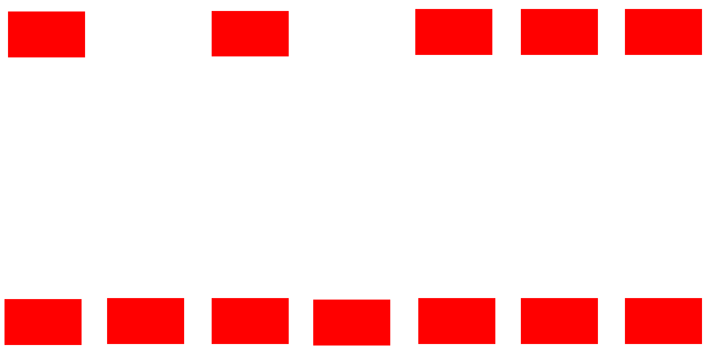
$$bel(x_t) = \eta\, p(z_t \mid x_{t-1})\, bel^*(x_t)$$

Markov localization

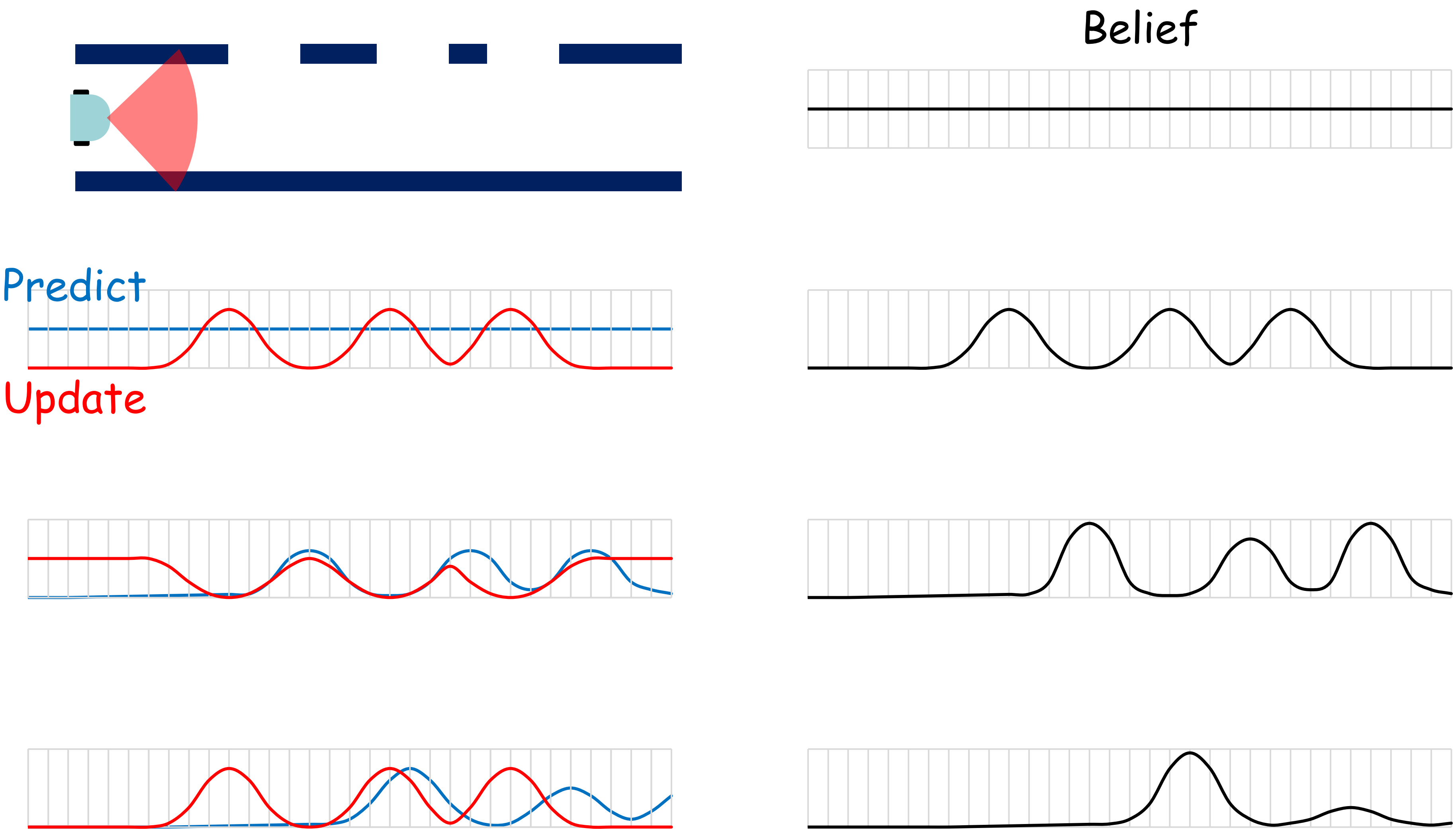
World



Measurement



Markov localization



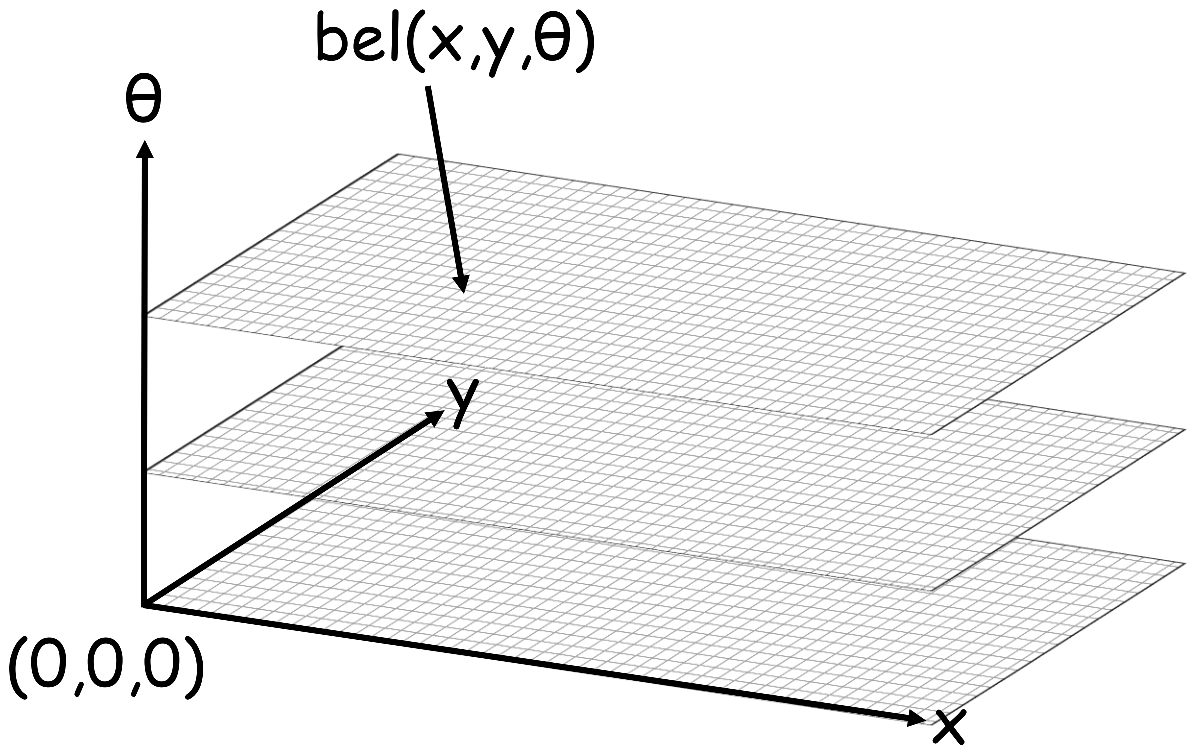
Markov localization

```
Markov_localize ( bel_{t-1}: ARRAY[ROBOT_POSE];
                  u_t: ROBOT_CONTROL;
                  z_t: SENSOR_MEASUREMENT;
                  m: MAP) : ARRAY[ROBOT_POSE]
    do
        from  i := bel_t.lower  until  i > bel_t.upper loop
                x_t := bel_t[i]
Predict         bel*_t[i] := ∫p(x_t | u_t, x_{t-1}, m) bel_{t-1}(x_{t-1}) dx_{t-1}
Update          bel_t[i] := η p(z_t | x_{t-1}, m) bel*_t[i]
                i := i + 1
        end
        Result := bel_t
    end
```

Predict: $bel^*_t[i] := \int p(x_t \mid u_t, x_{t-1}, m)\, bel_{t-1}(x_{t-1})\, dx_{t-1}$

