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).
Localization: process of locating an object in space

Map

Landmarks

Perception

Actuation
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 $P(x)$
  - $P(X = x) \geq 0$
  - $\sum_X P(X = x) = 1$ or $\int_X p(X = x) = 1$
Probability

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

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

- $P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}} : \text{Bayes' rule}$

- $P(y) = \sum_x P(x,y) = \sum_x P(y \mid x)P(x) : \text{Law of total probability}$
Bayes’ rule

\[
P(\text{door=open | sensor=far}) = \frac{P(\text{far | open}) P(\text{open})}{P(\text{far})}
\]

\[
= \frac{P(\text{far | open}) P(\text{open})}{P(\text{far | open}) P(\text{open}) + P(\text{far | closed}) P(\text{closed})}
\]
Bayes’ filter

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

Compute robot’s state: \( \text{bel}(x_t) \)

- Predict where the robot should be based on the control \( u_{1:t} \)
  \[ \text{bel}^*(x_t) = \int p(x_t \mid u_t, x_{t-1}) \text{bel}(x_{t-1}) \, dx_{t-1} \]

- Update the robot state using the measurement \( z_{1:t} \)
  \[ \text{bel}(x_t) = \eta \, p(z_t \mid x_{t-1}) \text{bel}^*(x_t) \]
Markov localization

World

Measurement
Markov localization

Predict

Update

Belief
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] := \int p(x_t | u_t, x_{t-1}, m) \, bel_{t-1}(x_{t-1}) \, dx_{t-1}

            Update bel_{t}[i] := \eta \, p(z_t | x_{t-1}, m) \, bel^*_{t}[i]

            i := i + 1
        end
    end

    Result := bel_t

end
Representation of the robot states

$\text{bel}(x, y, \theta)$

(0,0,0)
Markov localization

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

\[
X \sim N (\mu, \sigma^2) \quad \Rightarrow \quad Y \sim N (a\mu + b, a^2 \sigma^2)
\]

\[
X_1 \sim N (\mu_1, \sigma_1^2) \quad \Rightarrow \quad p(X_1) \cdot p(X_2) \sim N\left(\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \mu_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \mu_2, \frac{1}{\sigma_1^{-2} + \sigma_2^{-2}}\right)
\]

Multivariate

\[
X \sim N (\mu, \Sigma) \quad \Rightarrow \quad Y \sim N (A\mu + B, A\Sigma A^T)
\]

\[
X_1 \sim N (\mu_1, \Sigma_1) \quad \Rightarrow \quad p(X_1) \cdot p(X_2) \sim N\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2} \mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2} \mu_2, \frac{1}{\Sigma_1^{-1} + \Sigma_2^{-1}}\right)
\]
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

\[ x_t = A_t \, x_{t-1} + B_t \, u_t + \epsilon_t \]

Linear observation model

\[ z_t = C_t \, x_t + \delta_t \]
Kalman filter localization

Predict

Update

Belief
Kalman filter

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

do

\[ \mu_{t-1} := x_{t-1}.\text{mean} \]
\[ \Sigma_{t-1} := x_{t-1}.\text{covariance} \]

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( \mu_t, \Sigma_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 \( \mu_t \) with measurement

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

- estimate uncertainty of \( \mu_t \)

\( C_t \): measurement matrix relating the state variable and measurement
\( Q_t \): measurement noise covariance
Extended Kalman filter

\[
\text{Extended\_Kalman\_filter} \left( x_{t-1}; \text{ROBOT\_POSE}; \\
    u_t; \text{ROBOT\_CONTROL}; \\
    z_t; \text{SENSOR\_MEASUREMENT} \right) : \text{ROBOT\_POSE}
\]

do

\[
\mu_{t-1} := x_{t-1}.\text{mean} \\
\Sigma_{t-1} := x_{t-1}.\text{covariance}
\]

**Predict**

\[
\mu^*_t := g(u_t, \mu_{t-1}) \quad -- \quad g(u_t, x_{t-1}) = g(u_t, \mu_{t-1}) + G_t (x_{t-1} - \mu_{t-1}) \\
\Sigma^*_t := G_t \Sigma_{t-1} G_t^T + R_t
\]

\[
K_t := \Sigma^*_t H_t^T (H_t \Sigma^*_t H_t^T + Q_t)^{-1}
\]

**Update**

\[
\mu_t := \mu^*_t + K_t (z_t - h(\mu^*_t)) \quad -- \quad h(x_t) = h(\mu^*_t) + H_t (x_t - \mu^*_t) \\
\Sigma_t := (I - K_t H_t) \Sigma^*_t
\]

\[
\text{Result} := \text{create} \{\text{ROBOT\_POSE}\}.\text{make\_with\_variables}(\mu_t, \Sigma_t)
\]
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 \( \text{bel}(x_t) \) 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

\[ E_f[ I(x \in A) ] = \int f(x) I(x \in A) \, dx \]
\[ = \int \frac{f(x)}{g(x)} g(x) I(x \in A) \, dx \]
\[ = E_g[ w(x) I(x \in A) ] \]

