ETH Zürich
Chair of Software Engineering

Robotics Programming Laboratory

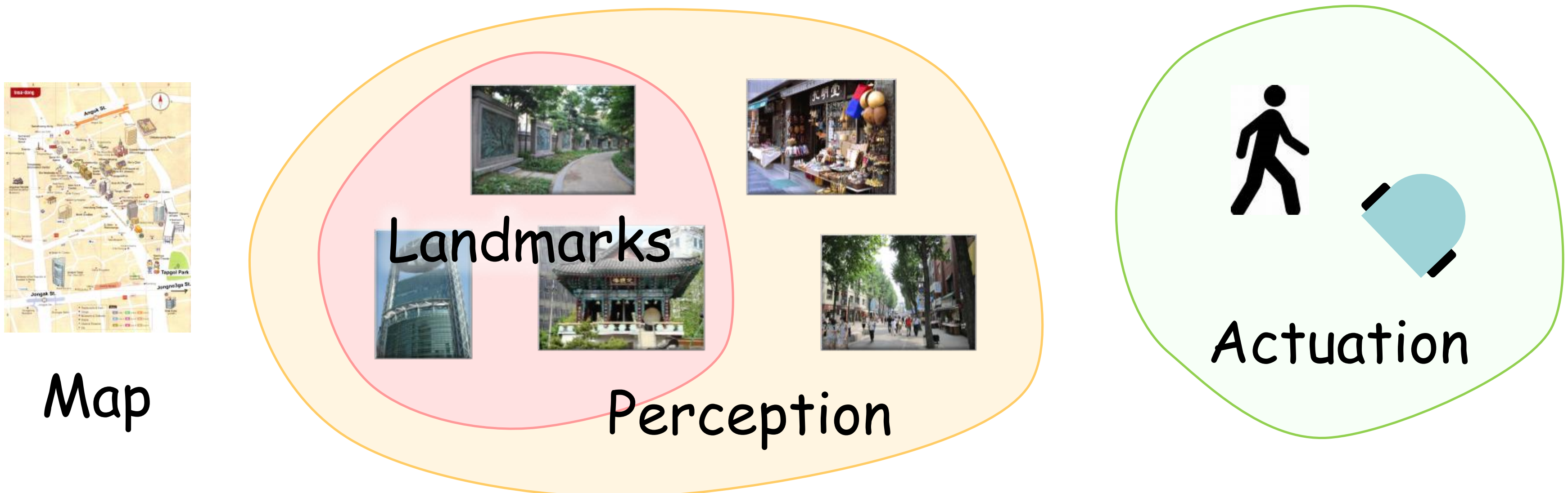
Bertrand Meyer
Jiwon Shin

Lecture 6: Localization

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

Localization

Localization: process of locating an object in space



Types of localization

- Global localization: initial pose unknown
  - Markov localization
  - Particle filter localization
- Local localization: initial pose known
  - Kalman filter 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)
    - $P(X = x) \geq 0$
    - $\sum_X P(X = x) = 1$ or $\int_X p(X = x) = 1$

