Robotics: From Basic Terminology to Monte Carlo Localization

Todd W. Neller

Robotics

- Robot – physical agent that performs tasks by sensing and manipulating the physical world
  - Sensors – input sensor data from environment for state estimation
  - Effectors – output control actions for actuating effectors to change state and/or environment
  - Maximizing expected utility – generate and execute control outputs that affect change so as to maximize rewards
- Environment complexity: partially observable, stochastic, people acting in environment (prediction needed), continuous state and action spaces, sometimes high-dimensional, real-time
- Outside of computation, most aspects of robotics struggle with the challenges of ***uncertainty***.

Localization

- “Where am I?” “What’s the situation?”
- Given:
  - Initial pose (state)
  - Environment map
  - Sensor/measurement model
  - Motion model
- Keep track of the most likely current state as the robot moves and senses in the environment.

Monte Carlo Localization

- Monte Carlo Localization
  - Initial set of random hypotheses about possible poses (i.e. states, particles)
  - As the robot moves and senses, a Darwinian survival-of-the-fittest process tends to reproduce the most likely hypotheses and tends to kill off the least likely.
  - Evolution of a cloud of hypotheses where the center, i.e. average, is the most likely robot pose.

MCL Algorithm

NOTE: These two *m* variables refer to the environment *map*, and not to the for-loop control variable that serves as an index to the poses/weights.

Algorithm MCL($\mathcal{X}_{t-1}, u_t, z_t, m$)**:**
    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
    for $m = 1$ to M do
        $x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
        $w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    endfor
    for $m = 1$ to M do
        draw i with probability $\propto w_t^{[i]}$
        add $x_t^{[i]}$ to $\mathcal{X}_t$
    endfor
    return $\mathcal{X}_t$

Figure 1: Pseudocode for MCL (Thrun, Burgard, and Fox 2005)

Motion Model

$$p(x_t \mid u_t, x_{t-1})$$

- u_t – motion control command at time t - 1 to affect time t
- $x_{t\text{-}1}$ – pose at time t - 1
- x_t – pose at time t
- Model p(x_t | u_t , $x_{t\text{-}1}$) – the probability of the pose being x_t given prior motion control command u_t and prior pose $x_{t\text{-}1}$

Example: 2D Robot with Rotation

- Simple resting pose (state): $(x, y, \theta)^T$
- Control translational and rotational velocities: $(v, \omega)^T$

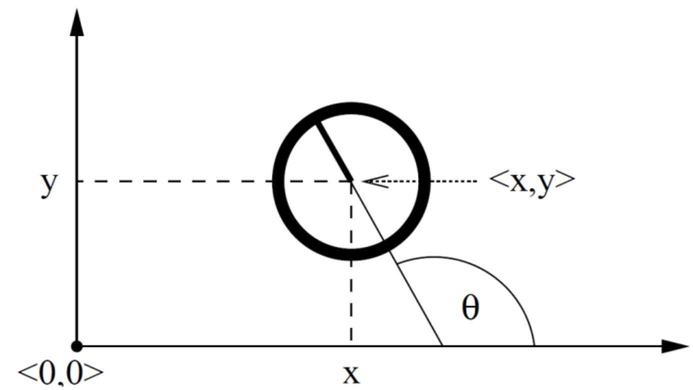


Figure 5.1 Robot pose, shown in a global coordinate system.

- Error parameters: $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$

Algorithm motion_model_velocity(x_t, u_t, x_{t-1}):

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

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

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

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

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

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

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

return $\mathbf{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \mathbf{prob}(\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \mathbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$

Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\mathbf{prob}(a, b^2)$ computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$

Turning radius circle position and radius

Algorithm motion_model_velocity(x_t, u_t, x_{t-1})**:**

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

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

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

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

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

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

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

return $\mathbf{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \mathbf{prob}(\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \mathbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$