\( f(x) \): target distribution
\( g(x) \): proposal distribution \(- f(x) > 0 \rightarrow g(x) > 0 \)
Particle filter localization

\[
\text{particle\_filter\_localize} \ (X_{t-1}; \ \text{ARRAY[ROBOT\_POSE]}; \\
\quad u_t; \ \text{ROBOT\_CONTROL}; \\
\quad z_t; \ \text{SENSOR\_MEASUREMENT}; \\
\quad m; \ \text{MAP}) : \ \text{ARRAY[ROBOT\_POSE]}
\]

do
\[
\text{from} \quad i := X_{t-1}.\text{lower} \quad \text{until} \quad i > X_{t-1}.\text{upper} \quad \text{loop}
\]
\[
x_{t-1} := X_{t-1}[i]
\]

**Predict**
\[
X_t[i].\text{pose} := \text{motion\_update}(x_{t-1}, u_t, t_{current} - t_{previous})
\]

**Update**
\[
X_t[i].\text{weight} := \text{sensor\_update}(z_t, m)
\]
\[
i := i + 1
\]
end

**Result** := \text{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.σᵥ² + α₂ u.σω²)
    u'.ω := u.ω + sample (α₃ u.σᵥ² + α₄ u.σω²)

    x'.x := x.x - u'.v / u'.ω sin (x.θ) + u'.v / u'.ω sin (x.θ + u'.ω Δt)
    x'.y := x.y + u'.v / u'.ω cos (x.θ) - u'.v / u'.ω cos (x.θ + u'.ω Δt)
    x'.θ := x.θ + u'.ω Δt + sample (α₅ u.σᵥ² + α₆ u.σω²) Δt

    Result := x'

    end
Odometry motion model

- Robot moves from \( \langle x, y, \theta \rangle \) to \( \langle x', y', \theta' \rangle \)
- Odometry information
  \[
  u = \left\langle \delta_{rot\;1}, \delta_{rot\;2}, \delta_{trans} \right\rangle
  \]

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

\[
\delta_{rot\;1} = \text{atan2} \left( y' - y, x' - x \right) - \theta
\]

\[
\delta_{rot\;2} = \theta' - \theta - \delta_{rot\;1}
\]
Sampling from odometry motion model

sample_motion_model_velocity ( x: ROBOTPOSE; u: ROBOTCONTROL Δt: REAL_64 ) : ROBOTPOSE

do

δ<sub>rot1</sub> := atan2 (u.<i>y</i>' - u.<i>y</i>, u.<i>x</i>' - u.<i>x</i>) - u.<i>θ</i>
δ<sub>trans</sub> := sqrt( (u.<i>x</i> - u.<i>x</i>')<sup>2</sup> + (u.<i>y</i> - u.<i>y</i>')<sup>2</sup> )
δ<sub>rot2</sub> := u.<i>θ</i>' - u.<i>θ</i> - δ<sub>trans</sub>

δ̂<sub>rot1</sub> := δ<sub>rot1</sub> + sample (α<sub>1</sub><i>δ<sub>rot1</sub></i><sup>2</sup> + α<sub>2</sub><i>δ<sub>trans</sub></i><sup>2</sup>)
δ̂<sub>trans</sub> := δ<sub>trans</sub> + sample (α<sub>3</sub><i>δ<sub>trans</sub></i><sup>2</sup> + α<sub>4</sub><i>δ<sub>rot1</sub></i><sup>2</sup> + α<sub>4</sub><i>δ<sub>rot2</sub></i><sup>2</sup>)
δ̂<sub>rot2</sub> := δ<sub>rot2</sub> + sample (α<sub>1</sub><i>δ<sub>rot2</sub></i><sup>2</sup> + α<sub>2</sub><i>δ<sub>trans</sub></i><sup>2</sup>)

<x<sub>′</sub>.<i>x</i> := <i>x</i>.<i>x</i> + δ̂<sub>trans</sub> cos (x.<i>θ</i> + δ̂<sub>rot1</sub>)
<x<sub>′</sub>.<i>y</i> := <i>x</i>.<i>y</i> + δ̂<sub>trans</sub> sin (x.<i>θ</i> + δ̂<sub>rot1</sub>)
<x<sub>′</sub>.<i>θ</i> := x.<i>θ</i> + δ<sub>rot1</sub> + δ<sub>rot2</sub>
Result := x<sub>′</sub>

end
Effect of different noise parameter settings

Velocity model

\(\alpha_1\) to \(\alpha_6\): moderate
\(\alpha_3\) and \(\alpha_4\): small
\(\alpha_1\) and \(\alpha_2\): large