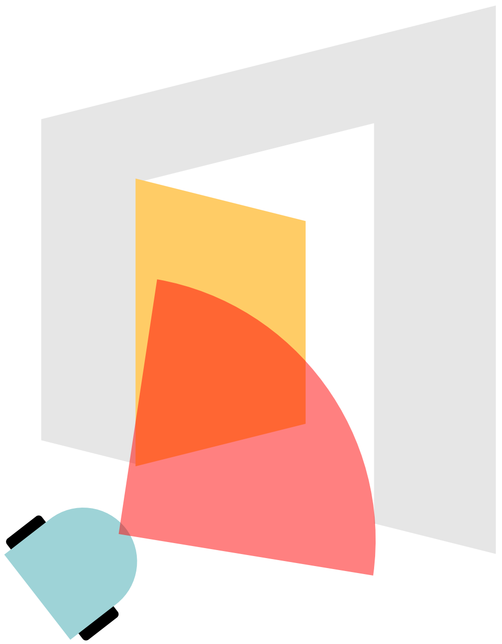
Probability

- $P(x,y)$: 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



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

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

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

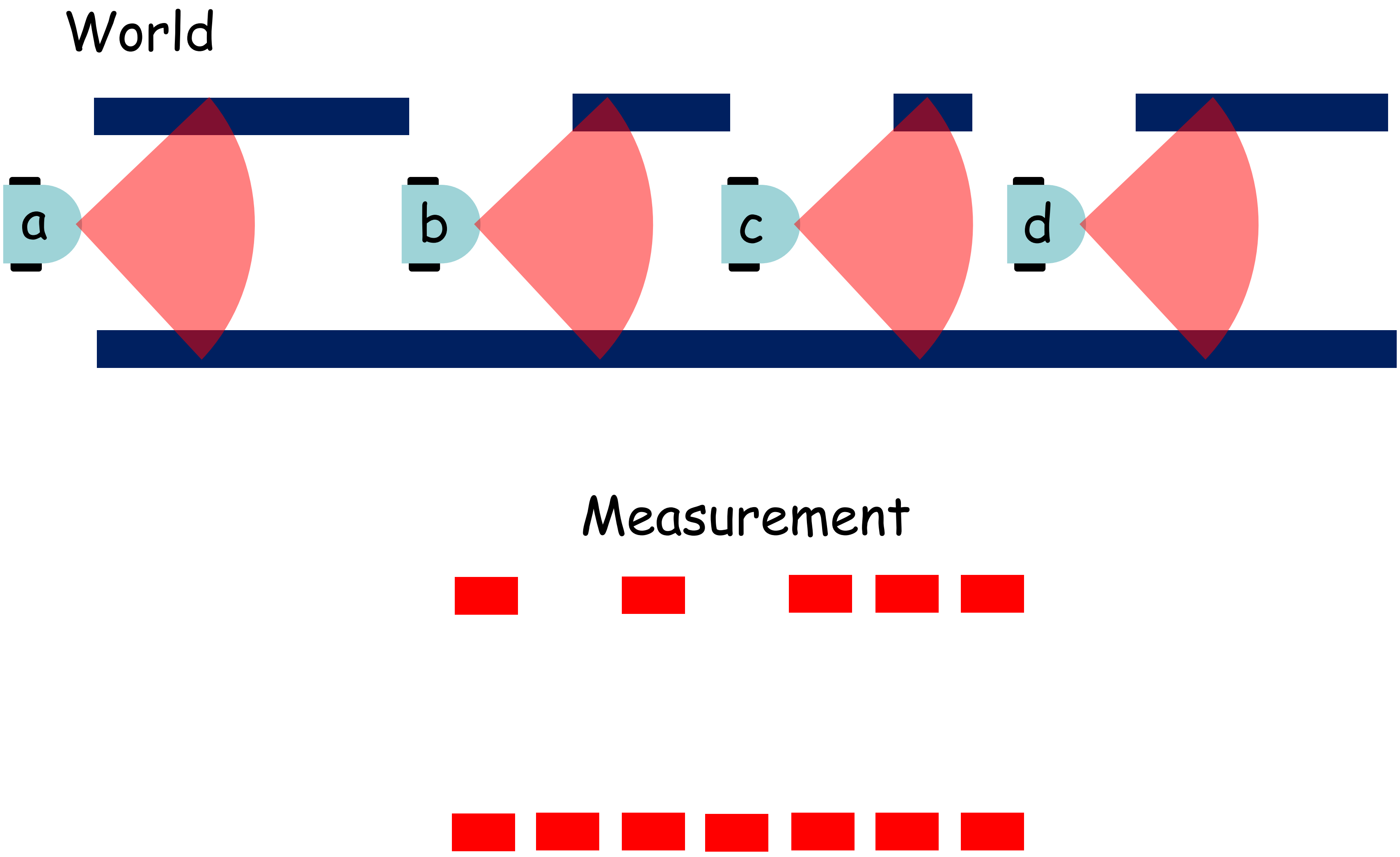
Bayes' filter

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

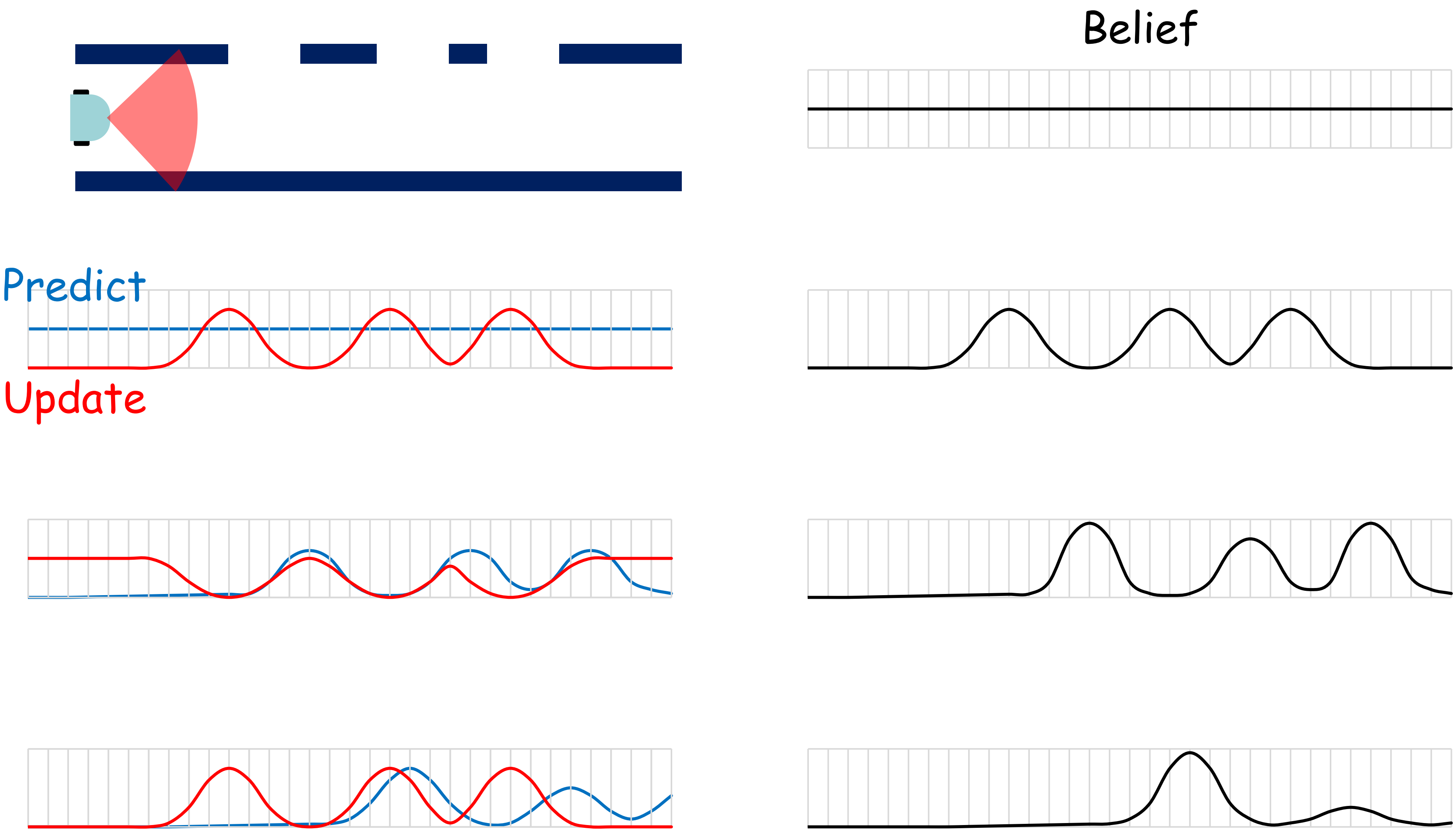
Compute robot's state: $bel(x_t)$

- Predict where the robot should be based on the control $u_{1:t}$
- Update the robot state using the measurement $z_{1:t}$