Update: $bel_t[i] := \eta\, p(z_t \mid x_{t-1}, m)\, bel^*_t[i]$

Representation of the robot states

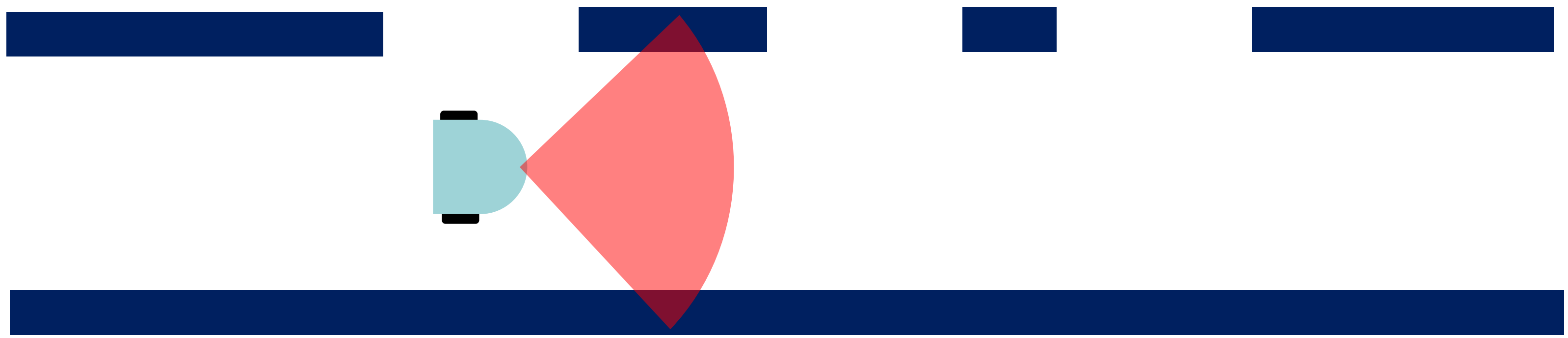


Markov localization

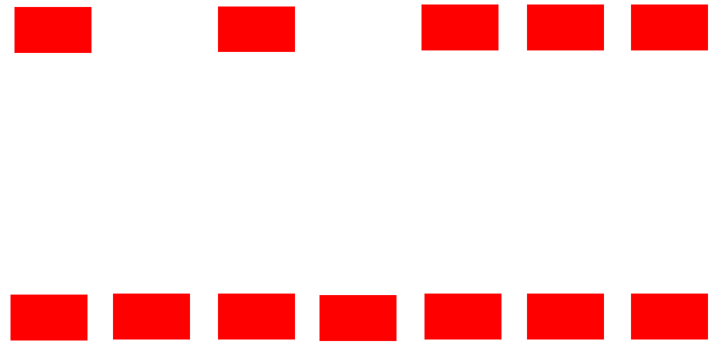
- Can be used for both local localization and global localization
  - If the initial pose (x^*_0) is known: point-mass distribution
    - $bel(x_0) = \begin{cases} 1 & \text{if } x_0 = x*_0 \\ 0 & \text{otherwise} \end{cases}$
  - If the initial pose (x^*_0) is known with uncertainty Σ:
    Gaussian distribution with mean at x^*_0 and variance Σ
    - $bel(x_0) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - x*_0)^T \Sigma^{-1} (x_0 - x*0)\right\}$
  - If the initial pose is unknown: uniform distribution
    - $bel(x_0) = \frac{1}{|X|}$
- Computationally expensive
  - Higher accuracy requires higher grid resolution

What if we know the initial pose?

Estimate the robot pose with a Gaussian distribution!



Measurement



Properties of Gaussian distribution

Univariate

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

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

Multivariate

$$\left.\begin{array}{l} X \sim N(\mu, \Sigma) \\ Y = AX + B \end{array}\right\} \Rightarrow Y \sim N(A\mu + B, A\Sigma A^T)$$

$$\left.\begin{array}{l} X_1 \sim N(\mu_1, \Sigma_1) \\ X_2 \sim N(\mu_2, \Sigma_2) \end{array}\right\} \Rightarrow p(X_1)\cdot p(X_2) \sim N\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2}\mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2}\mu_2, \quad \frac{1}{\Sigma_1^{-1} + \Sigma_2^{-1}}\right)$$

Kalman filter localization

A special case of Markov localization

- The system is linear (describable as a system of linear equations)
- The noise in the system has a Gaussian distribution

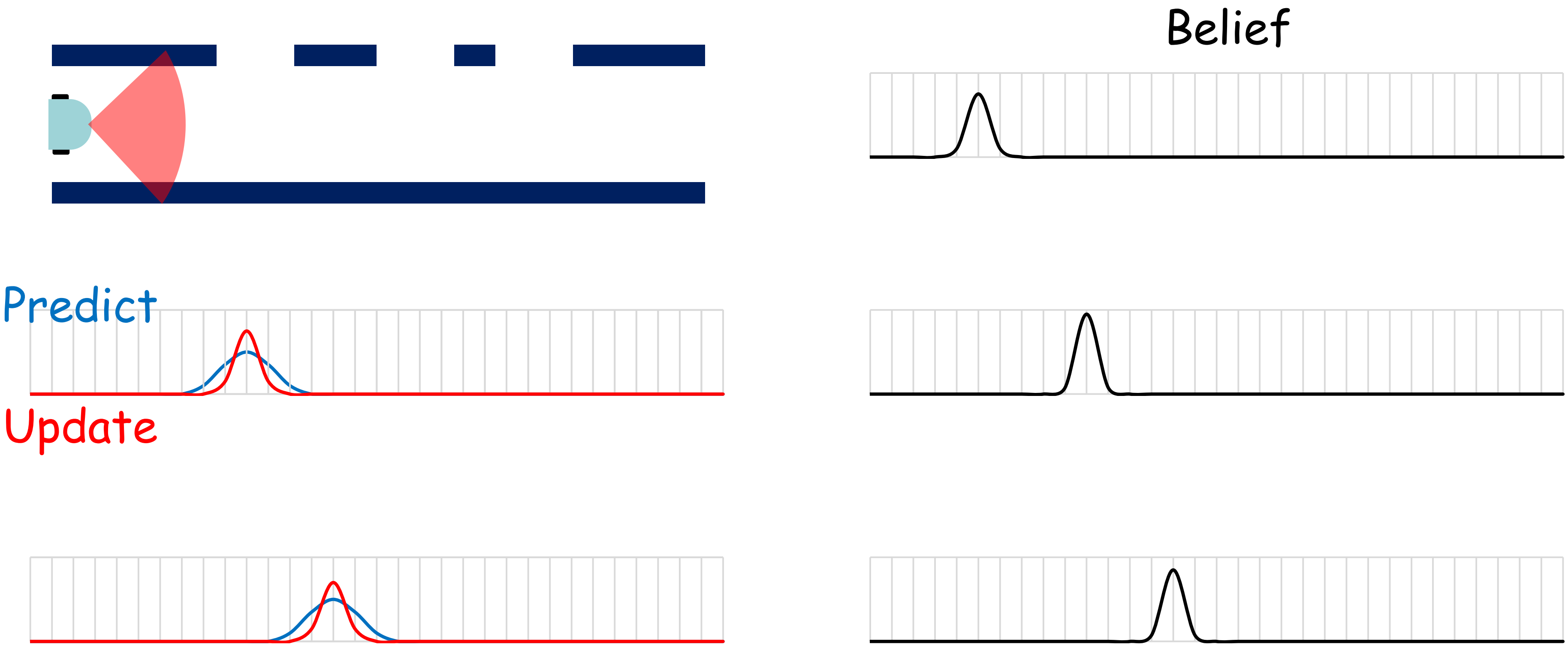
Linear transition model

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

Linear observation model

$$z_t = C_t\, x_t + \delta_t$$

Kalman filter localization



Kalman filter

```
Kalman_filter ( x_{t-1}: ROBOT_POSE;
                u_t: ROBOT_CONTROL;
                z_t: SENSOR_MEASUREMENT ) : ROBOT_POSE
    do
```