Odometry motion model

\(\alpha_1\) to \(\alpha_4\): moderate
\(\alpha_1\) and \(\alpha_4\): small
\(\alpha_2\) and \(\alpha_3\): large
\(\alpha_1\) and \(\alpha_4\): large
\(\alpha_2\) and \(\alpha_3\): 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
  - $p_{hit}(z^k_t \mid x_t, m) = \epsilon_\sigma(\text{dist})$
  - dist: distance between the measurement and the nearest obstacle in the map $m$

- **Failures**: Point-mass distribution
  - $p_{\text{max}}(z^k_t \mid x_t, m) = \begin{cases} 1 & \text{if } z = z_{\text{max}} \\ 0 & \text{otherwise} \end{cases}$

- **Unexplained random measurements**: Uniform distribution
  - $p_{\text{rand}}(z^k_t \mid x_t, m) = \begin{cases} \frac{1}{z_{\text{max}}} & \text{if } 0 \leq z^k_t \leq z_{\text{max}} \\ 0 & \text{otherwise} \end{cases}$

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

$z_{\text{hit}}, z_{\text{rand}}, z_{\text{max}}$ : 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
                Measurement coordinate
                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) + \frac{Z_{rand}}{Z_{max}} )
            end
        end
    end

    Result := q
end
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 $m_{\text{robot}}$ from the scans $z_t$ in robot frame
2. Transform the local map $m_{\text{robot}}$ to the global coordinate frame $m_{\text{local}}$
3. Compare the local map $m_{\text{local}}$ and the map $m$

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

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

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

\[
\text{split} \ (s: \text{POINT\_SET}) : \text{LINE\_SET} \ -- \text{sorted points}
\]

\[
do
\begin{align*}
p_{max} & := l.\text{compute\_farthest\_point}(s) \\
\text{if } l.\text{compute\_distance}(p_{max}) > d_{max} \text{ then} & \\
& \quad \text{lines.add\_set( split( s.split\_set(1, p_{max}) ) )} \\
& \quad \text{lines.add\_set( split( s.split\_set(p_{max}, l.size) ) )} \\
\text{else} & \\
& \quad \text{lines.add( l )}
\end{align*}
\]

\[
\text{Result} := \text{lines}
\]

end
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 ) < d_{\text{max}} then
                out_lines.add(l)
                lines.mark_current_pair_as_used
            end
        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

Data association

map

measurement
Data association: nearest neighbor

`nearest_neighbor( F, M: ARRAY[FEATURE] ) : HYPOTHESIS`

```
do
    from i := 1 until i > n loop
        fi := F.item(i)
        d_min := d_min.Max_value
    from j := 1 until j > l loop
        mj := M.item(j)
        d_temp := Mahalanobis2(fi, mj)
        if d_temp < d_min then
            d_min := d_temp
            m_nearest := mj
        end
    end
    end
    if d_min < X^2(d_i, a) then -- d_i = dim(z_i), a: desired confidence level
        H.add_pair(fi, m_nearest)
    else
        H.add_pair(fi, 0)
    end
end
Result := H
end```

Measurement: F = \{f_1, ..., f_n\}

Map features: M = \{m_1, ..., m_l\}
Data association: joint compatibility

\[
\text{joint\_compatibility}(H: \text{HYPOTHESIS}; i: \text{INTEGER}\_16; F, M: \text{ARRAY[FEATURE]}) \]
\[
\begin{align*}
\text{do} \\
& f_i := F.\text{item}(i) \\
& \text{if } i > l \text{ then} \\
& \quad \text{if } H.\text{score} > \text{Best.\text{score}} \text{ then} \\
& \quad \quad \text{Best} := H \\
& \quad \text{end} \\
& \text{else} \\
& \quad \text{from } j := 1 \text{ until } j > l \text{ loop} \\
& \quad \quad m_j := M.\text{item}(j) \\
& \quad \quad \text{if } \text{is\_compatible}(f_i, m_j) \text{ and } H.\text{is\_joint\_compatible}(f_i, m_j) \text{ then} \\
& \quad \quad \quad \text{joint\_compatibility}(H.\text{add\_pair}(f_i, m_j), i+1, F, M) \\
& \quad \text{end} \\
& \text{end} \\
& \quad \text{if } H.\text{score} + n - i >= \text{Best.\text{score}} \text{ then} \quad \text{-- can do better?} \\
& \quad \quad \text{joint\_compatibility}(H.\text{add\_pair}(f_i, 0), i+1, F, M) \\
& \quad \text{end} \\
& \text{end} \\
& \text{end}
\end{align*}
\]

Measurement: \( F = \{f_1, \ldots, f_n\} \)

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

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, \ldots , m_N \}

Mapping: the process of creating a map
Types of Maps

Location-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

\( m_i \): the grid cell with index \( i \)
\( z_t \): the measurement at time \( t \)
\( x_t \): the robot’s pose \((x, y, \theta)\) at time \( t \)

\[ p(m_i \mid z_t, x_t) \] : probability of occupancy

\[
\frac{p(m_i \mid z_t, x_t)}{p(\neg m_i \mid z_t, x_t)} = \frac{p(m_i \mid z_t, x_t)}{1 - p(m_i \mid z_t, x_t)} : \text{odds of occupancy}
\]

\[ l_{t,i} = \log \frac{p(m_i \mid z_t, x_t)}{1 - p(m_i \mid z_t, x_t)} : \text{log odds of occupancy} \]

\[ p(m_i \mid z_t, x_t) = 1 - \frac{1}{1 + \exp(l_{t,i})} \]
Bayes’ law using log odds

\[
p(A|B) = \frac{p(B|A) p(A)}{p(B)}
\]

\[
p(\neg A|B) = \frac{p(B|\neg A) p(\neg A)}{p(B)}
\]

\[
o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A) p(A)}{p(B|\neg A) p(\neg A)} = \omega(B|A) o(A)
\]

\[
\log( o(A|B) ) = \log( \omega(B|A) ) + \log( o(A) )
\]

- Ranges between \(-\infty\) and \(\infty\)
- Avoids truncation problem around probabilities near 0 and 1
Occupancy grid mapping