Markov localization



Markov localization



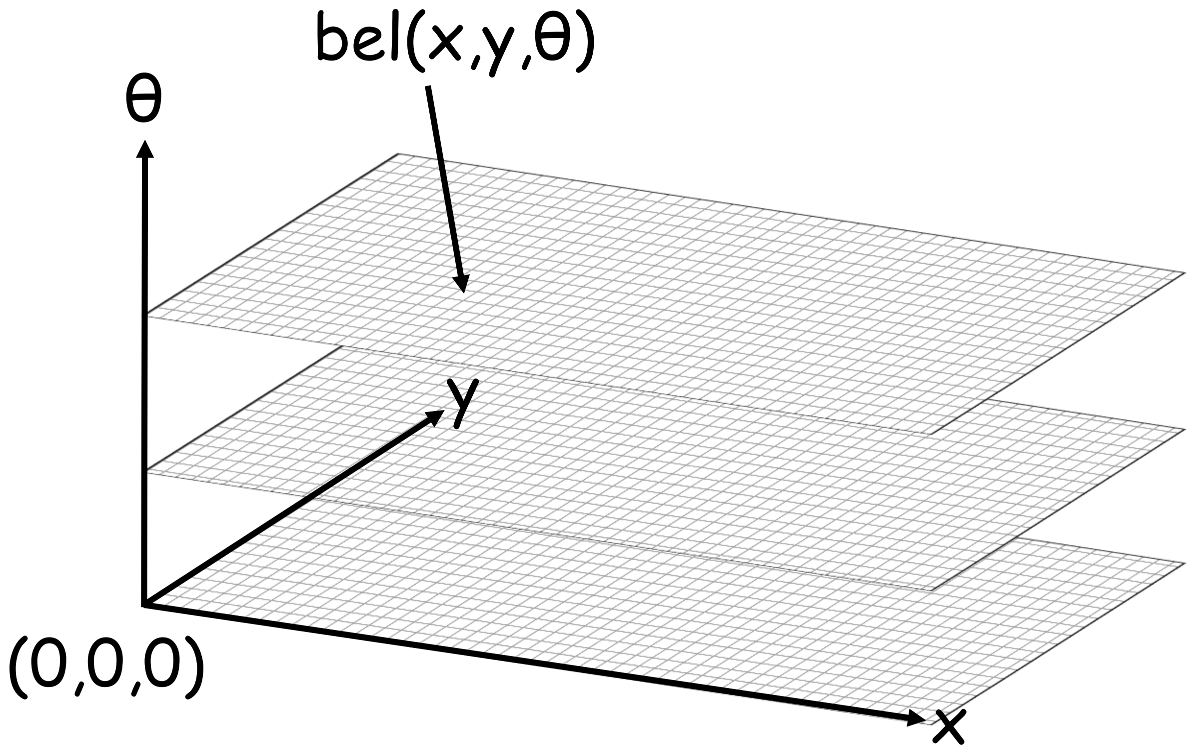
Markov localization

```
Markov_localize ( bel_{t-1}: ARRAY[BELIEF_ROBOT_POSE];
                  u_t: ROBOT_CONTROL;
                  z_t: SENSOR_MEASUREMENT;
                  m: MAP) : BELIEF_ROBOT_POSE
    local
        bel*_t : ARRAY[BELIEF_ROBOT_POSE_PARTICLE]
        bel_t : ARRAY[BELIEF_ROBOT_POSE_PARTICLE]
        x_t : ROBOT_POSE
    do
        create bel*_t.make_from_array( bel_{t-1} )
        create bel_t.make_from_array( bel_{t-1} )
        from  i := bel_t.lower  until  i > bel_t.upper loop
                  x_t := bel_t[i].pose
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

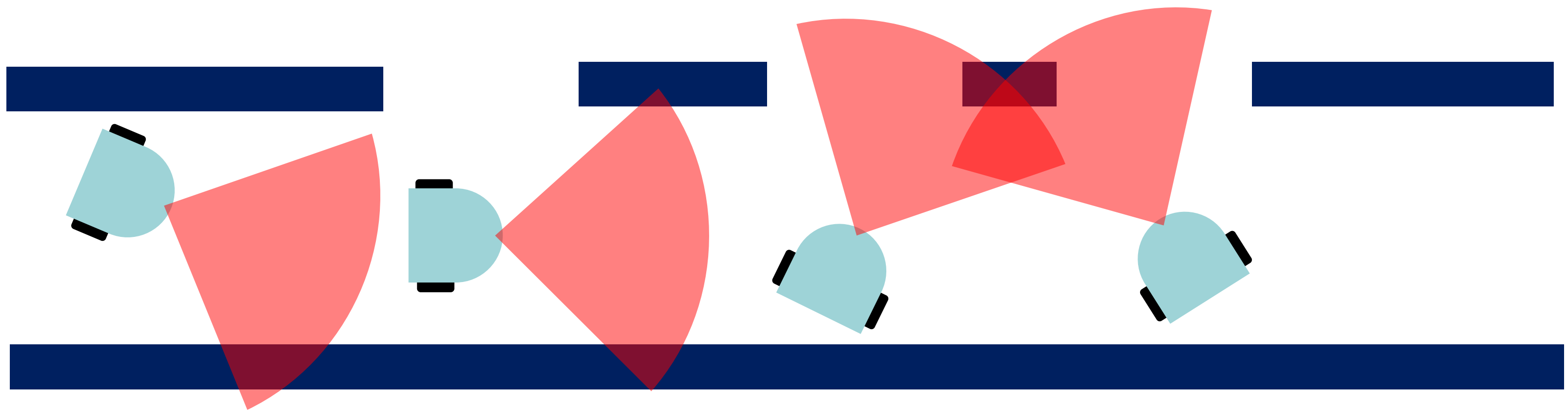


Markov localization

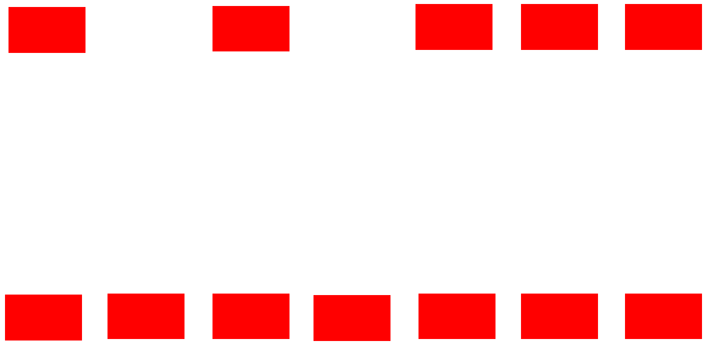
- Can be used for both local localization and global localization
  - If the initial pose (x^*_0) is known: point-mass distribution
    - $bel(x_0) = \begin{cases} 1 & \text{if } x_0 = x*_0 \\ 0 & \text{otherwise} \end{cases}$
  - If the initial pose (x^*_0) is known with uncertainty Σ:

    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 keep track of multiple robot pose?



Measurement

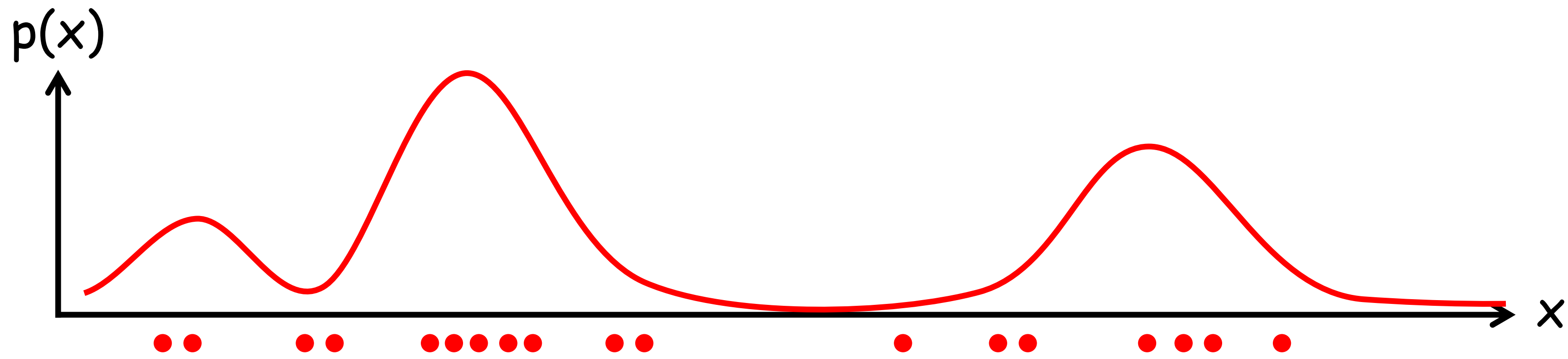


Particle filter

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

Sampling from motion model