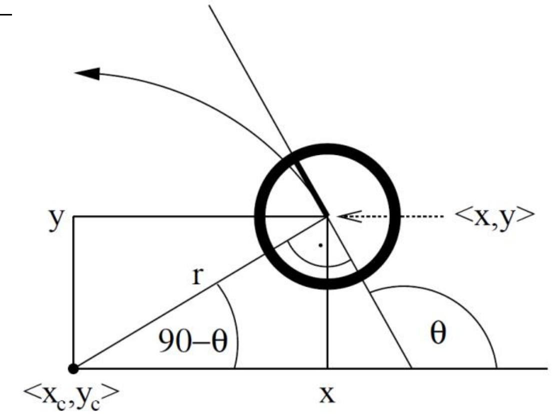
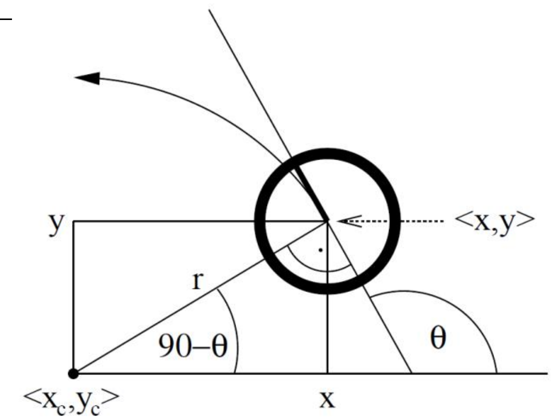


Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\mathbf{prob}(a, b^2)$ computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$



Turning radius circle position and radius

Change of heading

Algorithm motion_model_velocity(x_t, u_t, x_{t-1})**:**

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

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

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

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

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

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

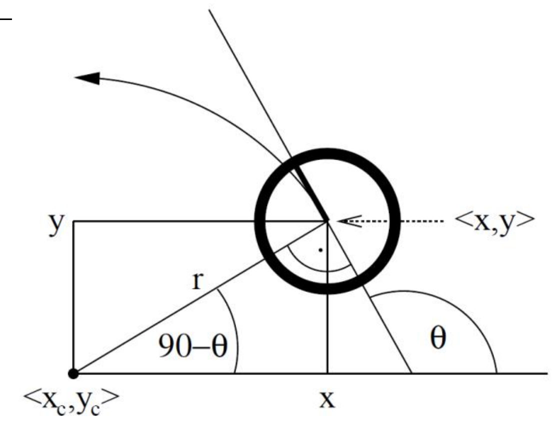
$$\hat{\gamma} = \frac{\theta' - \theta}{\Delta t} - \hat{\omega}$$

return $\mathbf{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \mathbf{prob}(\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \mathbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$

Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\mathbf{prob}(a, b^2)$ computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$



Algorithm motion_model_velocity(x_t, u_t, x_{t-1})**:**

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

Turning radius circle position and radius

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

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

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

Change of heading

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

Translational and rotational velocity, final rotation (error)

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

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

return $\textbf{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \textbf{prob}(\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \textbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$

Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\textbf{prob}(a, b^2)$ computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$

Turning radius circle position and radius

Change of heading

Translational and rotational velocity, final rotation (error)

Probabilities of discrepencies between commanded and computed velocities & rotation, given error parameters times velocity magnitudes

Algorithm motion_model_velocity(x_t, u_t, x_{t-1})**:**

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

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

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

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

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

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

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

return $\mathbf{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \mathbf{prob}(\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \mathbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$

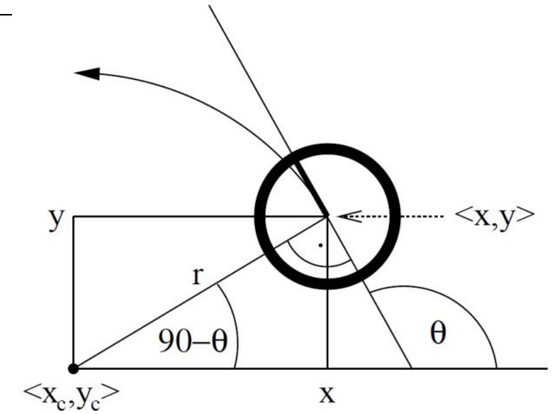
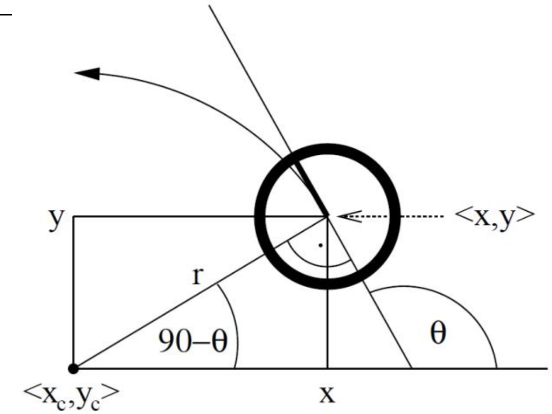


Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\mathbf{prob}(a, b^2)$ computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Motion Model Based on Velocity Command

$x_{t-1} = (x, y, \theta)^T$
$x_t = (x', y', \theta')^T$



Algorithm motion_model_velocity(x_t, u_t, x_{t-1}):

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

Turning radius circle position and radius

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

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

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

Change of heading

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

Translational and rotational velocity, final rotation (error)

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

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

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

Probabilities of discrepencies between commanded and computed velocities & rotation, given error parameters times velocity magnitudes

$$\textit{return}\ \mathbf{prob}(v-\hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2) \cdot \mathbf{prob}(\omega-\hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2) \cdot \mathbf{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$$

But what is **prob** ?

Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function **prob**(a, b^2) computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

Common Probability Distributions

$\mathbf{prob}$ is a zero-mean probability distribution with variance *b.* Examples:

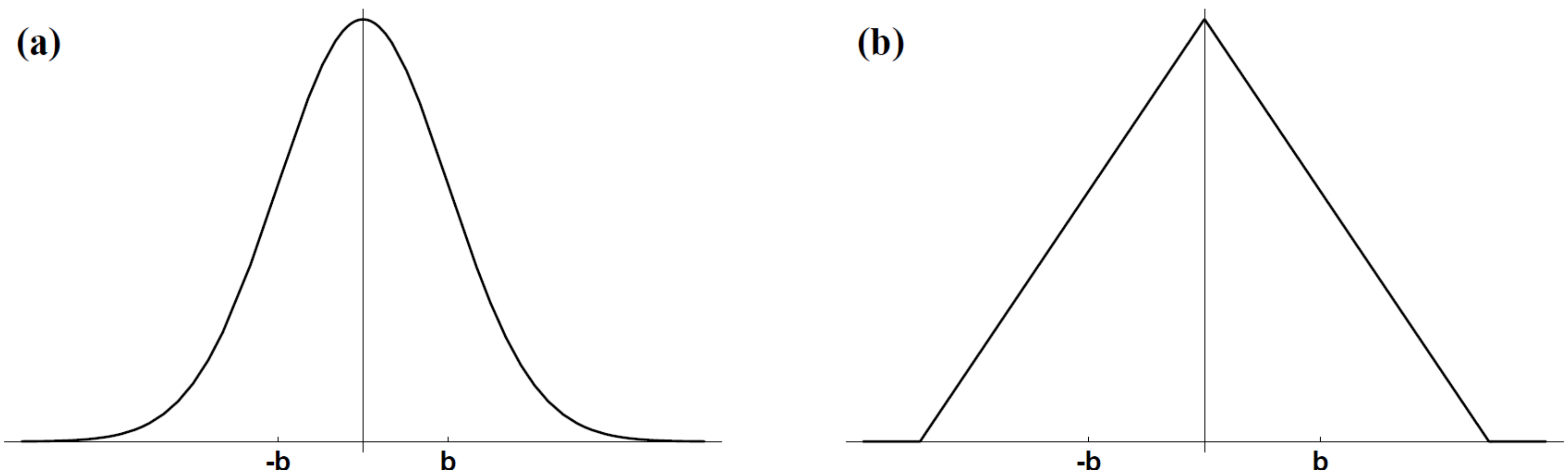


Figure 5.6 Probability density functions with variance b: (a) Normal distribution, (b) triangular distribution.