$\mu_{t-1} := x_{t-1}.mean$

$\Sigma_{t-1} := x_{t-1}.covariance$

Predict

$\mu^*_t := A_t\, \mu_{t-1} + B_t\, u_t$

$\Sigma^*_t := A_t\, \Sigma_{t-1}\, A_t^T + R_t$

$K_t := \Sigma^*_t\, C_t^T (C_t\, \Sigma^*_t\, C_t^T + Q_t)^{-1}$

Update

$\mu_t := \mu^*_t + K_t\, (z_t - C_t\, \mu^*_t)$

$\Sigma_t := (I - K_t\, C_t)\, \Sigma^*_t$

Result := **create** {ROBOT_POSE}.make_with_variables(μ_t, Σ_t)

end

Kalman filter: prediction

$\mu^*_t = A_t \, \mu_{t-1} + B_t \, u_t$

- system state estimation for time t

$\Sigma^*_t = A_t \, \Sigma_{t-1} \, A_t^T + R_t$

- estimation the system uncertainty

A_t: process matrix that describes how the state evolves from t to t-1 without controls or noise

B_t: matrix that describes how the control u_t changes the state from t to t-1

R_t : Process noise covariance

Kalman filter: update

$K_t = \Sigma^*_t C_t^T (C_t \Sigma^*_t C_t^T + Q_t)^{-1}$

- Kalman gain: how much to trust the measurement
- The lower the measurement error relative to the process error, the higher the Kalman gain will be

$\mu_t = \mu^*_t + K_t (z_t - C_t \mu^*_t)$

- update μ_t with measurement

$\Sigma_t = (I - K_t C_t) \Sigma^*_t$

- estimate uncertainty of μ_t

C_t: measurement matrix relating the state variable and measurement

Q_t: measurement noise covariance

Extended Kalman filter

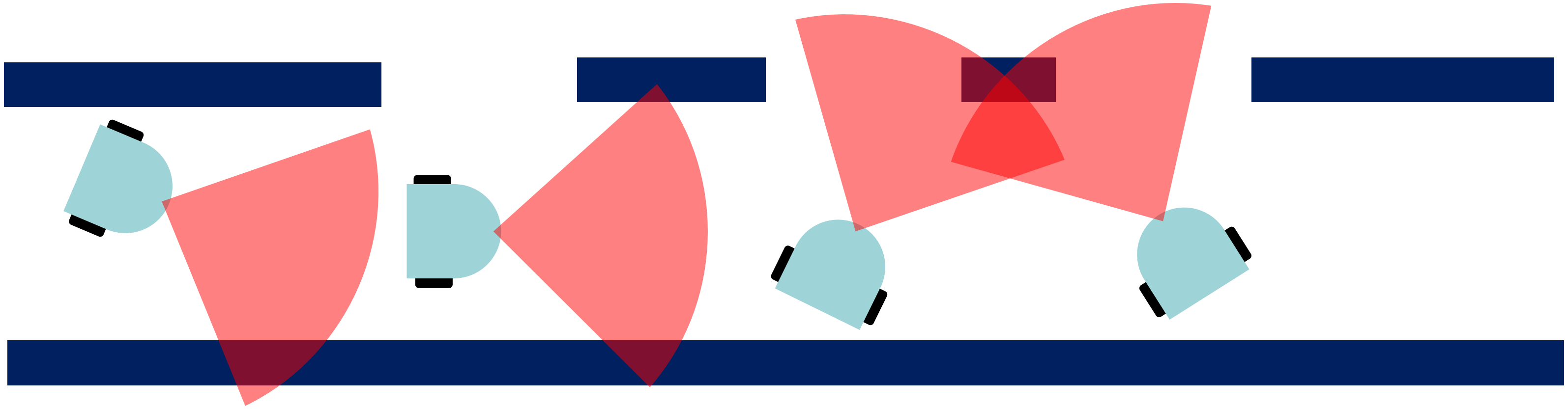
```
Extended_Kalman_filter ( x_{t-1}: ROBOT_POSE;
                         u_t: ROBOT_CONTROL;
                         z_t: SENSOR_MEASUREMENT ) : ROBOT_POSE
    do
        μ_{t-1} := x_{t-1}.mean
        Σ_{t-1} := x_{t-1}.covariance
Predict μ*_t := g(u_t, μ_{t-1})  -- g(u_t, x_{t-1}) = g(u_t, μ_{t-1}) + G_t (x_{t-1} - μ_{t-1})
        Σ*_t := G_t Σ_{t-1} G_t^T + R_t
        K_t := Σ*_t H_t^T (H_t Σ*_t H_t^T + Q_t)^{-1}
Update  μ_t := μ*_t + K_t (z_t – h(μ*_t)) -- h(x_t) = h(μ*_t) + H_t (x_t – μ*_t)
        Σ_t := (I - K_t H_t) Σ*_t

        Result := create {ROBOT_POSE}.make_with_variables( μ_t, Σ_t )
    end
```

Kalman filter localization

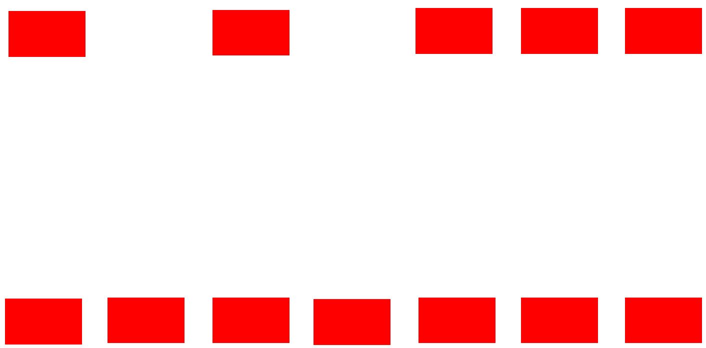
- Localization for linear systems
- Locally linearize update matrices for non-linear systems

- Unimodal model is not always realistic for many robot situations
- Matrix inversion is expensive
    - Limits the number of possible state values

What if we keep track of multiple robot pose?



Measurement

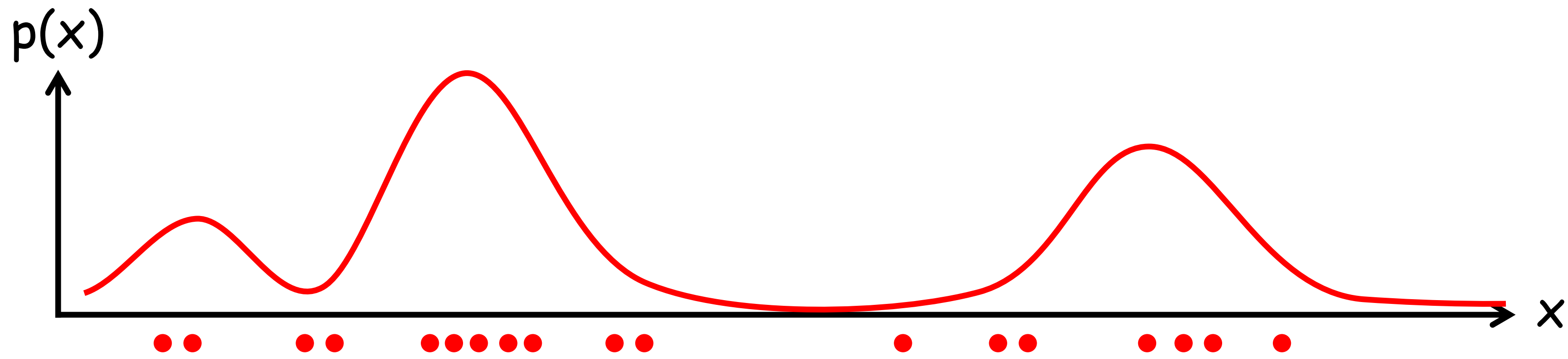


Particle filter

A sample-based Bayes filter

- Approximate the posterior bel(x_t) by a finite number of particles
- Each particle represents the probability of a particular state vector given all previous measurements
- The distribution of state vectors within the particle is representative of the probability distribution function for the state vector given all prior measurements

Importance sampling



Generate samples from a distribution

$$E_f[\ I(x \in A)\ ] = \int f(x)\ I(x \in A)\ dx$$
$$= \int f(x)/g(x)\ g(x)\ I(x \in A)\ dx$$
$$= E_g[\ w(x)\ I(x \in A)\ ]$$

$f(x)$: target distribution

$g(x)$: proposal distribution – $f(x) > 0 \rightarrow g(x) > 0$

Particle filter localization

```
particle_filter_localize ( X_{t-1}: ARRAY[ROBOT_POSE];
                           u_t: ROBOT_CONTROL;
                           z_t: SENSOR_MEASUREMENT;
                           m: MAP) : ARRAY[ROBOT_POSE]
    do
        from i := X_{t-1}.lower until i > X_{t-1}.upper loop
                x_{t-1} := X_{t-1}[i]
Predict         X_t[i].pose := motion_update( x_{t-1}, u_t, t_current - t_previous )
Update          X_t[i].weight := sensor_update(z_t, m)
                i := i + 1
        end
        Result := resample(X_t)
    end
```

