Motion Models (cont)

Computing the Density

- to compute

$$\text{prob}(v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2),$$
$$\text{prob}(\omega - \widehat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2), \text{ and}$$
$$\text{prob}(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2)$$

use the appropriate probability density function; i.e., for zero-mean Gaussian noise:

$$\text{prob}(a, b^2) = \frac{1}{\sqrt{2\pi b^2}} e^{-\frac{1a^2}{2b^2}}$$

Sampling from the Velocity Motion Model

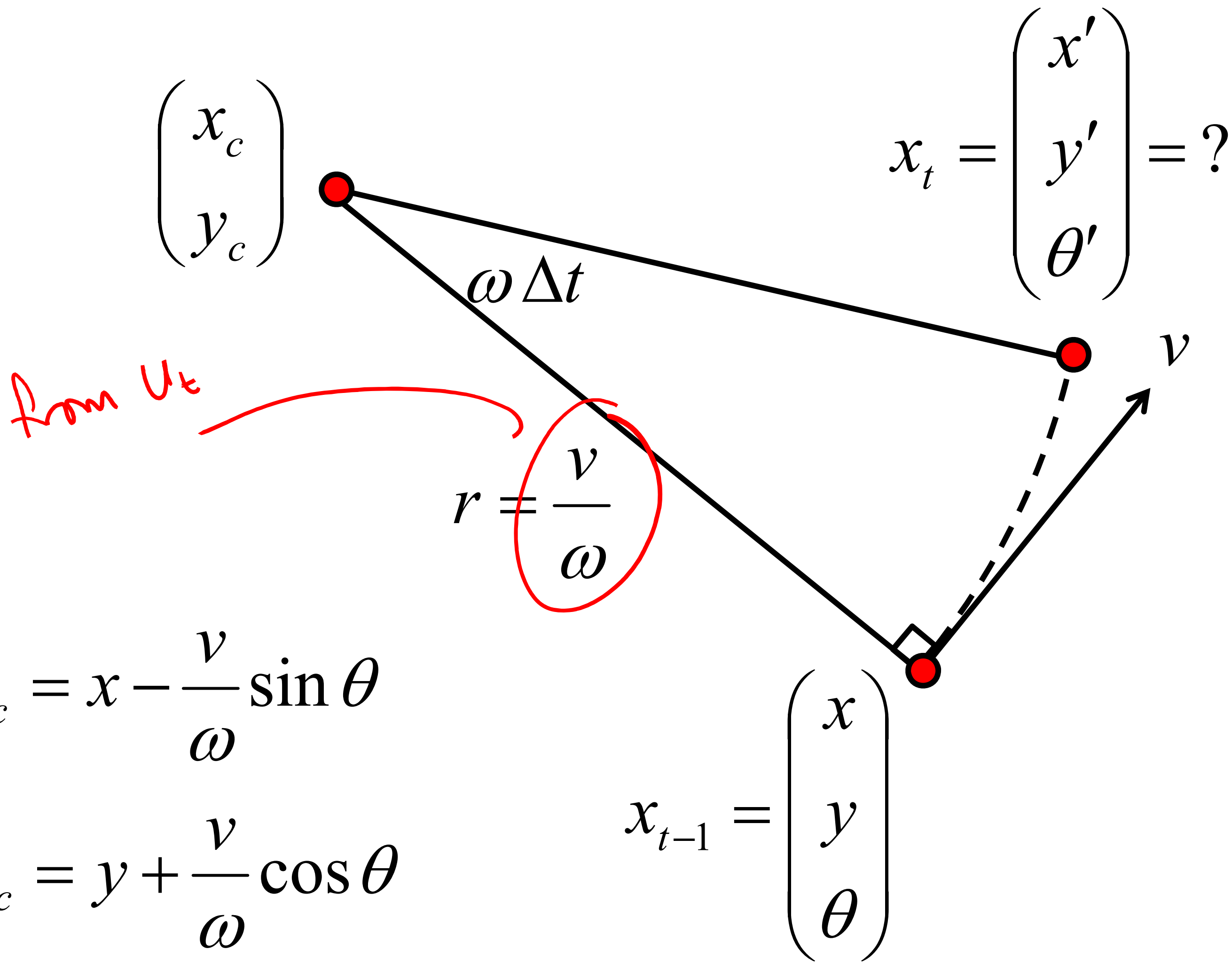
- suppose that a robot has a map of its environment and it needs to find its pose in the environment
  - this is the robot localization problem
  - several variants of the problem
    - the robot knows where it is initially
    - the robot does not know where it is initially
    - kidnapped robot: at any time, the robot can be teleported to another location in the environment
- a popular solution to the localization problem is the particle filter
  - uses simulation to sample the state density $p(x_t \mid u_t, x_{t-1})$

Sampling from the Velocity Motion Model

- sampling the conditional density is easier than computing the density because we only require the forward kinematics model
  - given the control u_t and the previous pose $x_{t\text{-}1}$ find the new pose x_t

random

Sampling from the Velocity Motion Model



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

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

Eqs 5.7, 5.8

Sampling from the Velocity Motion Model

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix}$$

$$= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t \end{pmatrix} \quad \text{Eqs 5.9}$$

*we already derived this for the differential drive!

Sampling from the Velocity Motion Model

- as with the original motion model, we will assume that given noisy velocities the robot can also make a small rotation in place to determine the final orientation of the robot

noisy velocities

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t + \hat{\gamma}\Delta t \end{pmatrix}$$

Sampling from the Velocity Motion Model

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

assumes zero-mean ~~additive~~ noise

– small in-place rotation of x_t

forward kinematics using noisy velocities

Sampling from the Velocity Motion Model

- the function $\mathrm{sample}(b^2)$ generates a random sample from a zero-mean distribution with variance b^2
- Matlab is able to generate random numbers from many different distributions
  - help randn
  - help stats

How to Sample from Normal or Triangular Distributions?

- Sampling from a normal distribution

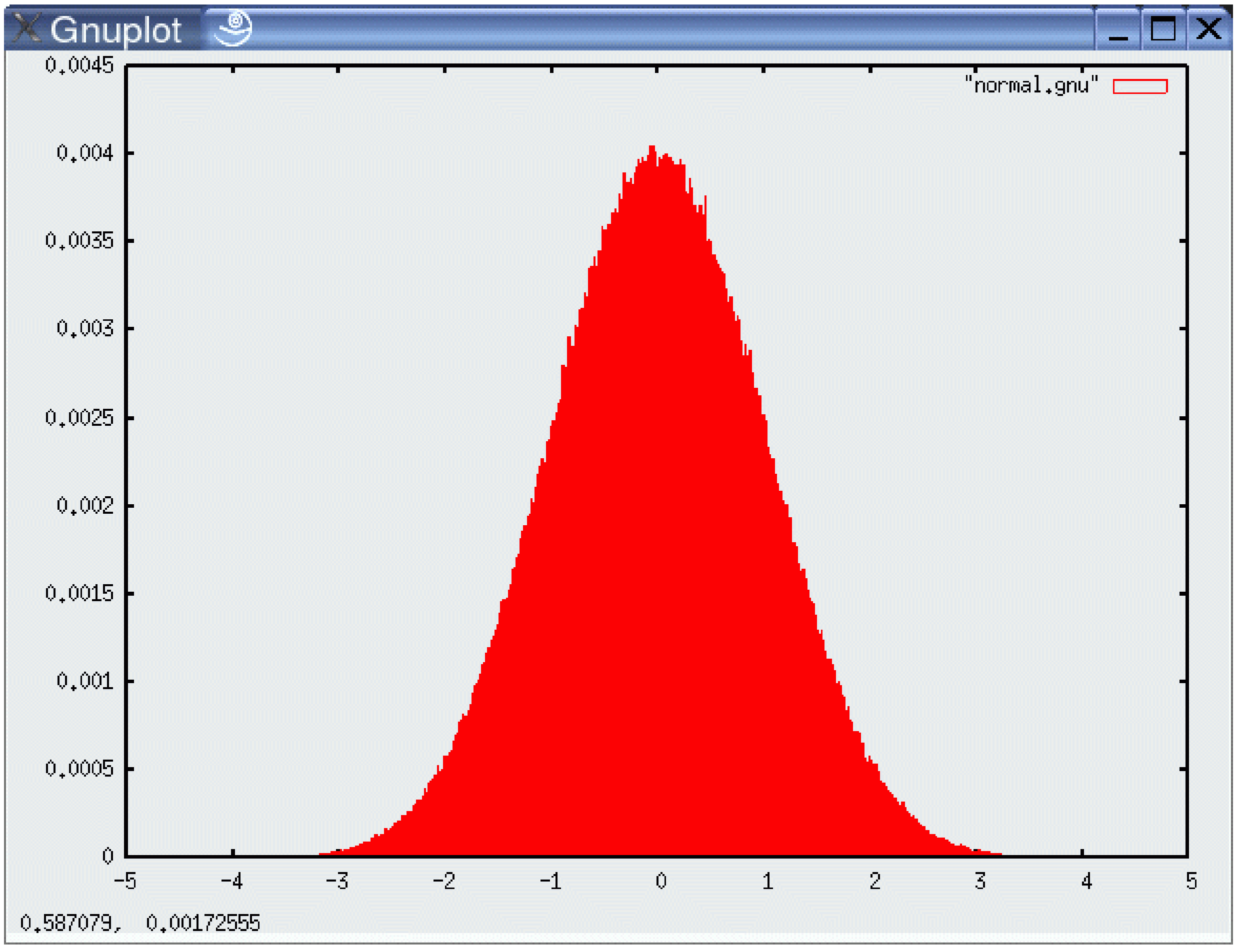
1. Algorithm **sample_normal_distribution**(b):
2. return $$\frac{1}{2}\sum_{i=1}^{12} rand(-b, b)$$