Computing Prob

1: **Algorithm prob_normal_distribution(a, b^2):**

2: *return* $\frac{1}{\sqrt{2\pi\, b^2}} \exp\left\{-\frac{1}{2}\frac{a^2}{b^2}\right\}$

3: **Algorithm prob_triangular_distribution(a, b^2):**

4: *return* $\max\left\{0, \frac{1}{\sqrt{6}\, b} - \frac{|a|}{6\, b^2}\right\}$

Table 5.2 Algorithms for computing densities of a zero-centered normal distribution and a triangular distribution with variance b^2.

prob_X(a, b^2) is the probability of a occurring for the zero-centered distribution X with variance b^2

Simulating Particles with Error Sampling

Algorithm sample_motion_model_velocity(u_t, x_{t-1}):

$\hat{v} = v + \textbf{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$

$\hat{\omega} = \omega + \textbf{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$

$\hat{\gamma} = \textbf{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$

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

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

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

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

Table 5.3 Algorithm for sampling poses $x_t = (x' \ \ y' \ \ \theta')^T$ from a pose $x_{t-1} = (x \ \ y \ \ \theta)^T$ and a control $u_t = (v \ \ \omega)^T$. Note that we are perturbing the final orientation by an additional random term, $\hat{\gamma}$. The variables α_1 through α_6 are the parameters of the motion noise. The function **sample**(b^2) generates a random sample from a zero-centered distribution with variance b^2. It may, for example, be implemented using the algorithms in Table 5.4.

Sampling with Normal and Triangular Distributions

1: **Algorithm sample_normal_distribution(b^2):**

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

rand(-b,b) in Java: b * (Math.random() + Math.random() - 1)

3: **Algorithm sample_triangular_distribution(b^2):**

4: $$\textit{return}\ \frac{\sqrt{6}}{2} \left[\mathbf{rand}(-b, b) + \mathbf{rand}(-b, b)\right]$$

Java: return (Math.random() + Math.random() - 1) * SCALE;
where final double SCALE = Math.sqrt(6 * variance);

Table 5.4 Algorithm for sampling from (approximate) normal and triangular distributions with zero mean and variance b^2; see Winkler (1995: p293). The function $\mathbf{rand}(x, y)$ is assumed to be a pseudo random number generator with uniform distribution in $[x, y]$.

See also: https://stackoverflow.com/questions/33220176/triangular-distribution-in-java

In General

- Collect a lot of motion data and use that data to create your motion model.

Measurement Model

$$p(z_t \mid x_t, m)$$

- z_t – sensor inputs at time t
- x_t – pose (state) at time t
- m – map of the environment
- $p(z_t \mid x_t, m)$ – probability of measuring z_t given pose x_t and map m

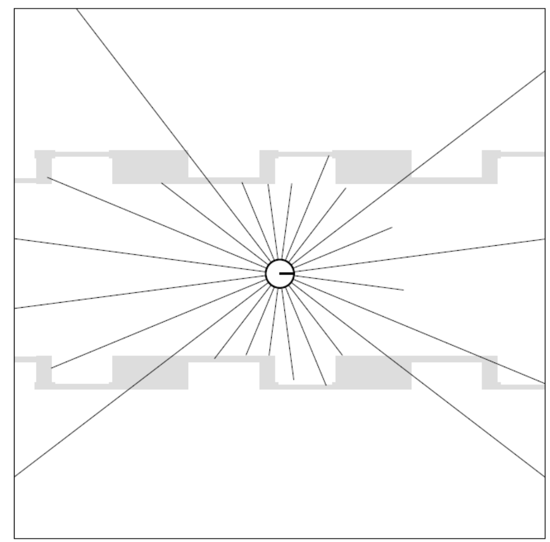


Figure 6.1 Typical ultrasound scan of a robot in its environment.