Particle filter localization

- Global localization
    - Track the pose of a mobile robot without knowing the initial pose
- Can handle kidnapped robot problem with little modification
    - Insert some random samples at every iteration
    - Insert random samples proportional to the average likelihood of the particles
- Approximate
    - Accuracy depends the number of samples

Motion models

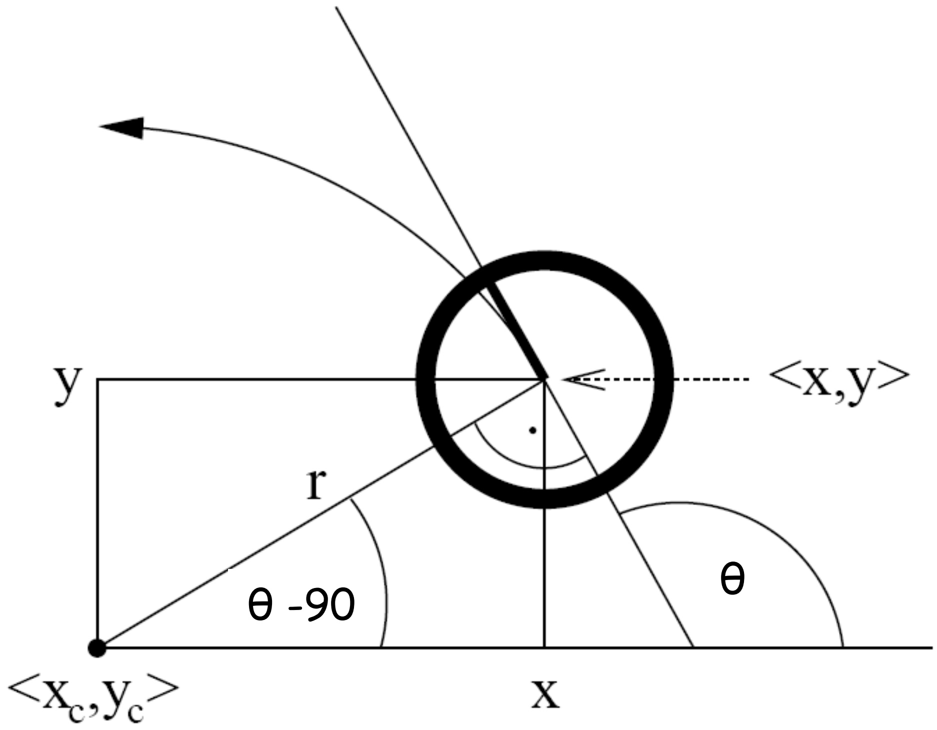
Velocity-based

- No wheel encoders are given.
- The new pose is based on the velocities and the time elapsed.

Odometry-based

- Systems are equipped with wheel encoders.

Velocity model



$$v = \omega * r$$

$$x_c = x - \frac{v}{\omega} \sin \theta$$

$$y_c = y + \frac{v}{\omega} \cos \theta$$

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

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

$$\theta' = \theta + \omega\,\Delta t$$

Sampling from velocity motion model

```
sample_motion_model_velocity ( x: ROBOT_POSE;
                               u: ROBOT_CONTROL
                               Δt: REAL_64 ) : ROBOT_POSE
    do
        u′.v := u.v + sample (α₁ u.σ_v² + α₂ u.σ_ω²)
        u′.ω := u.ω + sample (α₃ u.σ_v² + α₄ u.σ_ω²)
```

$$x'.x := x.x - \frac{u'.v}{u'.\omega} \sin(x.\theta) + \frac{u'.v}{u'.\omega} \sin(x.\theta + u'.\omega\, \Delta t)$$

$$x'.y := x.y + \frac{u'.v}{u'.\omega} \cos(x.\theta) - \frac{u'.v}{u'.\omega} \cos(x.\theta + u'.\omega\, \Delta t)$$

$$x'.\theta := x.\theta + u'.\omega\, \Delta t + \text{sample}(\alpha_5\, u.\sigma_v^2 + \alpha_6\, u.\sigma_\omega^2)\, \Delta t$$

```
        Result := x′
    end
```

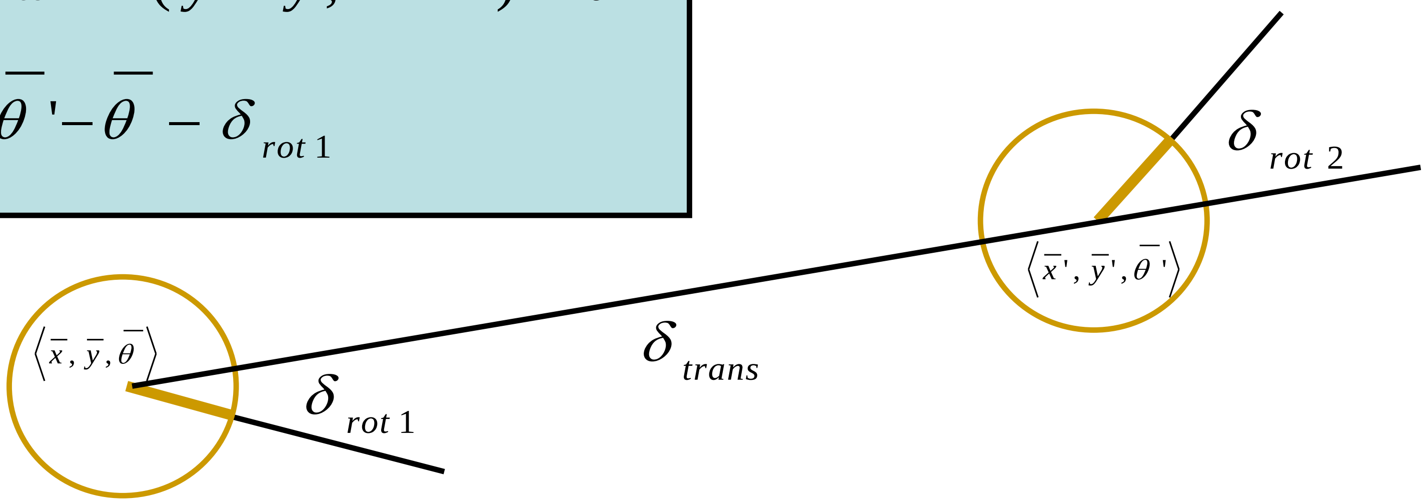
Odometry motion model

- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
- Odometry information $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2}$$

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$



Sampling from odometry motion model

sample_motion_model_velocity (x: ROBOT_POSE;
u: ROBOT_CONTROL
Δt: REAL_64) : ROBOT_POSE

do

$\delta_{rot1} := \text{atan2}(u.\bar{y}' - u.\bar{y}, u.\bar{x}' - u.\bar{x}) - u.\bar{\theta}$

$\delta_{trans} := \text{sqrt}((u.\bar{x} - u.\bar{x}')^2 + (u.\bar{y} - u.\bar{y}')^2)$

$\delta_{rot2} := u.\bar{\theta}' - u.\bar{\theta} - \delta_{trans}$

$\hat{\delta}_{rot1} := \delta_{rot1} + \text{sample}(\alpha_1 \delta^2_{rot1} + \alpha_2 \delta^2_{trans})$

$\hat{\delta}_{trans} := \delta_{trans} + \text{sample}(\alpha_3 \delta^2_{trans} + \alpha_4 \delta^2_{rot1} + \alpha_4 \delta^2_{rot2})$

$\hat{\delta}_{rot2} := \delta_{rot2} + \text{sample}(\alpha_1 \delta^2_{rot2} + \alpha_2 \delta^2_{trans})$

$x'.x := x.x + \hat{\delta}_{trans} \cos(x.\theta + \hat{\delta}_{rot1})$

$x'.y := x.y + \hat{\delta}_{trans} \sin(x.\theta + \hat{\delta}_{rot1})$

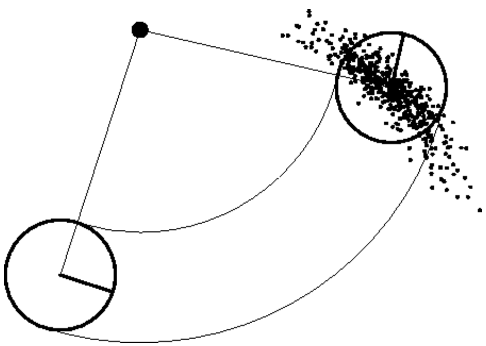
$x'.\theta := x.\theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