```
sample_motion_mode ( x: ROBOT_POSE;
                     u: ROBOT_CONTROL
                     Δt: REAL_64 ) : ROBOT_POSE
    local
        x': ROBOT_POSE
        u': ROBOT_CONTROL
    do
```

$u'.v := \text{Gaussian\_sample}(\ u.v,\ \alpha_1\, u.\sigma_v^2 + \alpha_2\, u.\sigma_\omega^2\ )$

$u'.\omega := \text{Gaussian\_sample}(\ u.\omega,\ \alpha_3\, u.\sigma_v^2 + \alpha_4\, u.\sigma_\omega^2)$

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

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

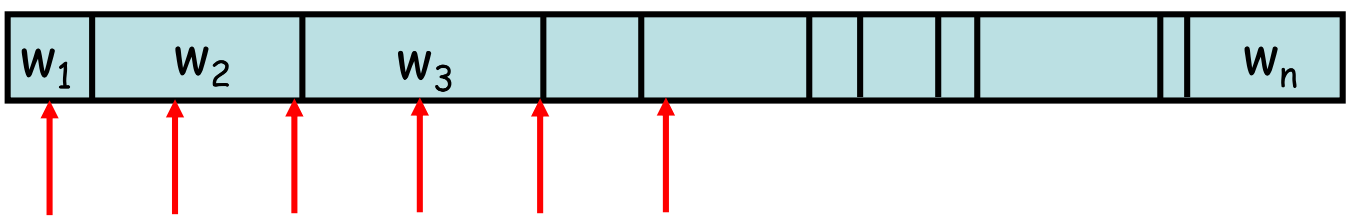
$$x'.\theta := x.\theta + u'.\omega\Delta t + \text{Gaussian\_sample}(\ 0,\ \alpha_5\, u.\sigma_v^2 + \alpha_6\, u.\sigma_\omega^2\ )\ \Delta t$$