Distributions for Modeling Different Kinds of Error Expectations

Expected variation around true object range value

Unexpected closer objects

Object missed by range finder and max range returned

General unexplained sensor noise

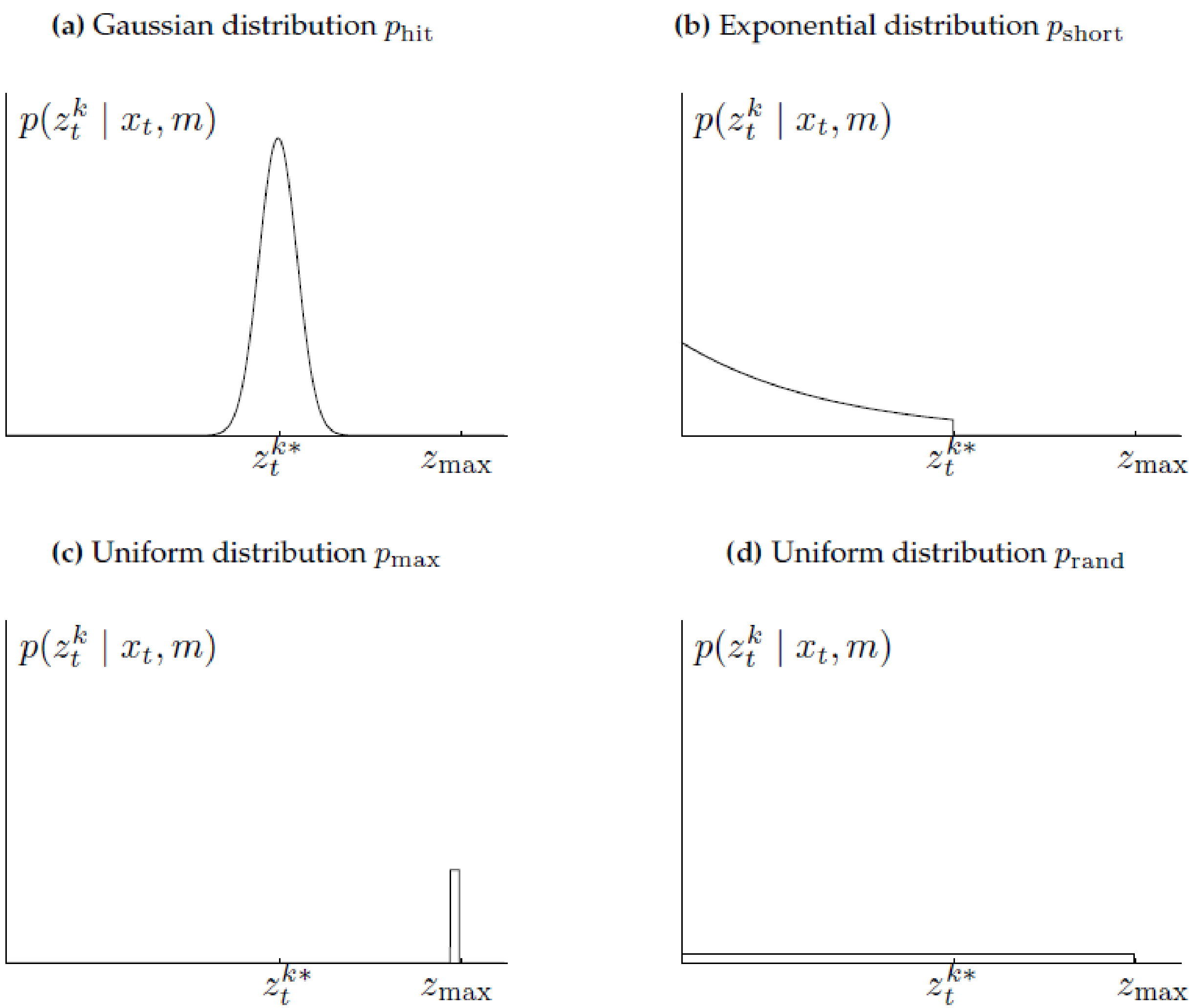


Figure 6.3 Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

Added together…

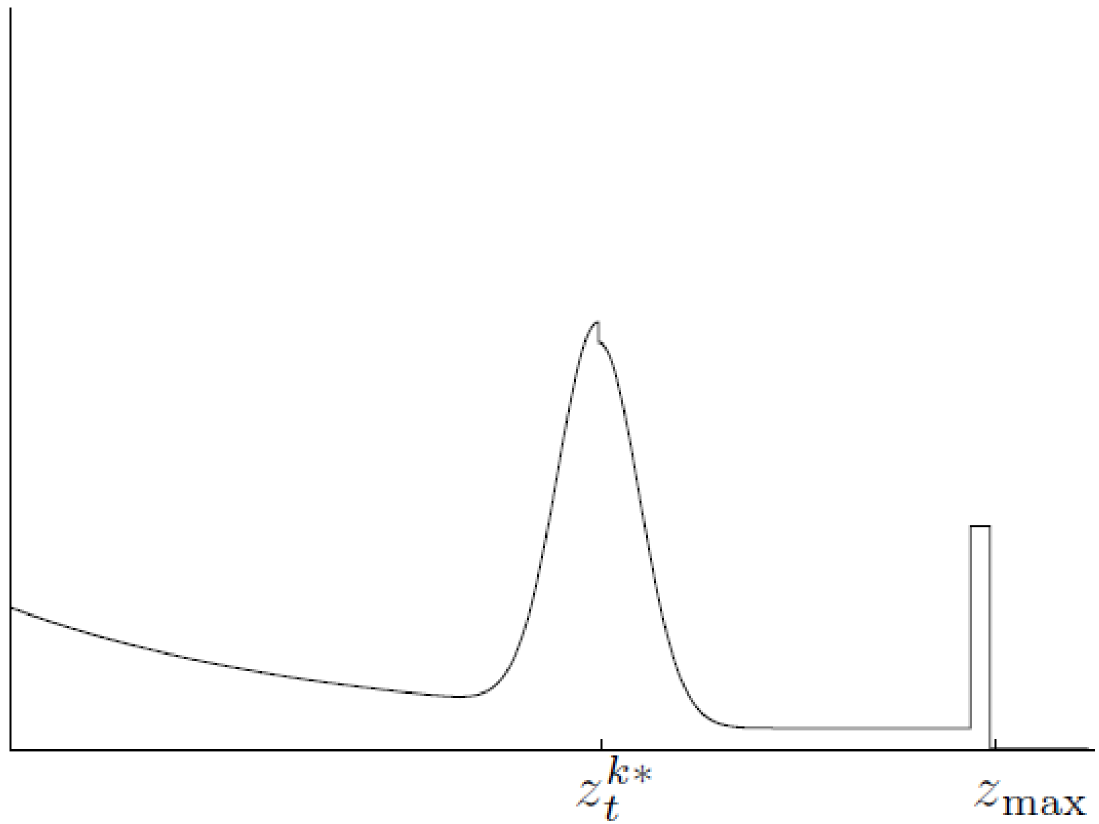


Figure 6.4 "Pseudo-density" of a typical mixture distribution $p(z_t^k \mid x_t, m)$.

… To Make a Weighted Average …

These four different distributions are now mixed by a weighted average, defined by the parameters z_{hit}, z_{short}, $z_{\max}$, and z_{rand} with $z_{\text{hit}} + z_{\text{short}} + z_{\max} + z_{\text{rand}} = 1$.

$$p(z_t^k \mid x_t, m) \;=\; \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix} \tag{6.12}$$

Algorithm beam_range_finder_model(z_t, x_t, m):

$q = 1$

for $k = 1$ *to* K *do*

  compute z_t^{k*} *for the measurement* z_t^k *using ray casting*

  $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$

    $+ z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$

  $q = q \cdot p$

return q

Table 6.1 Algorithm for computing the likelihood of a range scan z_t, assuming conditional independence between the individual range measurements in the scan.

… But How Should We Set the Weights?