Result := x′

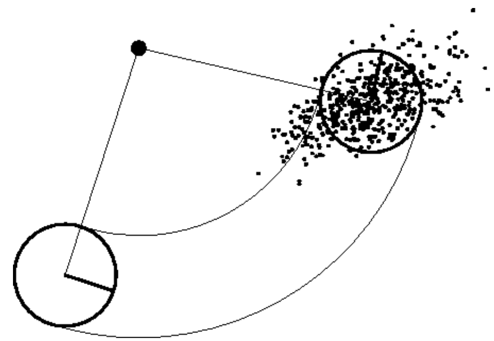
end

Effect of different noise parameter settings

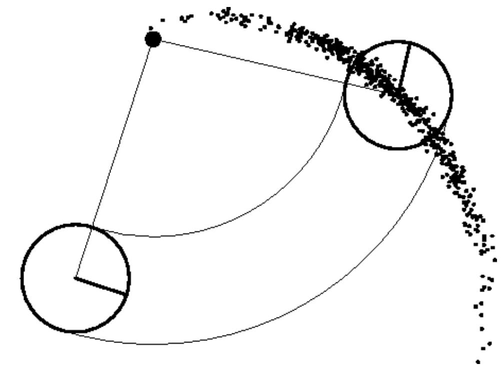
Velocity model



α_1 to α_6: moderate

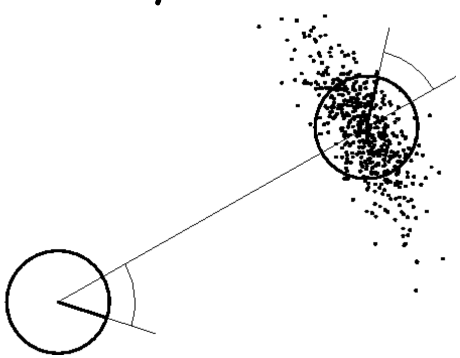


α_3 and α_4: small
α_1 and α_2: large

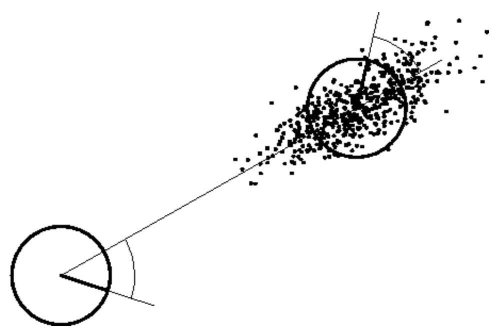


α_3 and α_4: large
α_1 and α_2: small

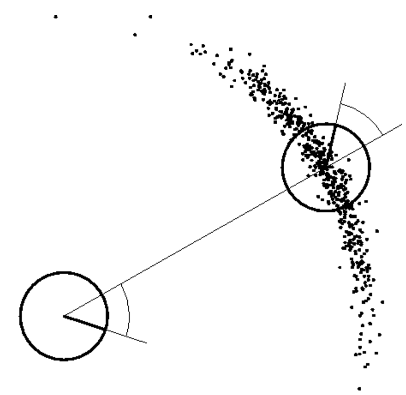
Odometry motion model



α_1 to α_4: moderate



α_1 and α_4: small
α_2 and α_3: large



α_1 and α_4: large
α_2 and α_3: small

Sensor models

Direct modeling of the sensor readings

Feature-based models

Likelihood fields

Project the end points of a sensor scan z_t into the map

- Measurement noise: Zero-centered Gaussian distribution
  - $p_{hit}(z^k_t \mid x_t, m) = \varepsilon_\sigma(dist)$
  - dist: distance between the measurement and the nearest obstacle in the map m
- Failures: Point-mass distribution
  - $$p_{max}(z^k_t \mid x_t, m) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases}$$
- Unexplained random measurements: Uniform distribution
  - $$p_{rand}(z^k_t \mid x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \le z^k_t \le z_{max} \\ 0 & \text{otherwise} \end{cases}$$

$p(z^k_t \mid x_t, m) = z_{hit}\, p_{hit} + z_{rand}\, p_{rand} + z_{max}\, p_{max}$

$z_{hit}, z_{rand}, z_{max}$: mixing weights

Likelihood fields

```
likelihood_field_range_finder ( x: ROBOT_POSE;
                                z: SENSOR_MEASUREMENT;
                                m: MAP ) : REAL_64
    do
        q := 1.0
        from  i := z.beam.lower  until  i > z.beam.upper loop
            if  z.beam[i].range < z_max then
                x_i := x.x + z.beam[i].x * cos(x.θ) – z.beam[i].y * sin(x.θ) +
                           z.beam[i].range * cos(x.θ + z.beam[i].θ)
                y_i := x.y + z.beam[i].y * cos(x.θ) + z.beam[i].x * sin θ +
                           z.beam[i].range * sin(x.θ + z.beam[i].θ)
                d := m.compute_distance_to_the_nearest_obstacle(x_i ,y_i)
                q := q · ( z_hit · prob(d, σ_hit) + z_rand / z_max )
            end
        end
        Result := q
    end
```

Measurement coordinate

$$q := q \cdot \left(z_{hit} \cdot prob(d, \sigma_{hit}) + \frac{z_{rand}}{z_{max}} \right)$$

Likelihood fields

Advantages

- Smooth
  - Small changes in the robot's pose result in small changes of the resulting distribution
- Computationally more efficient than ray casting

Disadvantages

- No modeling of dynamic objects
- Sensors can see through the wall
  - Nearest neighbor cannot determine if a path is obstructed by an obstacle
- No map uncertainty considered
  - Can change occupancy to occupied, free, and unknown

Correlation-based measurement model

Map matching

1. Compute a local map m_{robot} from the scans z_t in robot frame
2. Transform the local map m_{robot} to the global coordinate frame m_{local}
3. Compare the local map m_{local} and the map m

$$\rho = \frac{\Sigma_{x,y}(m_{x,y} - \bar{m}) \cdot (m_{x,y,local}(x_t) - \bar{m})}{\sqrt{\Sigma_{x,y}(m_{x,y} - \bar{m})^2 \, \Sigma_{x,y}(m_{x,y,local}(x_t) - \bar{m})^2}} \quad : \text{correlation}$$

$$\bar{m} = \frac{1}{2N} \Sigma_{x,y}(m_{x,y} + m_{x,y,local}) \quad : \text{average map value}$$

$$p(m_{local} \mid x_t, m) = \max\{\rho, 0\}$$

Correlation-based measurement model

Advantages

- Easy to compute
- Explicitly considers free-space

Disadvantages

- Does not yield smooth probability in pose x_t
  - May convolve the map m with a Gaussian kernel first
- Can incorporate inappropriate local map information
  - May contain areas beyond the maximum sensor range
- Does not include the noise characteristic of range sensors