samples from uniform distribution

- Sampling from a triangular distribution

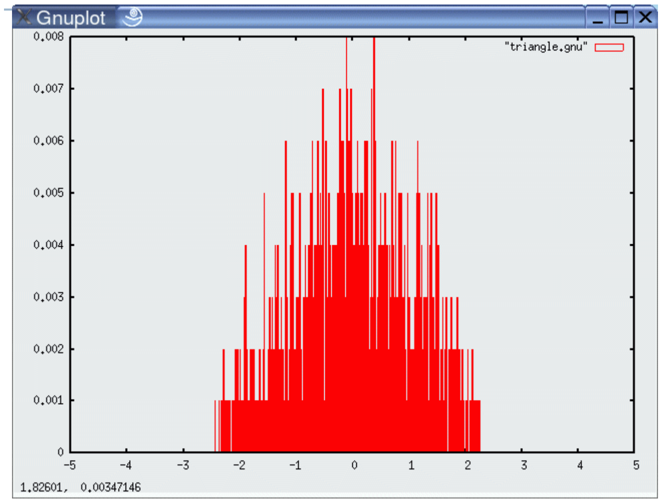
1. Algorithm **sample_triangular_distribution**(b):
2. return $$\frac{\sqrt{6}}{2}\left[\mathbf{rand}(-b, b) + \mathbf{rand}(-b, b)\right]$$