```
        Result := x'
    end
```

Resampling

Roulette wheel sampling

Stochastic universal sampling



distance between two samples = total weight / number of samples

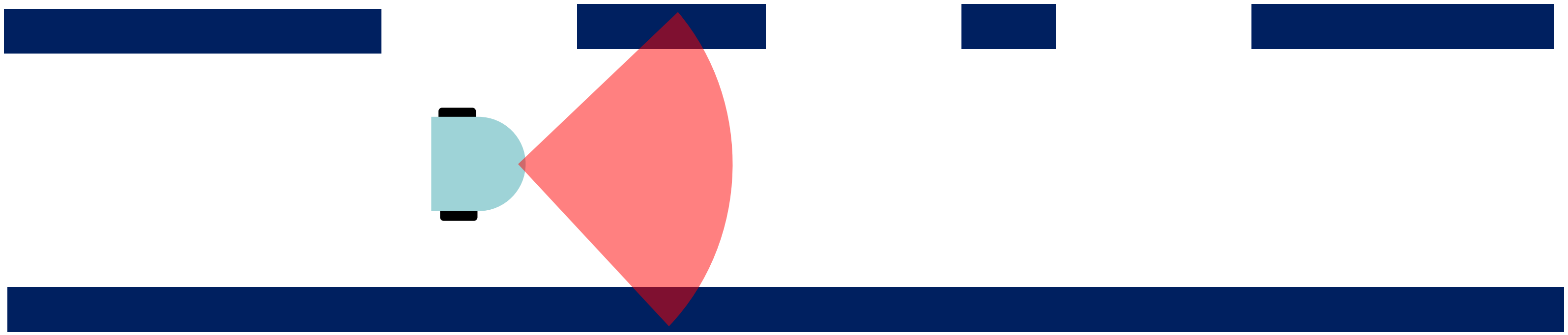
starting sample: random number in [0, distance between samples]

Particle filter localization

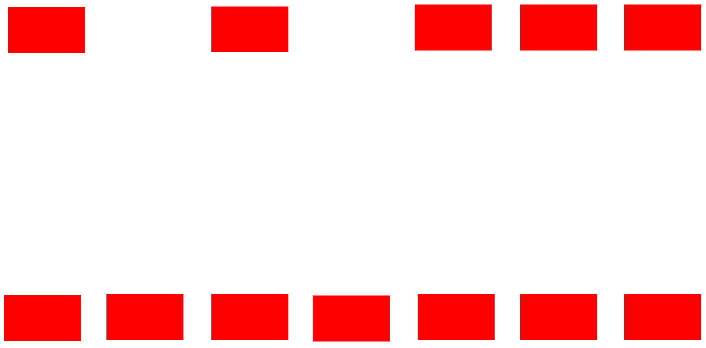
- Global localization
  - Track the pose of a mobile robot without knowing the initial pose
- Can handle kidnapped robot problem with little modification
  - Insert some random samples at every iteration
  - Insert random samples proportional to the average likelihood of the particles
- Approximate
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

If we know the initial pose, can we do better?

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

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

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

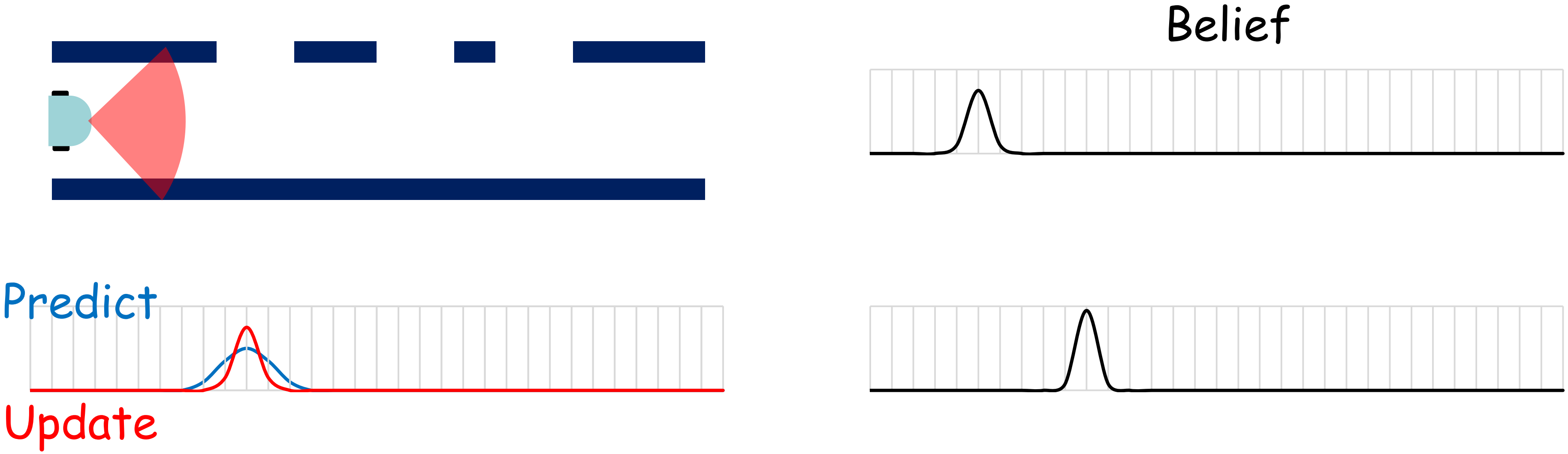
Kalman filter localization

A special case of Markov localization

Assumptions:

- The system is linear (describable as a system of linear equations)
- The noise in the system has a Gaussian distribution
- The error criteria is expressed as a quadratic equation (e.g. sum-squared error)

Kalman filter localization



Kalman filter