Feature extraction

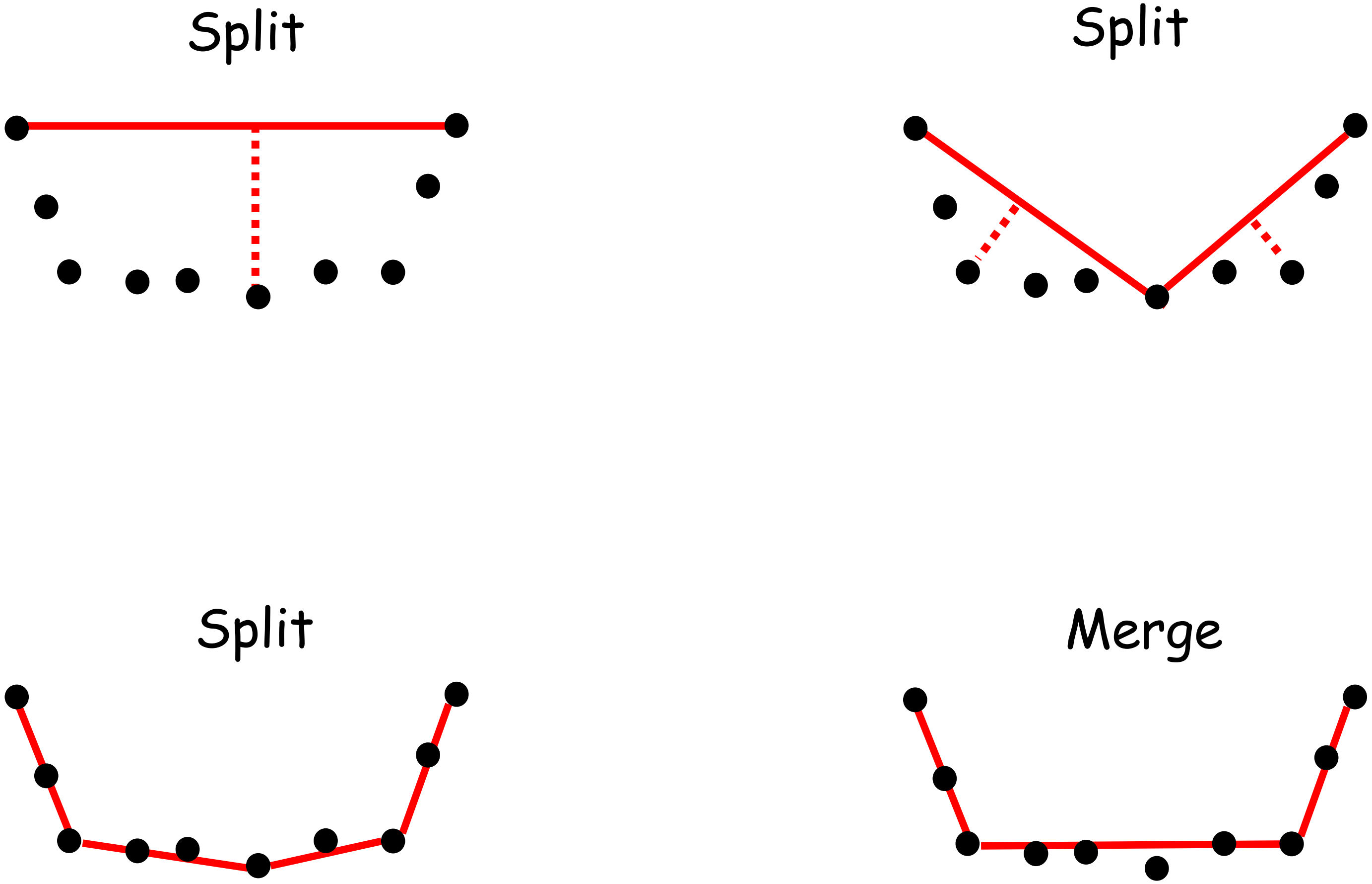
feature: compact representation of raw data

- Range scans: lines, corners, local minima in range scans, etc.
- Camera images: edges, corners, distinct patterns, etc.
- High level features in robotics: places

Advantages of using features

- Reduction of computational complexity
    - Increase in feature extraction
    - Decrease in feature matching

Feature extraction: split and merge



Feature extraction: split and merge

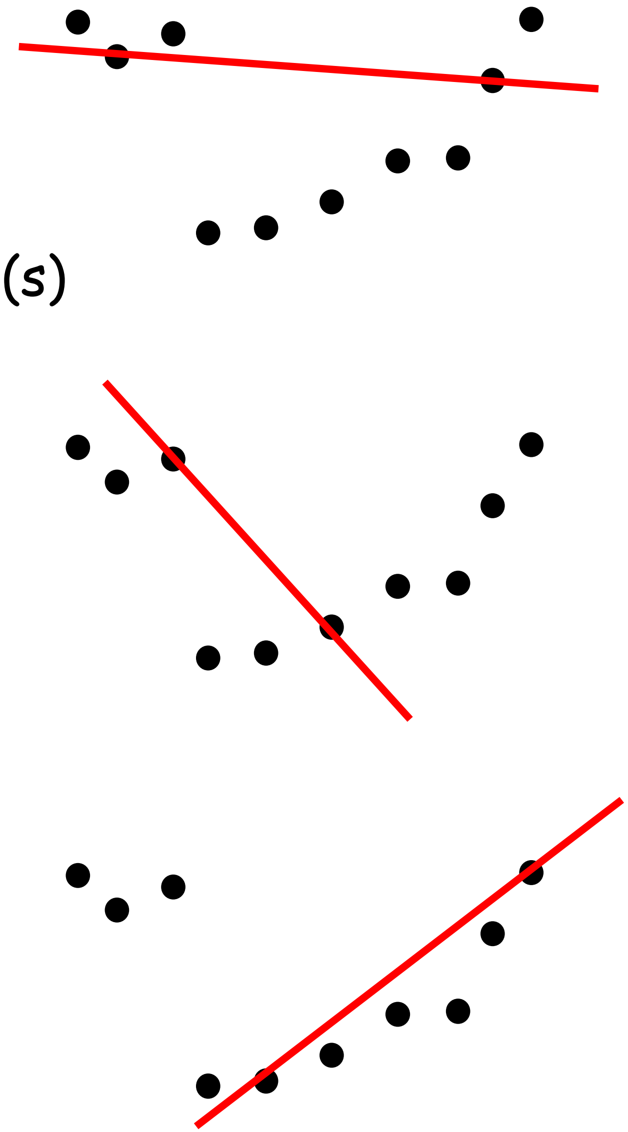
```
split( s: POINT_SET ) : LINE_SET -- sorted points
    do
        p_max := l.compute_farthest_point( s )
        if l.compute_distance( p_max ) > d_max then
            lines.add_set( split( s.split_set(1, p_max) ) )
            lines.add_set( split( s.split_set(p_max, l.size) ) )
        else
            lines.add( l )
        end
        Result := lines
    end
```

Feature extraction: split and merge

```
merge( lines: LINE_SET ) : LINE_SET
    do
        from until not lines.is_next_pair_collinear loop
                l.merge_lines( lines.left_line , lines.right_line )
                if l.compute_distance( l.compute_farthest_point ) < dmax then
                    out_lines.add(l)
                    lines.mark_current_pair_as_used
                end
                lines.increment_next_pair
        end
        out_lines.add_set( lines.get_all_unmarked_lines )
        Result := out_lines
    end
```

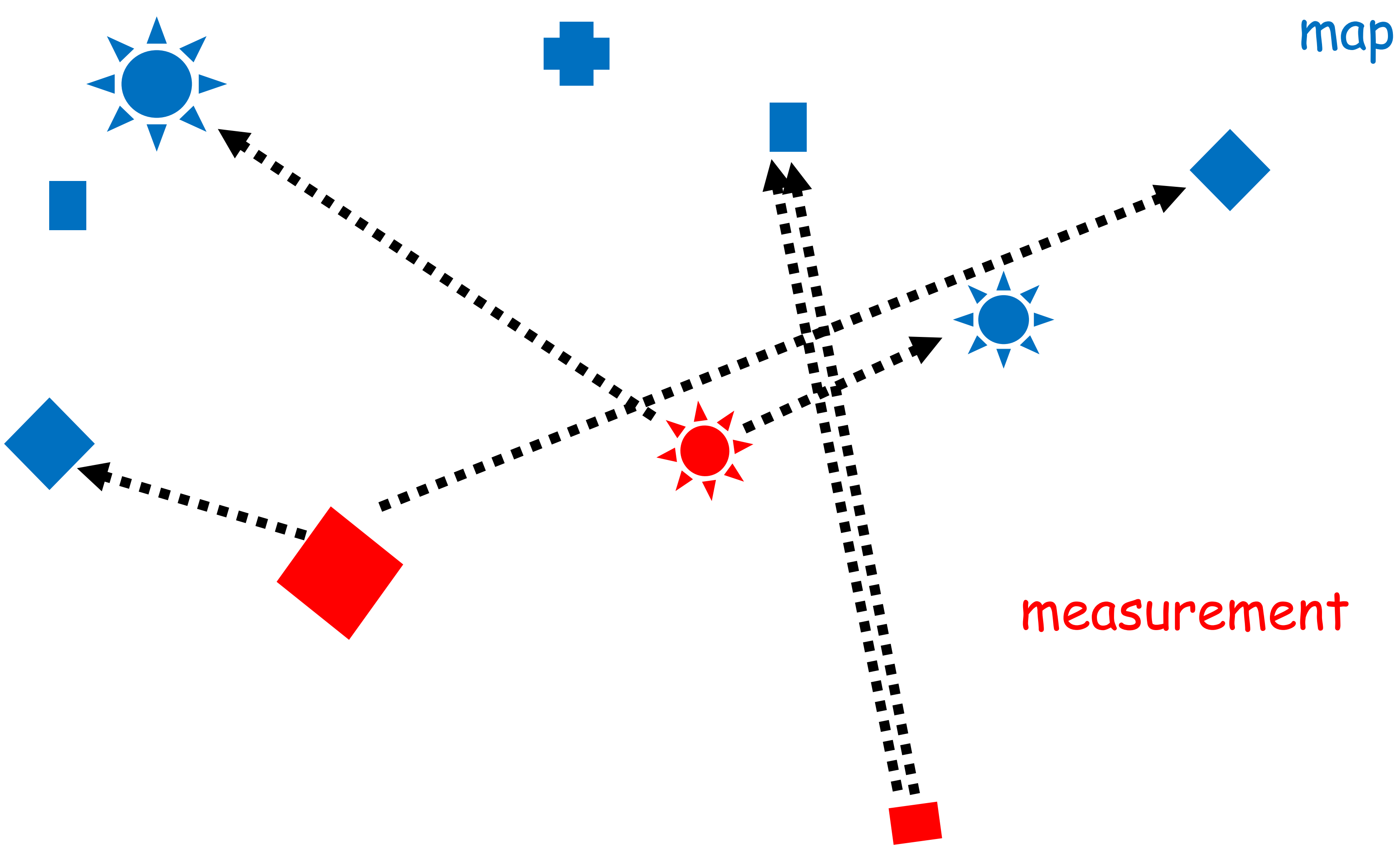
Feature extraction: RANSAC

```
RANSAC( s: POINT_SET ) : LINE
    do
        from c := 1 until c > c_max loop
            l.set_line_from_two_random_points(s)
            if l.count_inliners > num then
                num := l.count_inliners
                line := l
            end
        end
        Result := line
    end
```



Fischler, M. and Bolles, R. 1981. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Communications of the ACM. 24(6).