Normally Distributed Samples

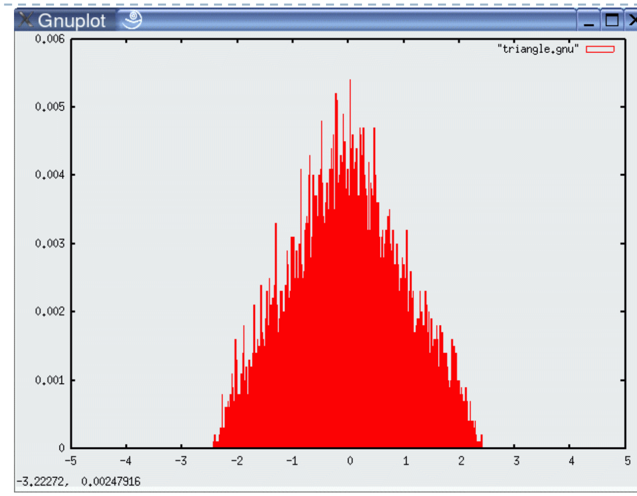


10^6 samples

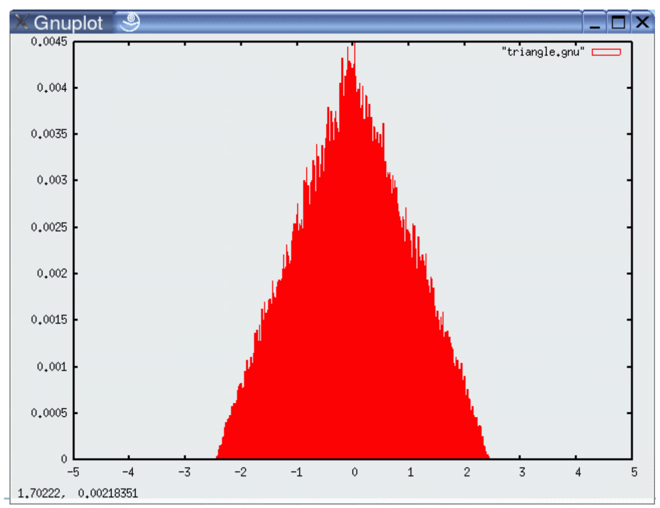
For Triangular Distribution



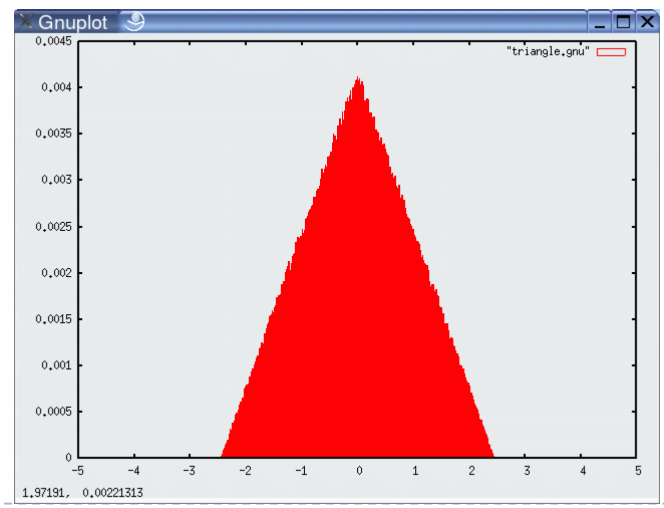
10^3 samples



10^4 samples



10^5 samples



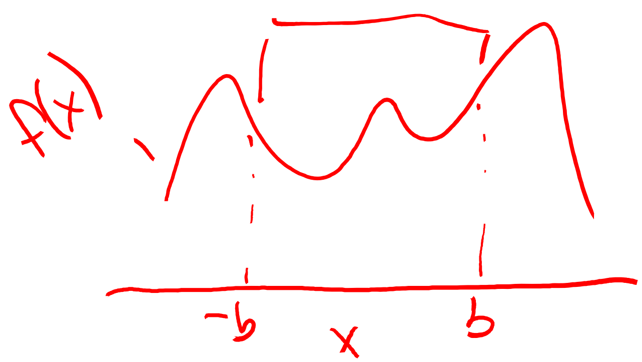
10^6 samples

Rejection Sampling

- Sampling from arbitrary distributions

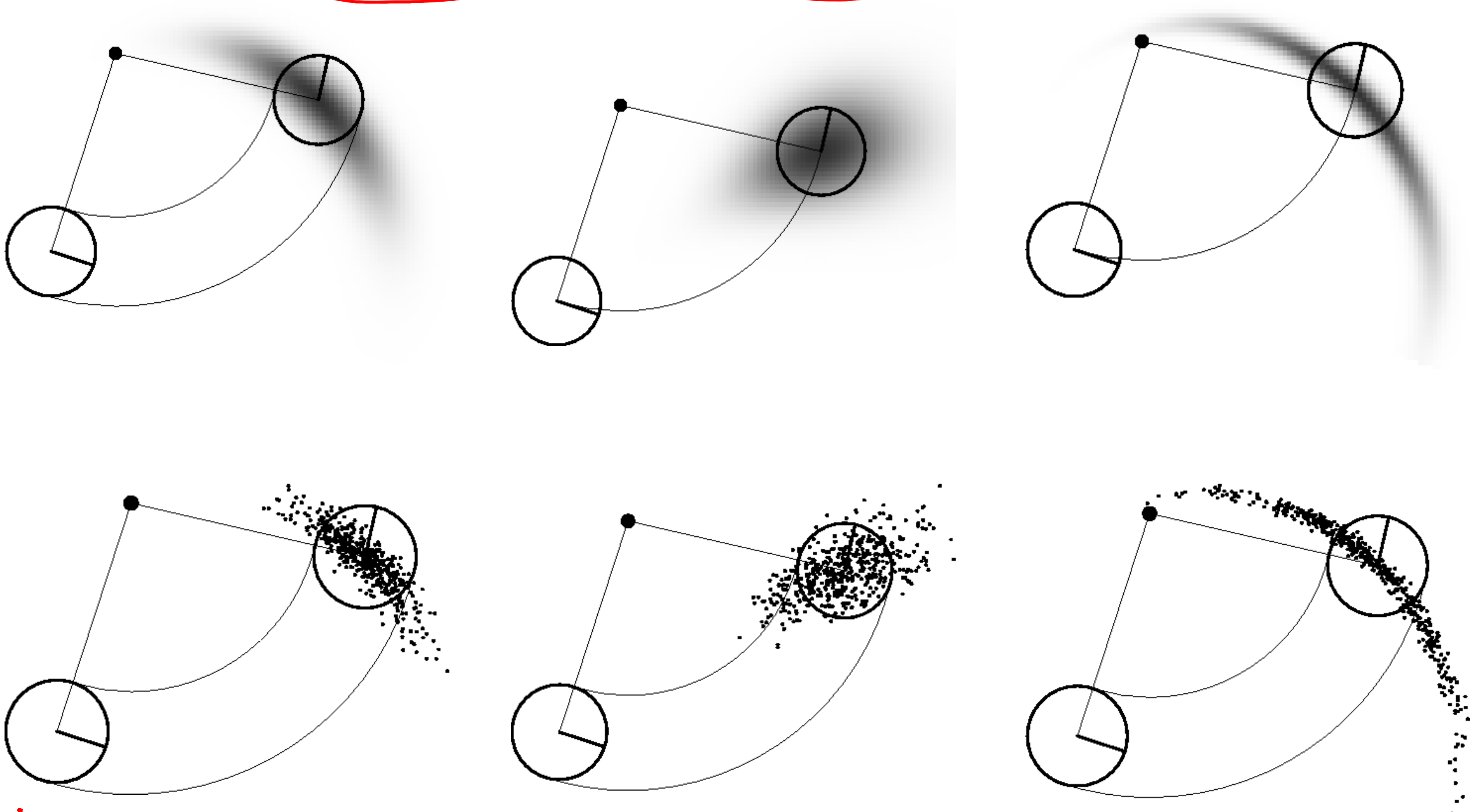
```
Algorithm sample_distribution(f,b):
repeat
        x = rand(-b, b)
        y = rand(0, max{f(x) | x ∈ (-b, b)})
until ( y ≤ f(x) )
return x
```

density you want to sample from



Examples

velocity motion model



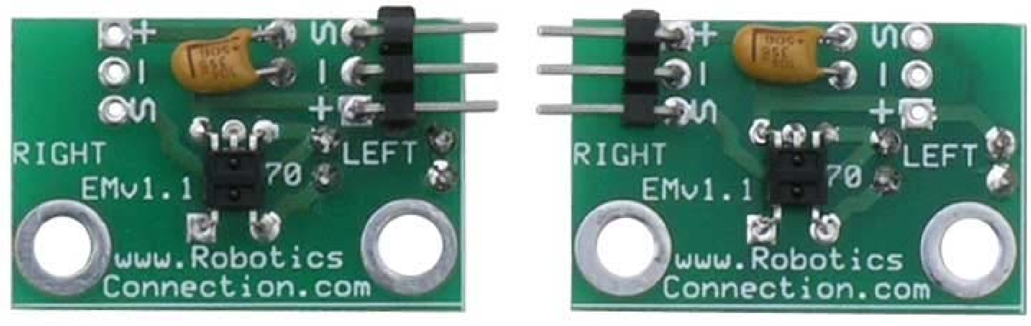
sampling velocity motion model

Odometry Motion Model

- many robots make use of odometry rather than velocity
- odometry uses a sensor or sensors to measure motion to estimate changes in position over time
- typically more accurate than velocity motion model, but measurements are available only after the motion has been completed
- technically a measurement rather than a control
  - but usually treated as control to simplify the modeling
- odometry allows a robot to estimate its pose
  - but no fixed mapping from odometer coordinates and world coordinates
- in wheeled robots the sensor is often a rotary encoder

Example Wheel Encoders

These modules require +5V and GND to power them, and provide a 0 to 5V output. They provide +5V output when they "see" white, and a 0V output when they "see" black.

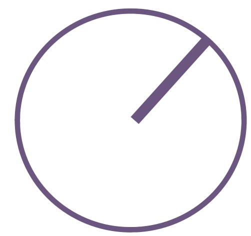


These disks are manufactured out of high quality laminated color plastic to offer a very crisp black to white transition. This enables a wheel encoder sensor to easily see the transitions.

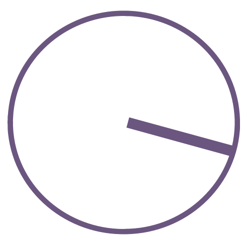
Source: http://www.active-robots.com/

Odometry Model

- when using odometry, the robot keeps an internal estimate of its pose at all time
  - for example, consider a robot moving from pose $\bar{x}_{t-1}$ to $\bar{x}_t$



$$\bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$



$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix}$$

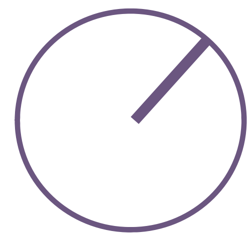
Note: bar indicates values in the robot's internal coordinate system

Odometry Model

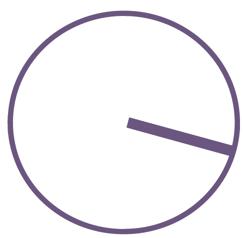
- the internal pose estimates $\bar{x}_{t-1}$ to $\bar{x}_t$ are treated as the control inputs to the robot:

$$u_t = \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix}$$

~ odometry information treated as control vector



$$\bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$

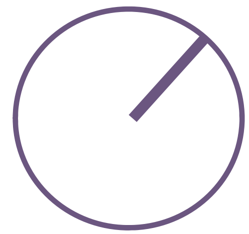


$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix}$$

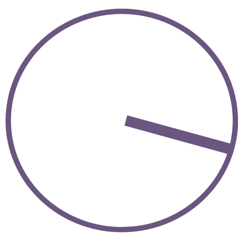
Note: bar indicates values in the robot's internal coordinate system