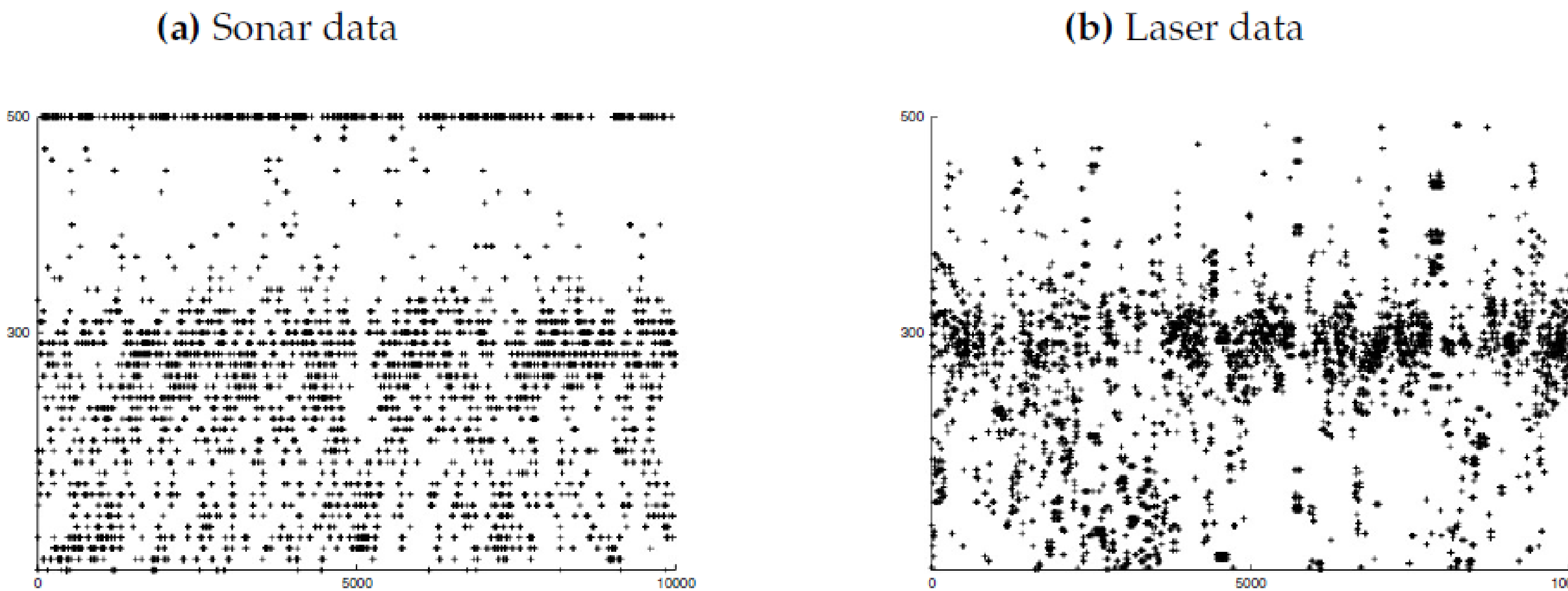


Figure 6.5 Typical data obtained with (a) a sonar sensor and (b) a laser-range sensor in an office environment for a "true" range of 300 cm and a maximum range of 500 cm.

Maximum Likelihood Estimation

- Iteratively tune parameters until the data distribution collected for known distance measurements achieves maximum likelihood.
- (Or manually tune for good match to data.)

Algorithm learn_intrinsic_parameters(Z, X, m):

repeat until convergence criterion satisfied
  for all z_i in Z do
    $\eta = [\, p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m)$
      $+ p_{\max}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m)\,]^{-1}$
    calculate z_i^*
    $e_{i,\text{hit}} = \eta\; p_{\text{hit}}(z_i \mid x_i, m)$
    $e_{i,\text{short}} = \eta\; p_{\text{short}}(z_i \mid x_i, m)$
    $e_{i,\max} = \eta\; p_{\max}(z_i \mid x_i, m)$
    $e_{i,\text{rand}} = \eta\; p_{\text{rand}}(z_i \mid x_i, m)$

  $z_{\text{hit}} = |Z|^{-1} \sum_i e_{i,\text{hit}}$
  $z_{\text{short}} = |Z|^{-1} \sum_i e_{i,\text{short}}$
  $z_{\max} = |Z|^{-1} \sum_i e_{i,\max}$
  $z_{\text{rand}} = |Z|^{-1} \sum_i e_{i,\text{rand}}$
  $\sigma_{\text{hit}} = \sqrt{\frac{1}{\sum_i e_{i,\text{hit}}} \sum_i e_{i,\text{hit}} (z_i - z_i^*)^2}$
  $\lambda_{\text{short}} = \frac{\sum_i e_{i,\text{short}}}{\sum_i e_{i,\text{short}}\, z_i}$
return $\Theta = \{z_{\text{hit}}, z_{\text{short}}, z_{\max}, z_{\text{rand}}, \sigma_{\text{hit}}, \lambda_{\text{short}}\}$

Table 6.2 Algorithm for learning the intrinsic parameters of the beam-based sensor model from data.