```
Kalman_filter ( x_{t-1}: ROBOT_POSE;
                u_t: ROBOT_CONTROL;
                z_t: SENSOR_MEASUREMENT ) : ROBOT_POSE
    local
        μ_{t-1}, μ*_t, μ_t : MEAN_ROBOT_POSE
        Σ_{t-1}, Σ*_t, Σ_t : ROBOT_POSE_COVARIANCE
        K_t : KALMAN_GAIN
    do
        μ_{t-1} := x_{t-1}.mean
        Σ_{t-1} := x_{t-1}.covariance
Predict μ*_t := A_t μ_{t-1} + B_t u_t
        Σ*_t := A_t Σ_{t-1} A_t^T + R_t
        K_t := Σ*_t C_t^T (C_t Σ*_t C_t^T + Q_t)^{-1}
Update  μ_t := μ*_t + K_t (z_t - C_t μ*_t)
        Σ_t := (I - K_t C_t) Σ*_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
    local
        μ_{t-1}, μ*_t, μ_t : MEAN_ROBOT_POSE
        Σ_{t-1}, Σ*_t, Σ_t : ROBOT_POSE_COVARIANCE
        K_t : KALMAN_GAIN
    do
        μ_{t-1} := x_{t-1}.mean
        Σ_{t-1} := x_{t-1}.covariance
Predict μ*_t := g(u_t, μ_{t-1})  -- linearized state transition : g(u_t, x_{t-1}) = g(u_t, x_{t-1}) + G_t (x_{t-1} - u_{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)) -- linearized measurement: h(x_t) = h(u*_t) + H_t (x_t – u*_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

- Local localization
- 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