Odometry Model

- we require a model of how the robot moves from $\bar{x}_{t-1}$ to $\bar{x}_t$
  - there are an infinite number of possible motions between $\bar{x}_{t-1}$ to $\bar{x}_t$



$$\bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$

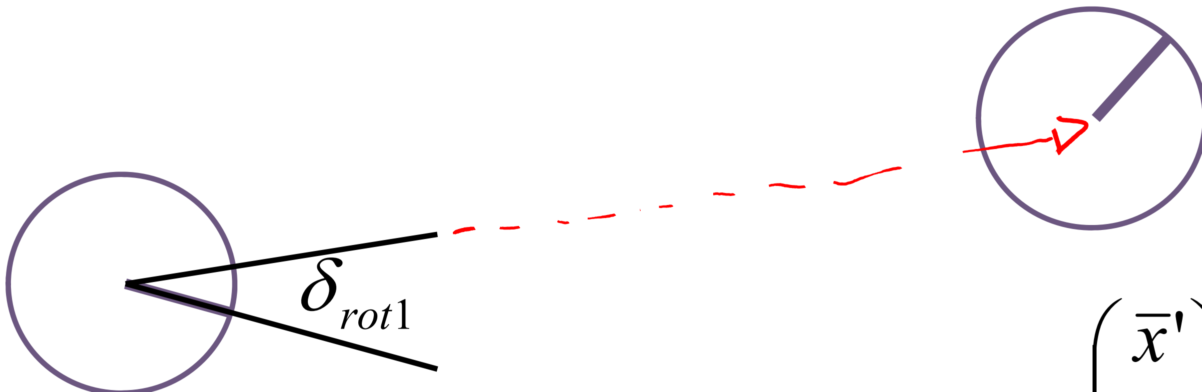


$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix}$$

Note: bar indicates values in the robot's internal coordinate system

Odometry Model

- assume the motion is accomplished in 3 steps:
  1. rotate in place by δ_{rot1}

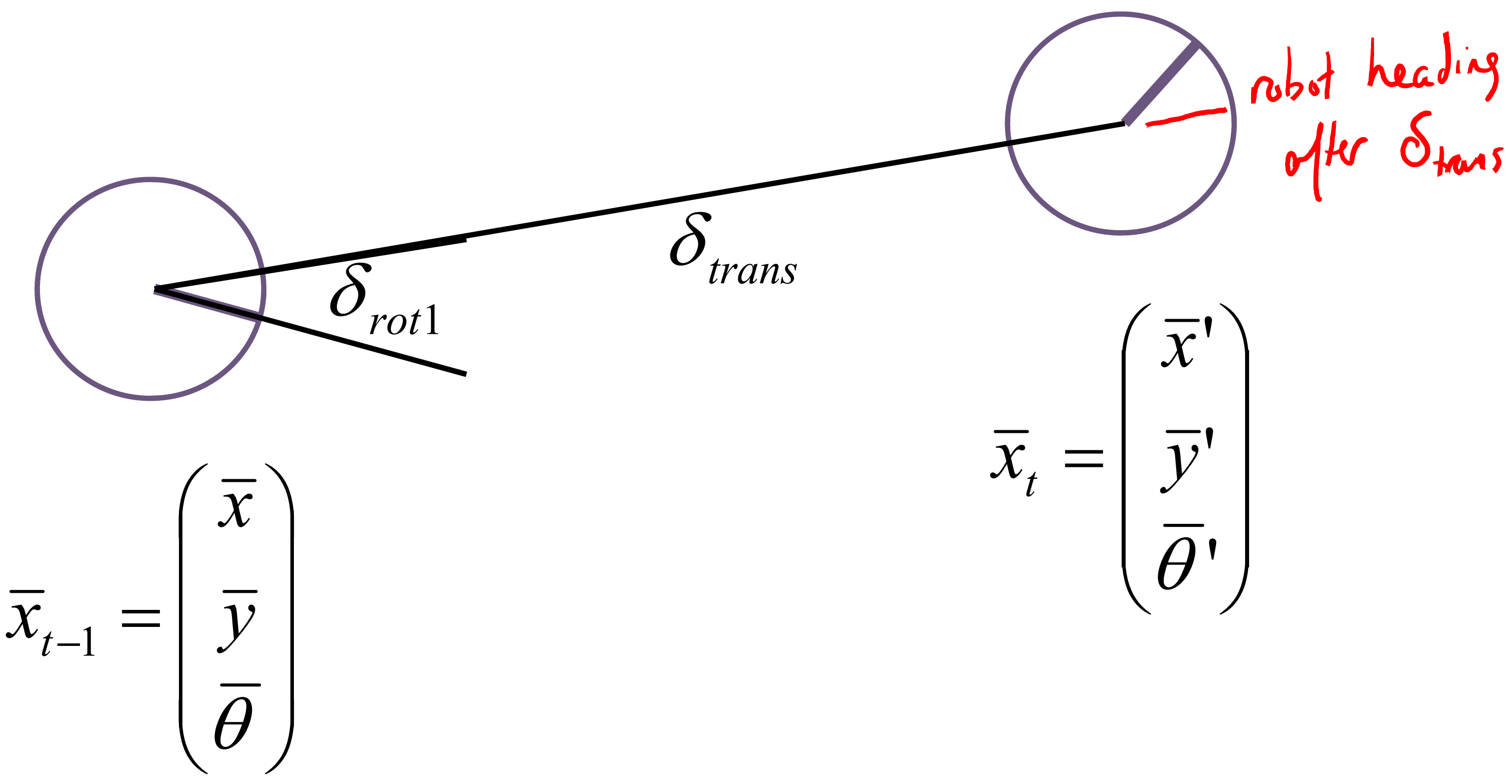


$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix} \qquad \bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$

Note: bar indicates values in the robot's internal coordinate system

Odometry Model

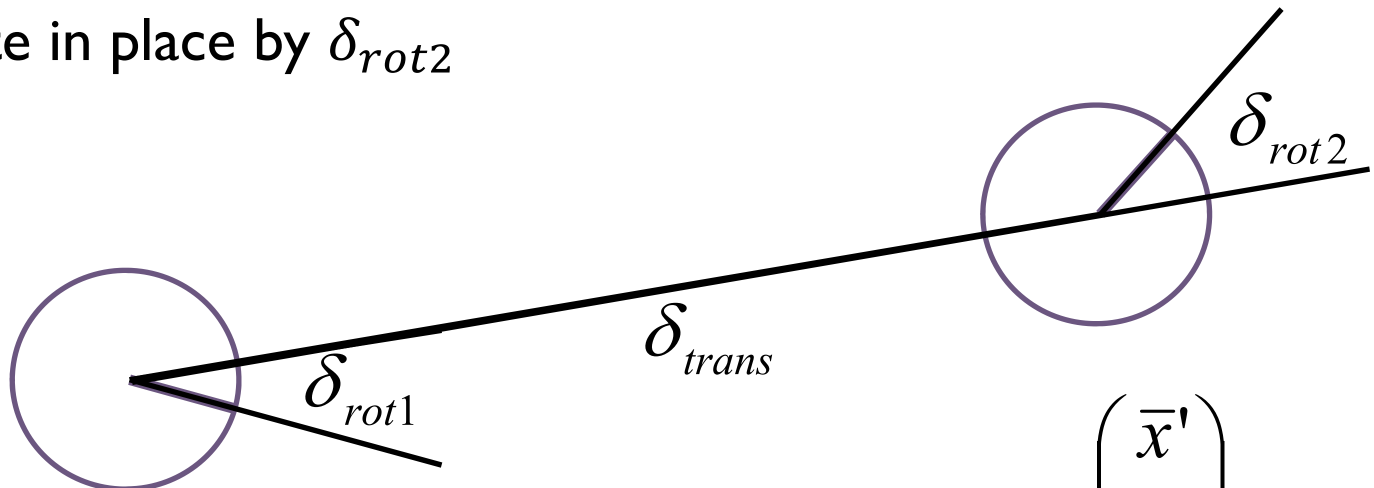
- assume the motion is accomplished in 3 steps:
  1. rotate in place by δ_{rot1}
  2. move in a straight line by δ_{trans}