Data association



Data association: nearest neighbor

Measurement: $F = \{f_1, \ldots, f_n\}$

Map features: $M = \{m_1, \ldots, m_l\}$

```
nearest_neighbor( F, M: ARRAY[FEATURE] ) : HYPOTHESIS
    do
        from i := 1 until i > n loop
            f_i := F.item(i)
            d_min := d_min.Max_value
            from j := 1 until j > l loop
                m_j := M.item(j)
                d_temp := Mahalanobis2(f_i, m_j)
                if d_temp < d_min then
                    d_min := d_temp
                    m_nearest := m_j
                end
            end
            if d_min < X^2(d_i, α) then -- d_i = dim(z_i), α: desired confidence level
                H.add_pair(f_i, m_nearest)
            else
                H.add_pair(f_i, 0)
            end
        end
        Result := H
    end
```

Data association: joint compatibility

```
joint_compatibility(H: HYPOTHESIS; i: INTEGER_16; F, M: ARRAY[FEATURE] )
    do
        f_i := F.item(i)
        if i > l then
            if H.score > Best.score then
                Best := H
            end
        else
            from j := 1 until j > l loop
                m_j := M.item(j)
                if is_compatible(f_i, m_j) and H.is_joint_compatible(f_i, m_j) then
                    joint_compatibility(H.add_pair(f_i, m_j) , i+1, F, M)
                end
            end
             if H.score + n – i >= Best.score then -- can do better?
                joint_campatibility(H.add_pair(f_i, 0), i+1, F, M)
             end
        end
    end
```

Measurement: $F = \{f_1, \ldots, f_n\}$

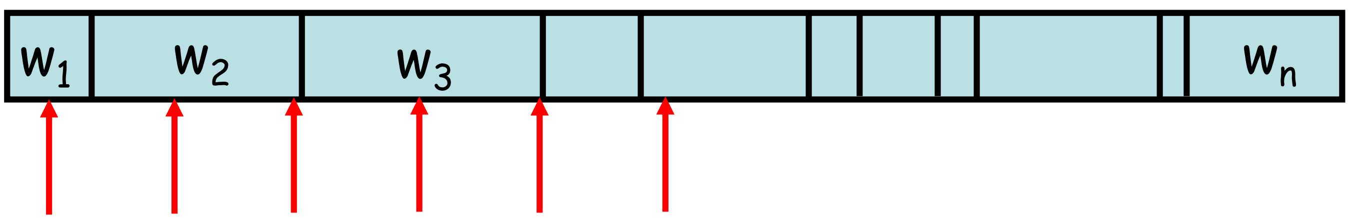
Map features: $M = \{m_1, \ldots, m_l\}$

Neira, J. Tardos, J.D. 2001. "Data association in stochastic mapping using the joint compatibility test", Robotics and Automation, IEEE Transactions on 17 (6): 890–897.

Resampling

Roulette wheel sampling

Stochastic universal sampling



distance between two samples = total weight / number of samples

starting sample: random number in [0, distance between samples)

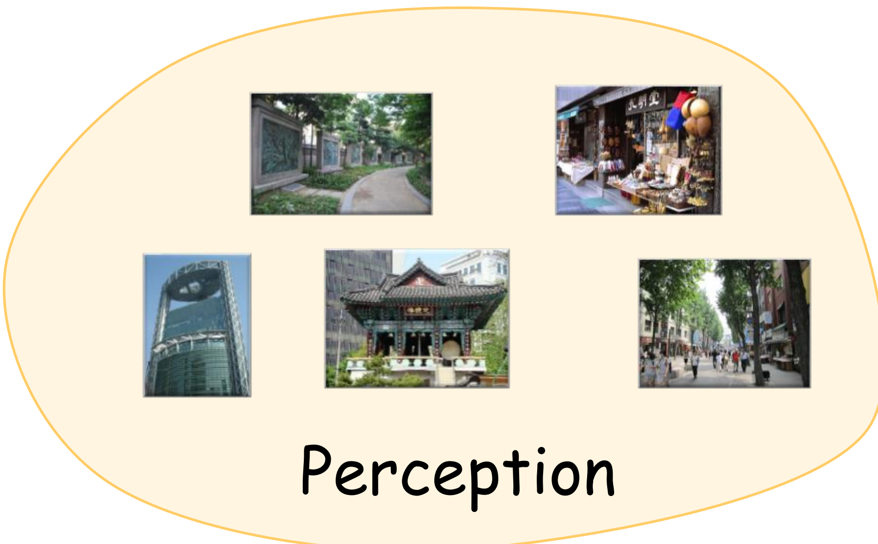
Mapping

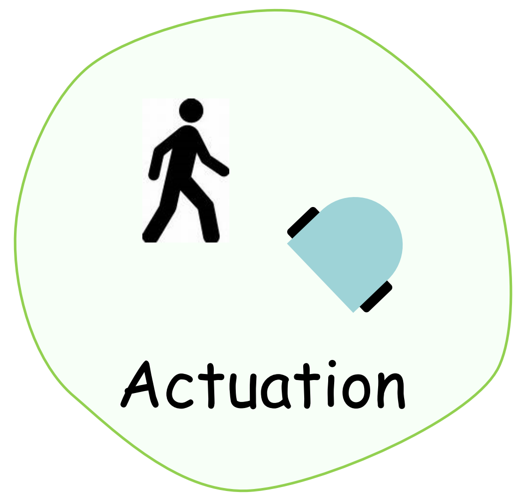
Map: a list of objects and their locations in an environment

- Given N objects in an environment

$$m = \{ m_1, \dots , m_N \}$$

Mapping: the process of creating a map





Types of Maps

Lacation-based map

- $m = \{ m_1, \ldots , m_N \}$ contains N locations
- Volumetric representation
  - A label for any location in the world
  - Knowledge of presence and absence of objects