```plaintext
do
    from i := m.cell.lower until i > m.cell.upper loop
        if m.cell[i].is_in_perceptual_field(z) then
            m.log_odds[i] := m.log_odds[i] + inverse_sensor_model (m.cell[i], x, z) - l_0
        end
    end
end
```

\[ m.log\_odds[i] := \log \frac{p( m.cell[i] | x_{i:t}, z_{i:t} )}{1 - p( m.cell[i] | x_{i:t}, z_{i:t} )} \]

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

\[
\text{inverse\_range\_sensor\_model}\ (x: \text{ROBOT\_POSE}; \\
\hspace{1cm} z: \text{SENSOR\_MEASUREMENT}; \\
\hspace{1cm} g: \text{GRID\_CELL}): \text{REAL\_64}
\]

do
\[
\begin{align*}
\hspace{1cm} x_i & := g.\text{center\_of\_mass}.x \\
\hspace{1cm} y_i & := g.\text{center\_of\_mass}.y
\end{align*}
\]
\[
\begin{align*}
\hspace{1cm} r & := \sqrt{(x_i - x.x)^2 + (y_i - x.y)^2) \\
\hspace{1cm} \varphi & := \text{atan2}(y_i - x.y, x_i - x.x) - x.\theta
\end{align*}
\]
\[
\begin{align*}
\hspace{1cm} k & := \arg\min_j | \varphi - z.\text{beam}[j].\theta |
\end{align*}
\]
\[
\text{if } r > \min( z_{\text{max}}, z.\text{beam}[k].\text{range} + \alpha/2 ) \text{ or } | \varphi - z.\text{beam}[k].\theta | > \beta/2 \text{ then}
\]
\[
\begin{align*}
\hspace{1cm} \text{Result} & := l_0 \\
& \quad \text{grid out of range or behind an obstacle}
\end{align*}
\]
\[
\text{elseif } z.\text{beam}[k].\text{range} < z_{\text{max}} \text{ and } | r - z.\text{beam}[k].\text{range} | < \alpha/2 \text{ then}
\]
\[
\begin{align*}
\hspace{1cm} \text{Result} & := l_{\text{occ}} \\
& \quad \text{grid in the obstacle}
\end{align*}
\]
\[
\text{else } -- r <= z.\text{beam}[k]
\]
\[
\begin{align*}
\hspace{1cm} \text{Result} & := l_{\text{free}} \\
& \quad \text{grid unoccupied}
\end{align*}
\]
end
end

\[\alpha: \text{thickness of the obstacle} \]
\[\beta: \text{opening angle of the beam} \]
\[z_{\text{max}}: \text{max range of the beam} \]
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

\[ p( x_{1:t}, m | z_{1:t}, u_{0:t-1} ) = p( x_{1:t} | z_{1:t}, u_{0:t-1} ) p( m | x_{1:t}, z_{0:t-1} ) \]

SLAM posterior = robot path posterior * mapping with known poses

\[ p( x_{1:t} | z_{1:t}, u_{0:t-1} ): \text{localization} \]
\[ p( m | x_{1:t}, z_{0:t-1} ): \text{mapping} \]

\( x_{1:t} \): the robot's poses \((x, y, \theta)\)
\( m \): the map
\( z_{1:t} \): the measurements
\( u_{0:t-1} \): the controls

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 \(\rightarrow\) fewer particles necessary