Note: bar indicates values in the robot's internal coordinate system

Odometry Model

- assume the motion is accomplished in 3 steps:
  1. rotate in place by δ_{rot1}
  2. move in a straight line by δ_{trans}
  3. rotate in place by δ_{rot2}



$$\bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$

$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix}$$

Note: bar indicates values in the robot's internal coordinate system

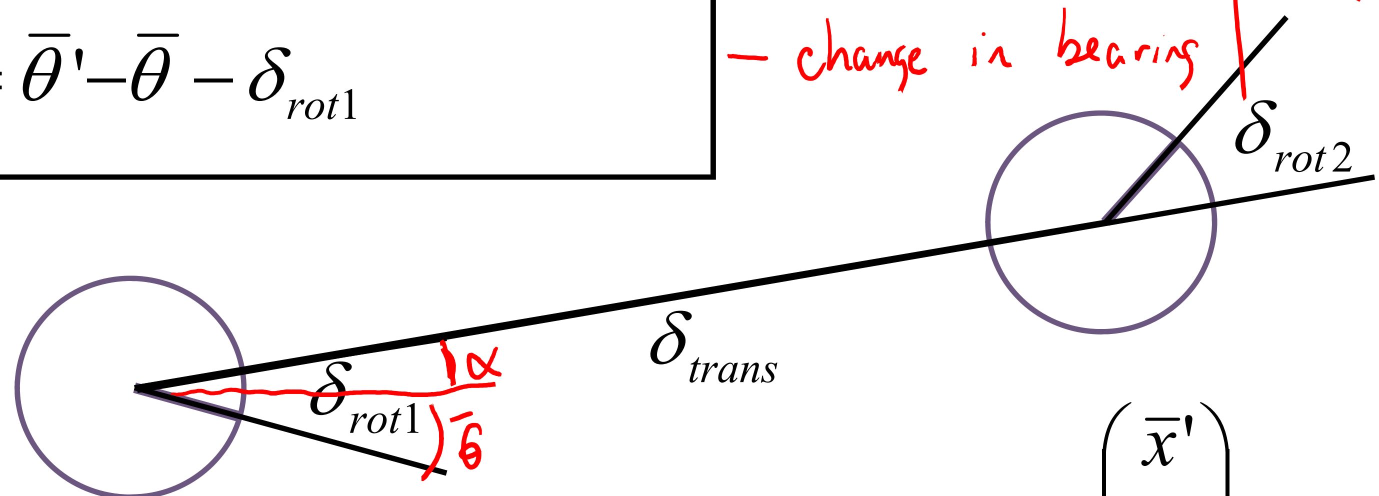
Odometry Model

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

$$\delta_{rot1} = \operatorname{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$ — change in bearing

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$ — change in bearing

do not depend on coordinate system



$$\bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$

$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix}$$

Note: bar indicates values in the robot's internal coordinate system

Noise Model for Odometry

- the difference between the true motion of the robot and the odometry motion is assumed to be a zero-mean random value

from odometry — true value — zero-mean random value

$$\delta_{rot1} - \hat{\delta}_{rot1} = \varepsilon_{\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2}$$

$$\delta_{trans} - \hat{\delta}_{trans} = \varepsilon_{\alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot1}^2 + \delta_{rot2}^2)}$$

$$\delta_{rot2} - \hat{\delta}_{rot2} = \varepsilon_{\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2}$$

Sampling from the Odometry Motion Model

- suppose you are given the previous pose of the robot in world coordinates (x_{t-1}) and the most recent odometry from the robot (u_t)
- how do you generate a random sample of the current pose of the robot in world coordinates (x_t)?
  1. use odometry to compute motion parameters $\delta_{rot1}, \delta_{trans}, \delta_{rot2}$
  2. use noise model to generate random true motion parameters $\hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2}$
  3. use random true motion parameters to compute a random x_t

$$u_t = \begin{bmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{bmatrix}$$

Sample Odometry Motion Model

1. Algorithm **sample_motion_model**(u_t, x_{t-1}):

pose & orientation in world coordinates

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

motion parameters from odometry (coordinate frame independent)*

5. $\hat{\delta}_{rot1} = \delta_{rot1} - sample(\alpha_1 \, \delta_{rot1}^2 + \alpha_2 \, \delta_{trans}^2)$
6. $\hat{\delta}_{trans} = \delta_{trans} - sample(\alpha_3 \, \delta_{trans}^2 + \alpha_4 \, (\delta_{rot1}^2 + \delta_{rot2}^2))$
7. $\hat{\delta}_{rot2} = \delta_{rot2} - sample(\alpha_1 \, \delta_{rot2}^2 + \alpha_2 \, \delta_{trans}^2)$

generate noisy motion parameters

8. $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
9. $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
10. $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

use noisy motion parameters to compute forward kinematics

11. return $[x' \quad y' \quad \theta']^T$

possible new location in world coordinates

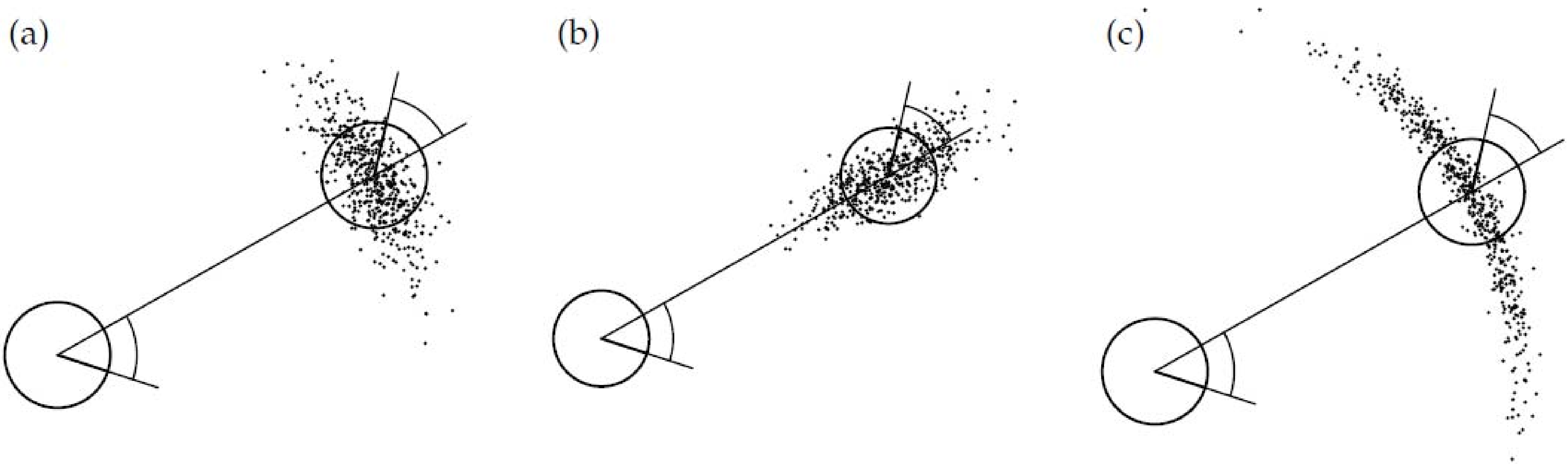
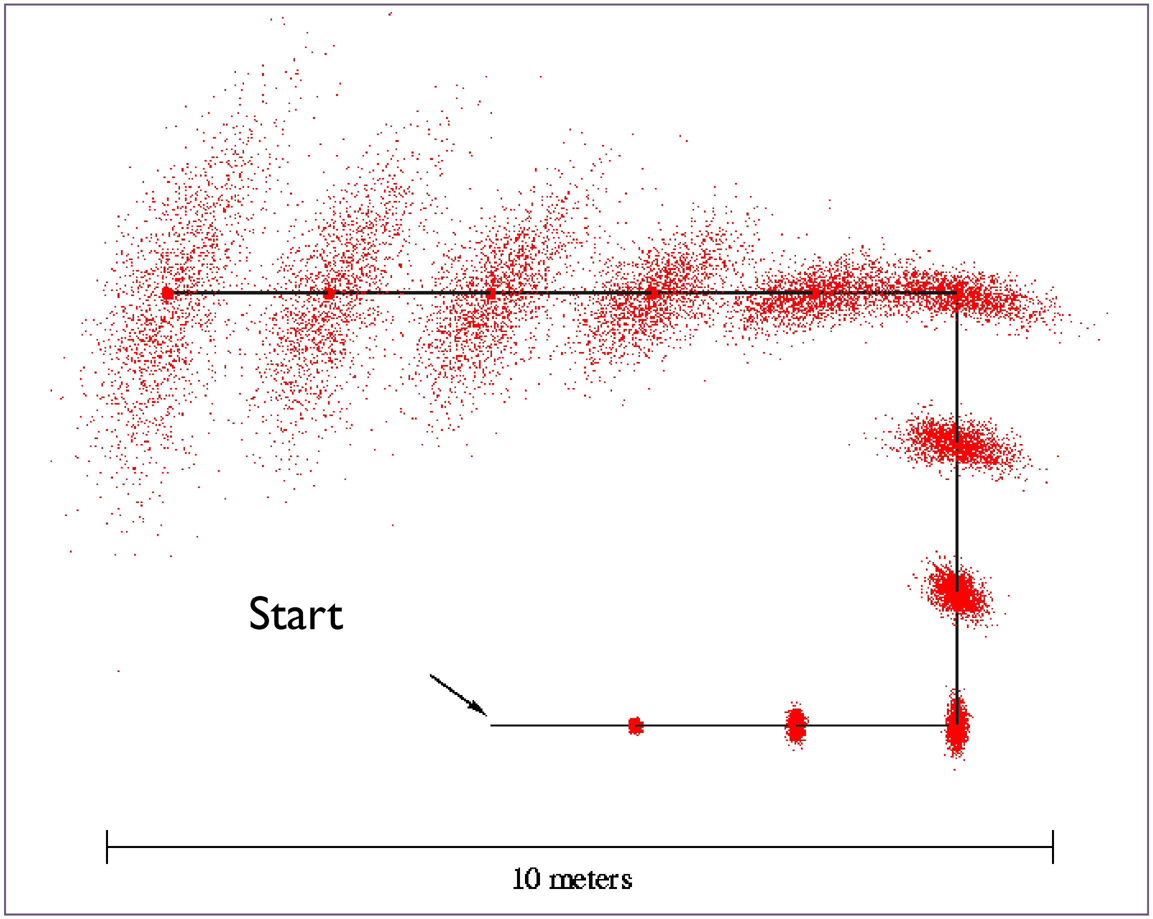


Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

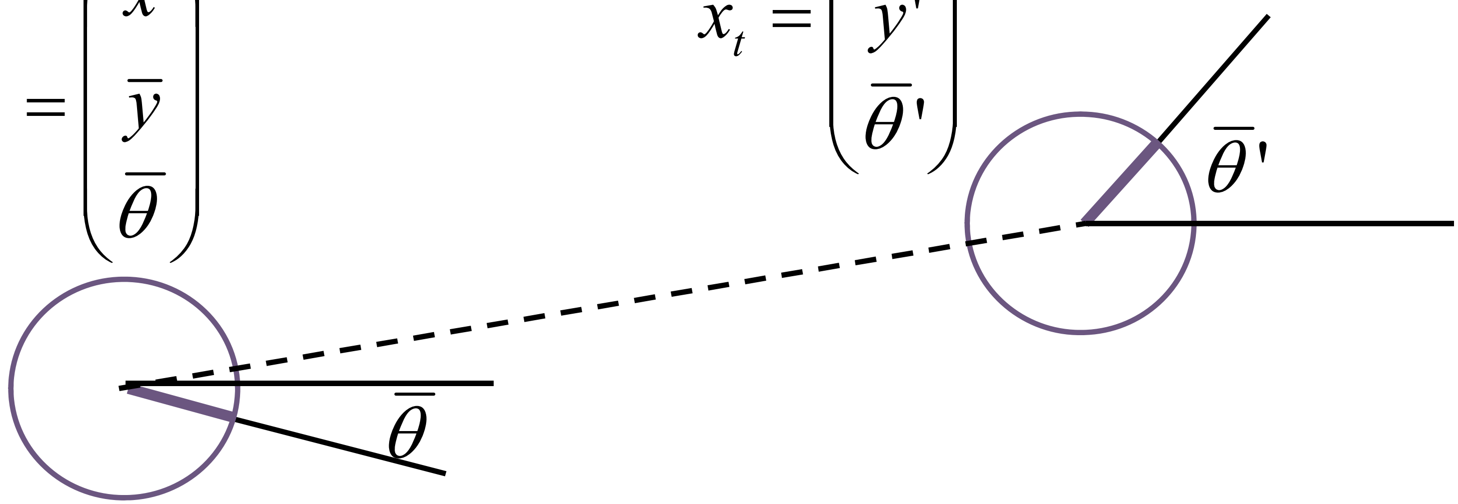
Sampling from Our Motion Model



Odometry Motion Model

- the key to computing $p(x_t \mid u_t, x_{t-1})$ for the odometry motion model is to remember that the robot has an internal estimate of its pose

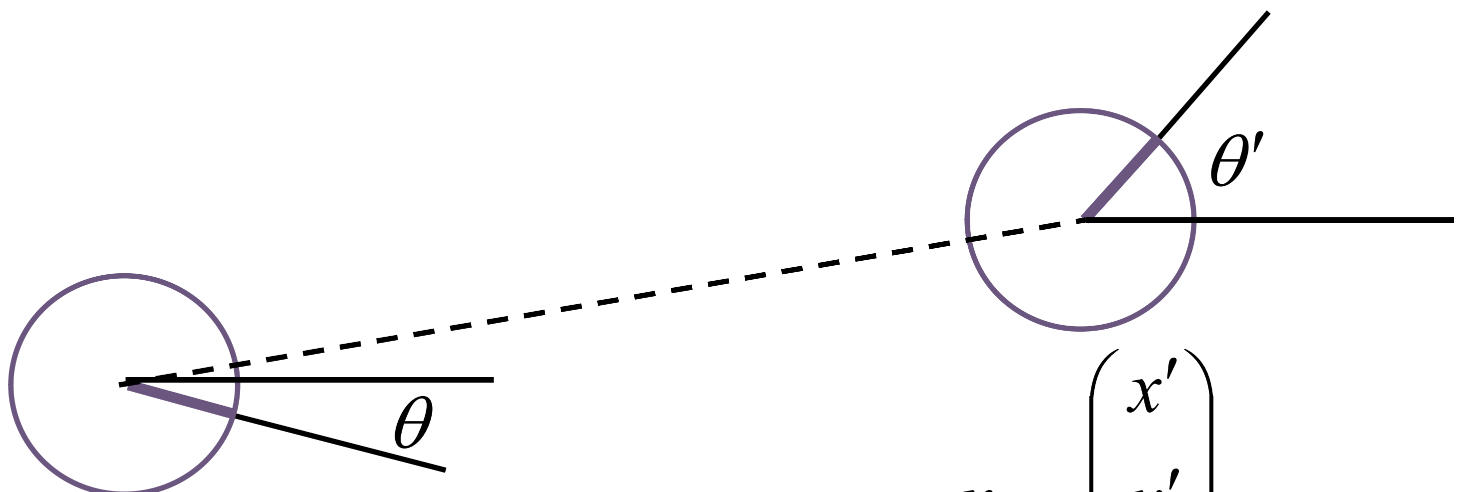
$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix} \qquad \bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$



robot's internal poses

Odometry Motion Model

- the key to computing $p(x_t \mid u_t, x_{t-1})$ for the odometry motion model is to remember that the robot has an internal estimate of its pose