Algorithm learn_intrinsic_parameters(Z, X, m):

repeat until convergence criterion satisfied

for all z_i in Z do

$$\eta = [\, p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m) + p_{\text{max}}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m)\,]^{-1}$$

calculate z_i^*

$$e_{i,\text{hit}} = \eta\, p_{\text{hit}}(z_i \mid x_i, m)$$

$$e_{i,\text{short}} = \eta\, p_{\text{short}}(z_i \mid x_i, m)$$

$$e_{i,\text{max}} = \eta\, p_{\text{max}}(z_i \mid x_i, m)$$

$$e_{i,\text{rand}} = \eta\, p_{\text{rand}}(z_i \mid x_i, m)$$

$$z_{\text{hit}} = |Z|^{-1} \sum_i e_{i,\text{hit}}$$

$$z_{\text{short}} = |Z|^{-1} \sum_i e_{i,\text{short}}$$

$$z_{\text{max}} = |Z|^{-1} \sum_i e_{i,\text{max}}$$

$$z_{\text{rand}} = |Z|^{-1} \sum_i e_{i,\text{rand}}$$

$$\sigma_{\text{hit}} = \sqrt{\frac{1}{\sum_i e_{i,\text{hit}}} \sum_i e_{i,\text{hit}} (z_i - z_i^*)^2}$$

$$\lambda_{\text{short}} = \frac{\sum_i e_{i,\text{short}}}{\sum_i e_{i,\text{short}}\, z_i}$$

return $\Theta = \{z_{\text{hit}}, z_{\text{short}}, z_{\text{max}}, z_{\text{rand}}, \sigma_{\text{hit}}, \lambda_{\text{short}}\}$

Compute the relative likelihood of each measurement interpretation for each measurement.

Think of these as normalized weights for the interpretation of each measurement.

Set the new weights for the weighted sum according to their average relative likelihood

Also compute the most likely parameters for our distributions.

In General

- Collect a lot of measurement data and use that data to create your measurement model.

Project Goal: Monte Carlo Localization for the Kidnapped Robot Problem

- Goals:
  - Acquire a map of the environment, e.g. FASTSLAM or other appropriate techniques.
  - Implement Monte Carlo localization.
  - Solve the Kidnapped Robot Problem:
    - An autonomous robot is transported to an unknown state and must localize.
  - Optional:
    - Designate a robot home state.
    - Have the robot return home after being kidnapped and successfully localizing.

Project Tips

- Set simple goals. Follow the KISS Principle. (You can always set more ambitious goals if you achieve these early.) Example:
  - 1D state space:
    - Have a fixed robot that can rotate range finder(s), a camera, or other localizing sensor to determine state θ.
    - When you start the system, let initial state θ_0 be the "home state".
    - Have the system rotate and sense to build a mapping from sensor inputs to probable locations.
    - After it has terminated mapping, put it in a new mode seeking to return home when it does not appear to be home.
    - "Kidnap" it by rotating the robot and demonstrate that it can relocalize and return home.
- Divide labor: project lead, documentation, version control, sensor model, motor model, etc.
- Plan team meeting times in advance. Budget for 18 total hours for each over 2 weeks beyond class. Log hours.

Project Platform: Anki Cozmo

- Programming tools:
  - Cozmo SDK: https://www.anki.com/en-us/cozmo/SDK
  - cozmo-tools: https://github.com/touretzkyds/cozmo-tools
- Possible projects:
  1. Create a new project where the robot is restricted to rotational movement only and uses visual camera sensing to localize.
  2. Find and build upon prior Cozmo 2D localization and mapping work you might find.