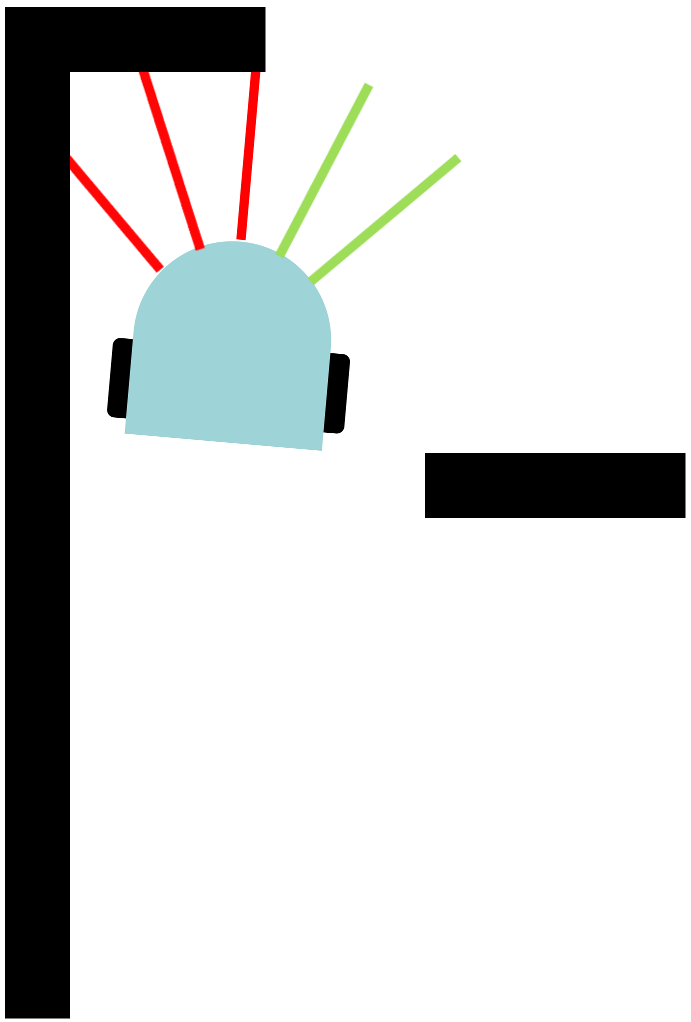
Feature-based map

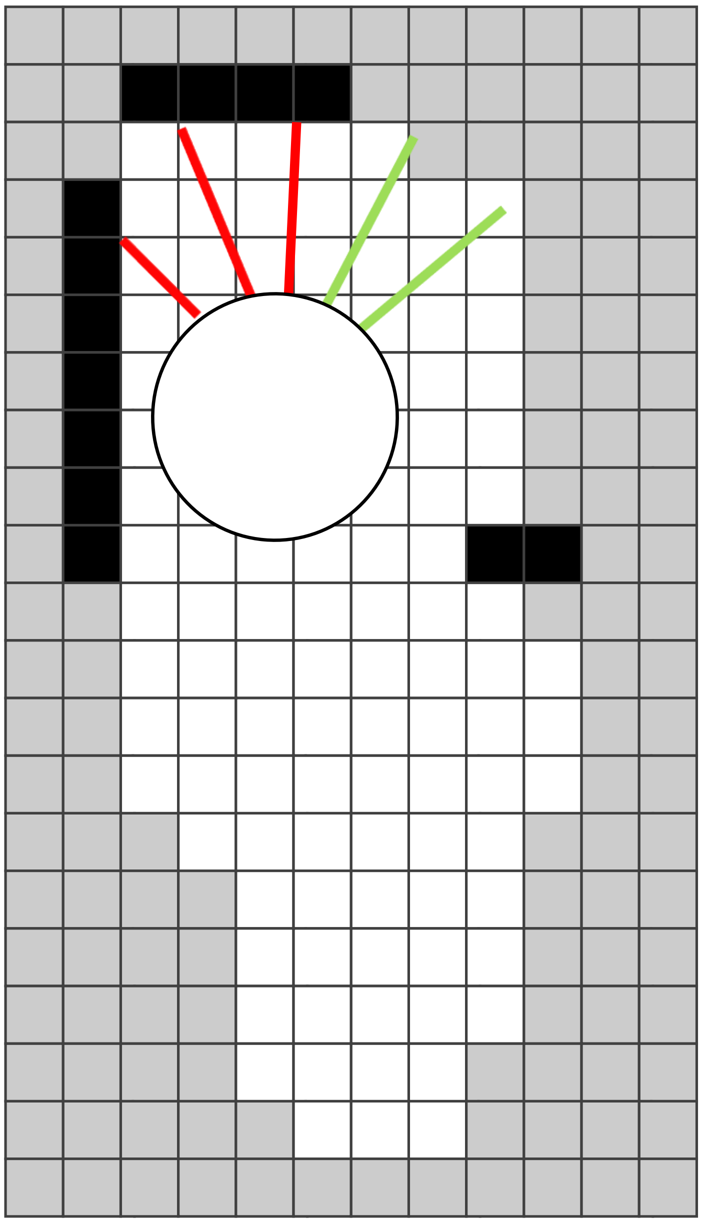
- $m = \{ m_1, \ldots , m_N \}$ contains N features
- Sparse representation
  - A label for each object location
  - Easier to adjust the position of an object

Occupancy grid map

- Location-based map
- An environment as a collection of grid cells
- Each grid cell with a probability value that the cell is occupied
  - Each grid cell is independent!
- Easy to combine different sensor scans and different sensor modalities
- No assumption about type of features

Occupancy grid mapping





Occupancy grid cells

$\mathbf{m}_i$: the grid cell with index i

z_t: the measurement at time t

x_t: the robot's pose (x, y, θ) at time t

$p(\mathbf{m}_i \mid z_t, x_t)$: probability of occupancy

$$\frac{p(\mathbf{m}_i \mid z_t, xt)}{p(\neg \mathbf{m}_i \mid z_t, xt)} = \frac{p(\mathbf{m}_i \mid z_t, xt)}{1 - p(\mathbf{m}_i \mid z_t, xt)}$$: odds of occupancy

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid z_t, xt)}{1 - p(\mathbf{m}_i \mid z_t, xt)}$$: log odds of occupancy

$$p(\mathbf{m}_i \mid z_t, x_t) = 1 - \frac{1}{1 + \exp(lt_{,i})}$$

Bayes' law using log odds

$$p(A|B) = \frac{p(B|A)\,p(A)}{p(B)}$$

$$p(\neg A|B) = \frac{p(B|\neg A)\,p(\neg A)}{p(B)}$$

$$o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A)\,p(A)}{p(B|\neg A)\,p(\neg A)} = \lambda(B|A)\,o(A)$$

$$\log(\,o(A|B)\,) = \log(\,\lambda(B|A)\,) + \log(\,o(A)\,)$$

- Ranges between -∞ and ∞
- Avoids truncation problem around probabilities near 0 and 1

Occupancy grid mapping

```
occupancy_grid_mapping ( x: ROBOT_POSE;
                         z: SENSOR_MEASUREMENT;
                         m: MAP )
    do
        from i := m.cell.lower until i > m.cell.upper loop
            if m.cell[i].is_in_perceptiual_field(z) then
                m.log_odds[i] := m.log_odds[i] +
                        inverse_sensor_model (m.cell[i], x, z ) - l0
            end
        end
    end
```

$$m.log\_odds[i] := \log \frac{p(\ m.cell[i] \mid xi_{:t},\ zi_{:t}\ )}{1 - p(\ m.cell[i] \mid x_{i:t},\ zi_{:t}\ )}$$

$$l_0 := \log \frac{p(\ m.cell[i] = 1\ )}{p(\ m.cell[i] = 0\ )} := \log \frac{p(\ m.cell[i]\ )}{1 - p(\ m.cell[i]\ )}$$

Occupancy grid mapping

```
inverse_range_sensor_model ( x: ROBOT_POSE;
                             z: SENSOR_MEASUREMENT;
                             g: GRID_CELL) : REAL_64
    do
        x_i := g.center_of_mass.x
        y_i := g.center_of_mass.y
        r := √( (x_i - x.x)^2 + (y_i - x.y)^2 )                    grid range
        φ := atan2(y_i - x.y, x_i - x.x) - x.θ                     grid angle
        k := argmin_j | φ - z.beam[j].θ |                          beam index
        if r > min( z_max, z.beam[k].range + α/2 ) or | φ - z.beam[k].θ | > β/2 then
                Result := l_0            grid out of range or behind an obstacle
        elseif z.beam[k].range < z_max and | r - z.beam[k].range | < α/2 then
                Result := l_occ                                    grid in the obstacle
        else -- r <= z.beam[k]
                Result := l_free                                   grid unaccupied
        end
    end
```

α: thickness of the obstacle
β: opening angle of the beam
z_{max}: max range of the beam

But what about drift?

Localization

- If we have a map, we can localize

Mapping

- If we know the robot's pose, we can map

Do both!

- Estimate a map
- Localize itself relative to the map

Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping

Localization: $p(x \mid m, z, u)$

Mapping: $p(m \mid x, z)$

SLAM: $p(x, m \mid z, u)$

- The map depends on the robot's pose during the measurement
- If the pose is known, mapping is easy

Rao-Blackwellization

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

SLAM posterior = robot path posterior * mapping with known poses

$p(x_{1:t} \mid z_{1:t}, u_{0:t-1})$: localization

$p(m \mid x_{1:t}, z_{0:t-1})$: mapping

$x_{1:t}$: the robot's poses (x, y, θ)

m: the map

$z_{1:t}$: the measurements

$u_{0:t-1}$: the controls

Murphy, K. 1999. Bayesian map learning in dynamic environments. In NIPS '99 (Neural Info. Proc. Systems)

Rao-Blackwellized particle filter SLAM

Use a particle filter to represent potential trajectories of the robot

- Every particle carries its own map
- The probability of survival of a particle is proportional to the likelihood of the measurement with respect to the particle's own map

Problem: big map * large number of particles!

Improve pose estimate

- Use scan matching to compute locally consistent pose correction
- Smaller error -> fewer particles necessary