$$x_t = \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}$$

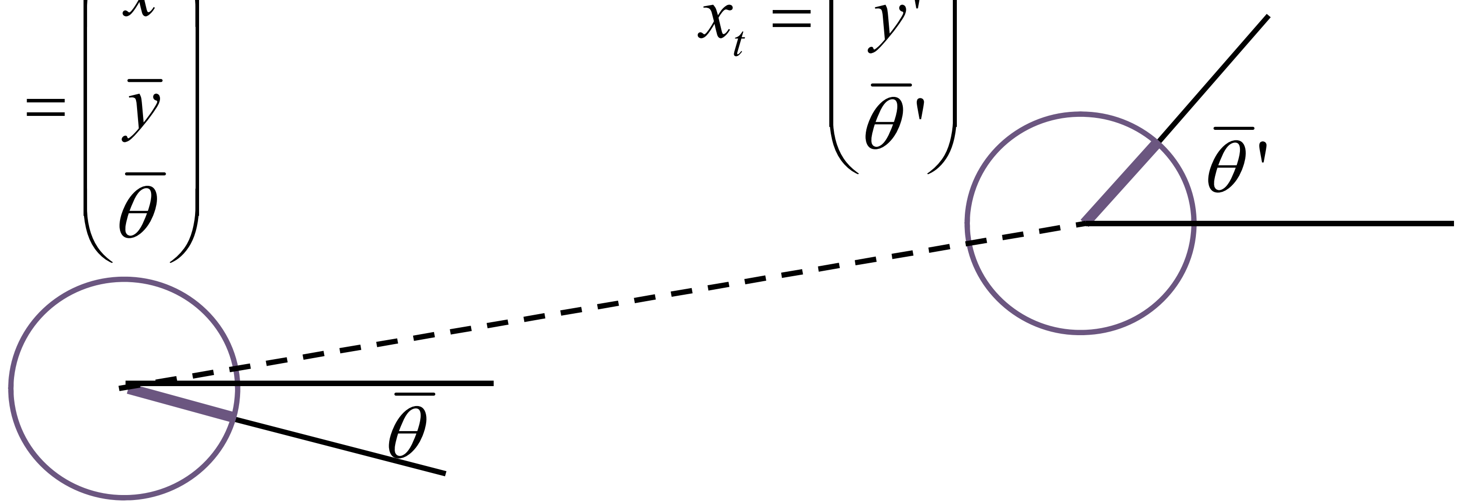
$$x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

given poses

Odometry Motion Model

- the key to computing $p(x_t \mid u_t, x_{t-1})$ for the odometry motion model is to remember that the robot has an internal estimate of its pose

$$\bar{x}_{t-1} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{\theta} \end{pmatrix} \qquad \bar{x}_t = \begin{pmatrix} \bar{x}' \\ \bar{y}' \\ \bar{\theta}' \end{pmatrix}$$



robot's internal poses

Odometry Motion Model

- the control vector is made up of the robot odometry

$$u_t = \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix}$$

- use the robot's internal pose estimates to compute the δ

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

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

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

Odometry Motion Model

- use the given poses to compute the δ

$$\hat{\delta}_{trans} = \sqrt{(x'-x)^2 + (y'-y)^2}$$

$$\hat{\delta}_{rot1} = \text{atan2}(y'-y, x'-x) - \theta$$

$$\hat{\delta}_{rot2} = \theta' - \theta - \hat{\delta}_{rot1}$$

- as with the velocity motion model, we have to solve the inverse kinematics problem here
  - but the problem is much simpler than in the velocity motion model

Odometry Motion Model

- recall the noise model

$$\delta_{trans} - \hat{\delta}_{trans} = \varepsilon_{\alpha_3 \hat{\delta}^2_{trans} + \alpha_4 (\hat{\delta}^2_{rot1} + \hat{\delta}^2_{rot2})}$$
$$\delta_{rot1} - \hat{\delta}_{rot1} = \varepsilon_{\alpha_1 \hat{\delta}^2_{rot1} + \alpha_2 \hat{\delta}^2_{trans}}$$
$$\delta_{rot2} - \hat{\delta}_{rot2} = \varepsilon_{\alpha_1 \hat{\delta}^2_{rot2} + \alpha_2 \hat{\delta}^2_{trans}}$$

which makes it easy to compute the probability densities of observing the differences in the δ

$$p_1 = \text{prob}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3 \hat{\delta}^2_{trans} + \alpha_4 (\hat{\delta}^2_{rot1} + \hat{\delta}^2_{rot2}))$$
$$p_2 = \text{prob}(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1 \hat{\delta}^2_{rot1} + \alpha_2 \hat{\delta}^2_{trans})$$
$$p_3 = \text{prob}(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1 \hat{\delta}^2_{rot2} + \alpha_2 \hat{\delta}^2_{trans})$$

Odometry Motion Model

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

Lines 2–4: odometry values (u)

Lines 5–7: values of interest (x,x')

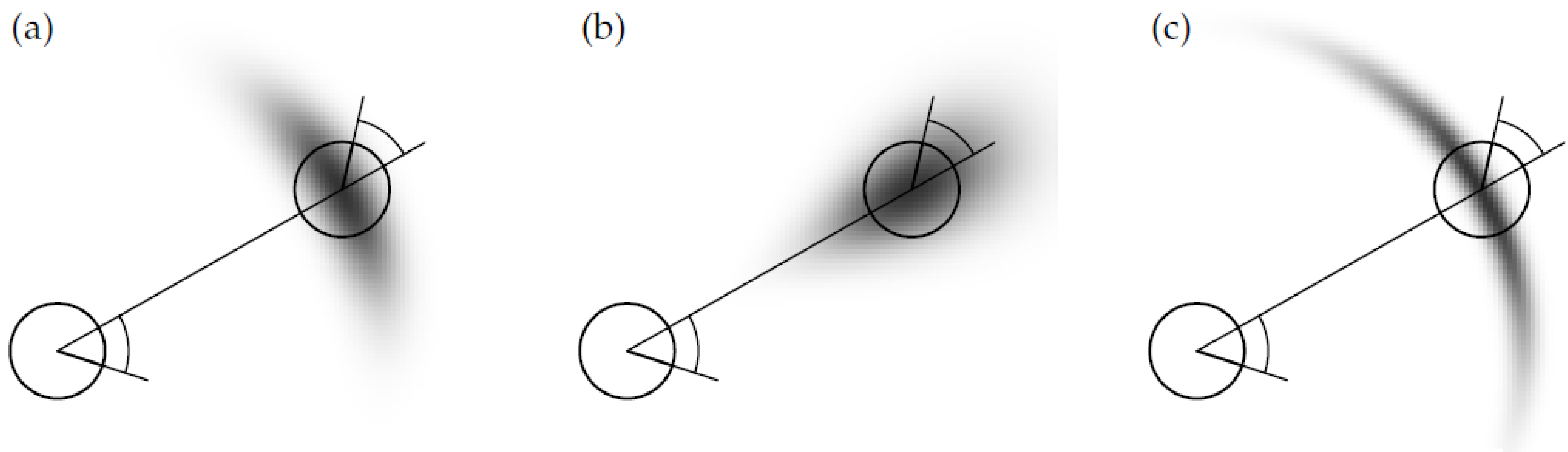
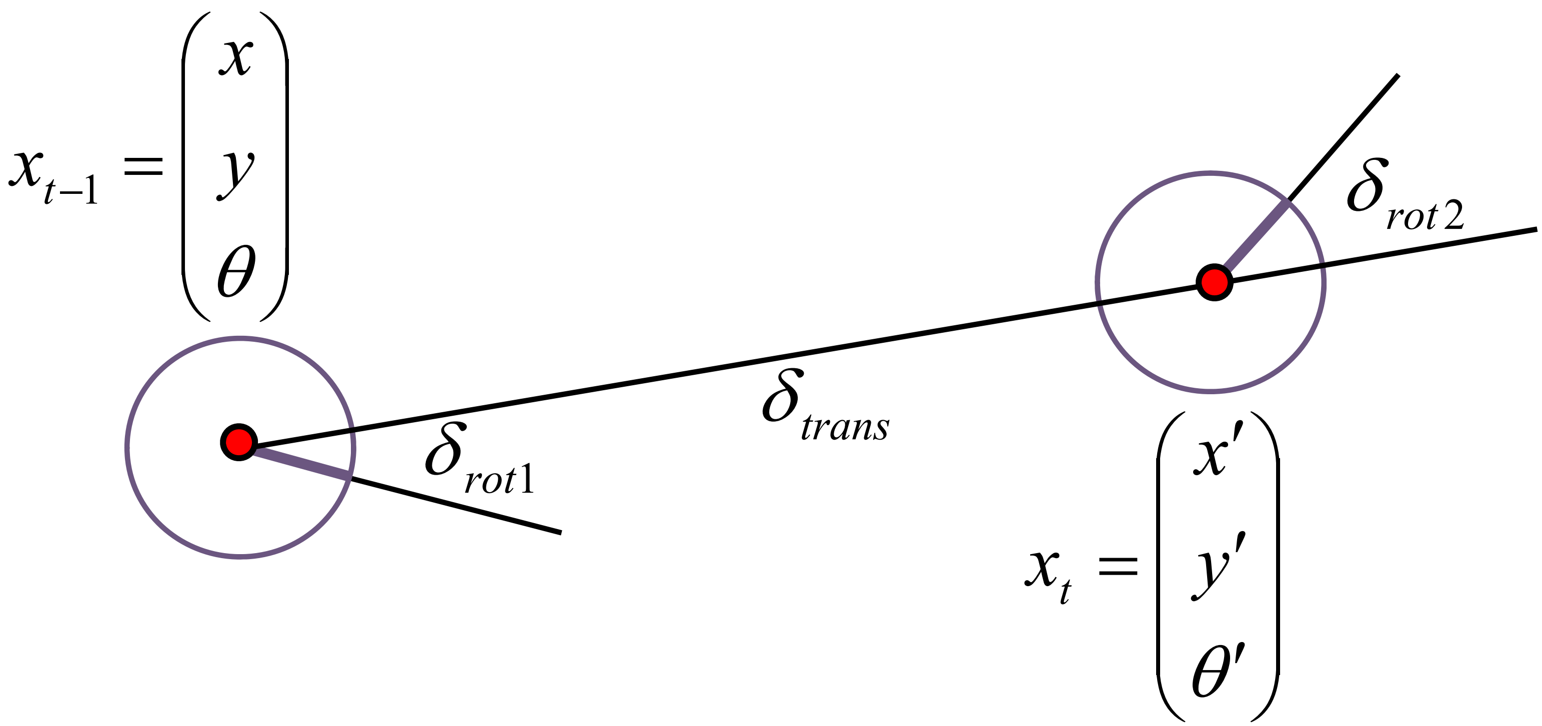


Figure 5.8 The odometry motion model, for different noise parameter settings.

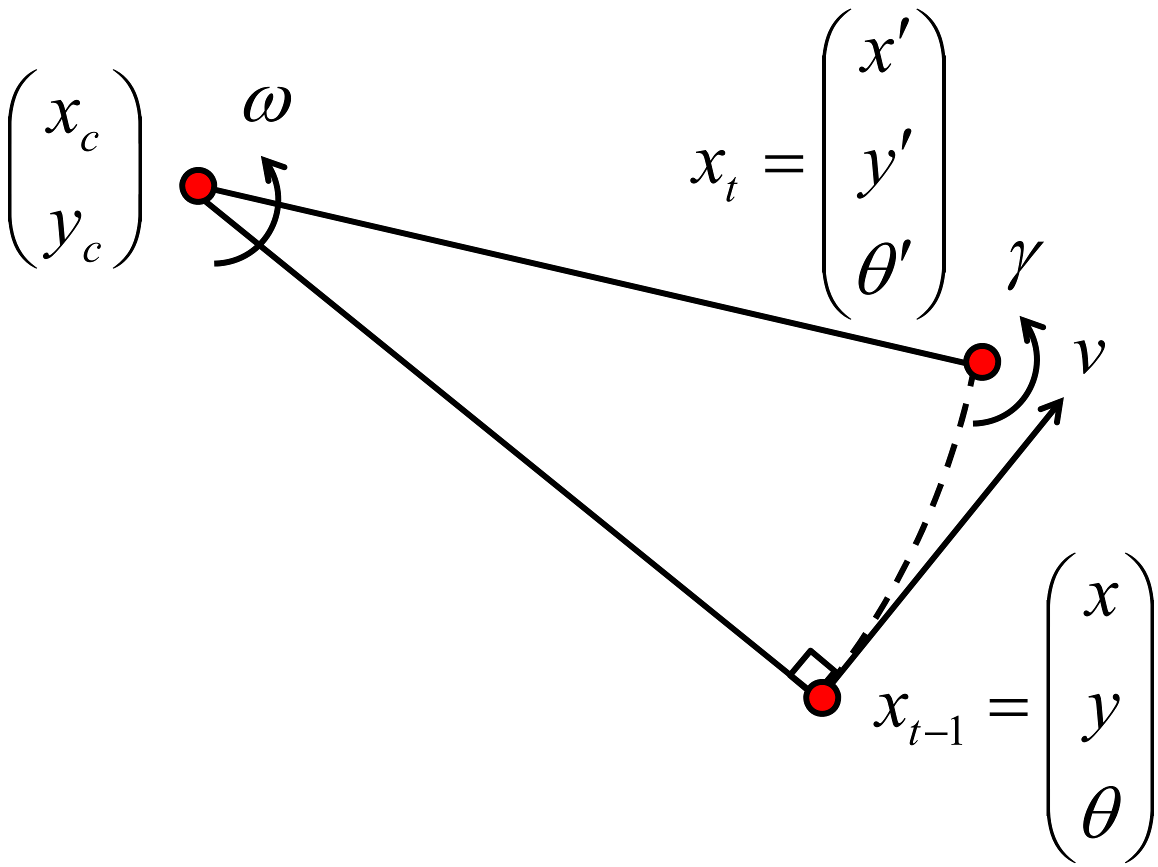
Recap

- odometric motion model
  - control variables were derived from odometry
    - initial rotation, translation, final rotation



Recap

- velocity motion model
  - control variables were linear velocity, angular velocity about ICC, and final angular velocity about robot center



Recap

- for both models we assumed the control inputs u_t were noisy
- the noise models were assumed to be zero-mean additive with a specified variance

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} v_{\text{noise}} \\ \omega_{\text{noise}} \end{pmatrix}$$

actual velocity — commanded velocity — noise

$$\text{var}(v_{\text{noise}}) = \alpha_1 v^2 + \alpha_2 \omega^2$$
$$\text{var}(\omega_{\text{noise}}) = \alpha_3 v^2 + \alpha_4 \omega^2$$

Recap

- for both models we assumed the control inputs u_t were noisy
- the noise models were assumed to be zero-mean additive with a specified variance

$$\begin{pmatrix} \hat{\delta}_{trans} \\ \hat{\delta}_{rot1} \\ \hat{\delta}_{rot2} \end{pmatrix} = \begin{pmatrix} \delta_{trans} \\ \delta_{rot1} \\ \delta_{rot2} \end{pmatrix} + \begin{pmatrix} \delta_{trans,noise} \\ \delta_{rot1,noise} \\ \delta_{rot2,noise} \end{pmatrix}$$

actual motion — commanded motion — noise

$$\mathrm{var}(\delta_{trans,noise}) = \alpha_3\, \hat{\delta}^2_{trans} + \alpha_4\, (\hat{\delta}^2_{rot1} + \hat{\delta}^2_{rot2})$$

$$\mathrm{var}(\delta_{rot1,noise}) = \alpha_1\, \hat{\delta}^2_{rot1} + \alpha_2\, \hat{\delta}^2_{trans}$$

$$\mathrm{var}(\delta_{rot2,noise}) = \alpha_1\, \hat{\delta}^2_{rot2} + \alpha_2\, \hat{\delta}^2_{trans}$$

Recap

- for both models we studied how to derive $p(x_t \mid u_t, x_{t-1})$
  - given
    - $x_{t\text{-}1}$ current pose
    - u_t control input
    - x_t new pose

    find the probability density that the new pose is generated by the current pose and control input
- required inverting the motion model to compare the *actual* with the *commanded* control parameters

Recap

- for both models we studied how to sample from $p(x_t \mid u_t, x_{t-1})$
  - given
    - $x_{t\text{-}1}$ current pose
    - u_t control input

  generate a random new pose x_t consistent with the motion model
- sampling from $p(x_t \mid u_t, x_{t-1})$ is often easier than calculating $p(x_t \mid u_t, x_{t-1})$ directly because only the forward kinematics are required