Robotics

Lecture 5: Monte Carlo Localisation

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

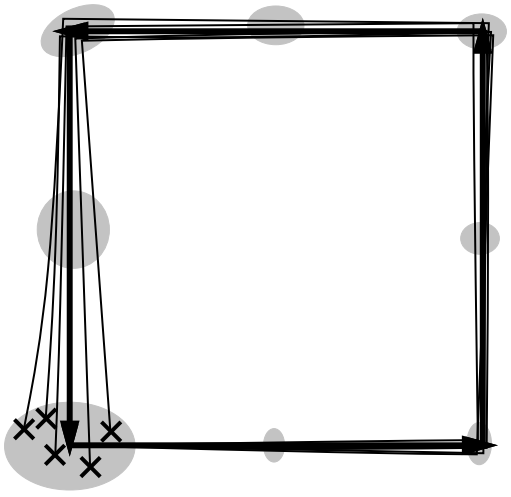
Andrew Davison
Department of Computing
Imperial College London

Review: Probabilistic Motion and Sensing Practical 4

The preliminaries for Monte Carlo Localisation:

- Updating a probability distribution, using particles, to represent motion uncertainty. We want to see the particles spreading out in a realistic manner to represent uncertainty over a square trajectory, and hear about how you found the right parameter settings to achieve this.
- Waypoint-based navigation.
- Investigating the performance of the sonar sensor probabilistically.

Motion Prediction



- We know that uncertainty grows during blind motion.
- So when the robot makes a movement, the particle distribution needs to shift its mean position but also spread out.
- We achieve this by passing the state part of each particle through a function which has a deterministic component and a random component.

Motion Prediction

- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos\theta \\ y + (D + e)\sin\theta \\ \theta + f \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$$

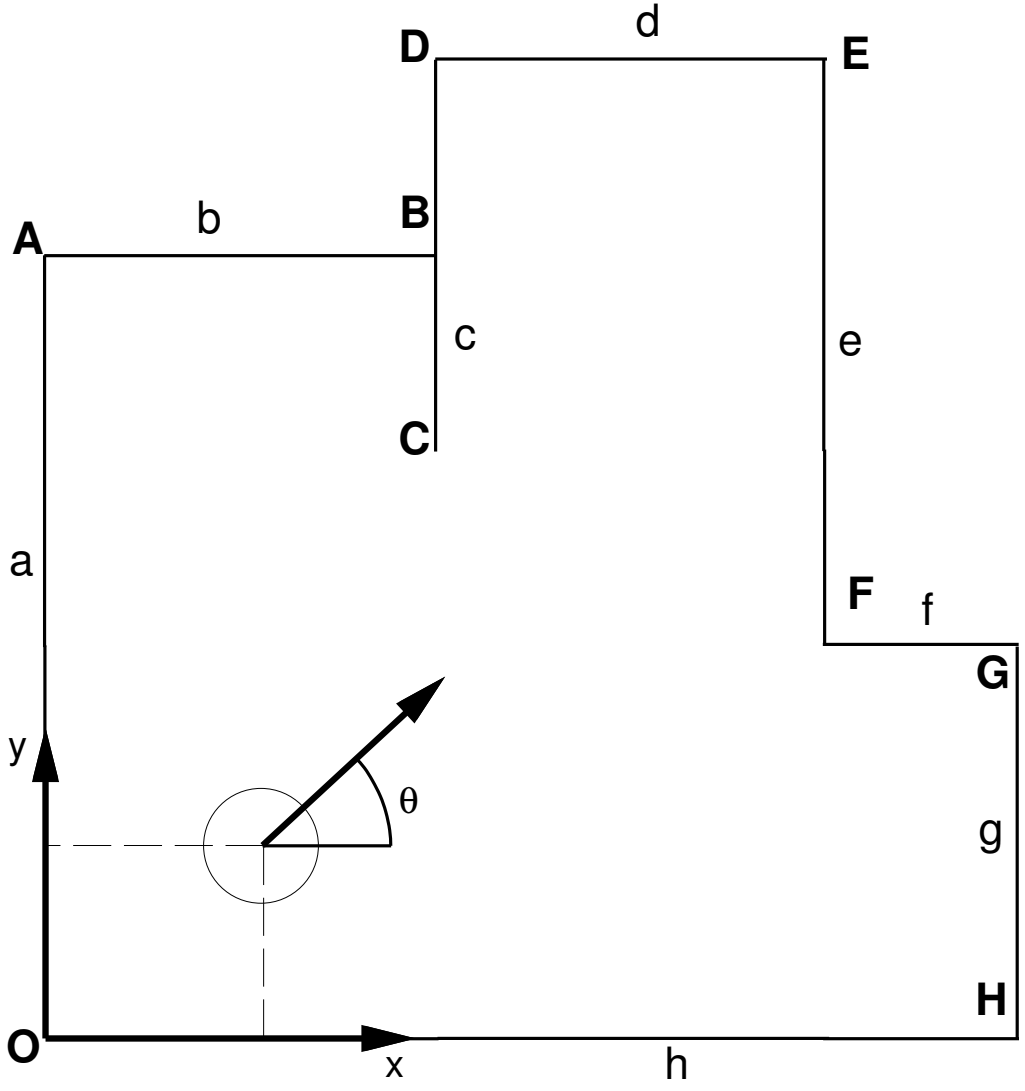
- Here e, f and g are zero mean *noise* terms — i.e. random numbers *sampled from* with a Gaussian distribution.
- The spreading out comes from sampling a different set of random numbers for each particle.

Review: Probabilistic Motion and Sensing Practical 4

- Although your robot may have been very precise in the square experiment we did in week 2, it is best to be conservative with the uncertainty values you use in a probabilistic filter such as MCL. Especially because the robots are now driving on variable carpet, odometry precision is likely to be less and you should make sure your particles spread out enough to represent this.
- This goes for the sonar sensor too: in a real navigation setting, there may be more variation (e.g. in how it responds to different surfaces, angles, etc.) than in your controlled experiments.
- In this practical we explicitly asked you to use fixed size steps (10cm and 90°), so you just need to find single values for the variances of each of e, f and g in the particle motion prediction equations. If you later on want to use varying distance and angle step sizes for particle filter updates, you can come up with constants which are scale constants representing the growth in uncertainty per cm or per degree. Be careful though that variance should vary linearly with distance or angle, not standard deviation. This is to make sure that you reach the same position uncertainty after one long movement as after two short movements of half the distance.

Monte Carlo Localisation (MCL)

In Practical 5 we will aim at repeatable localisation within an environment like this which we will set up with boards in the lab.



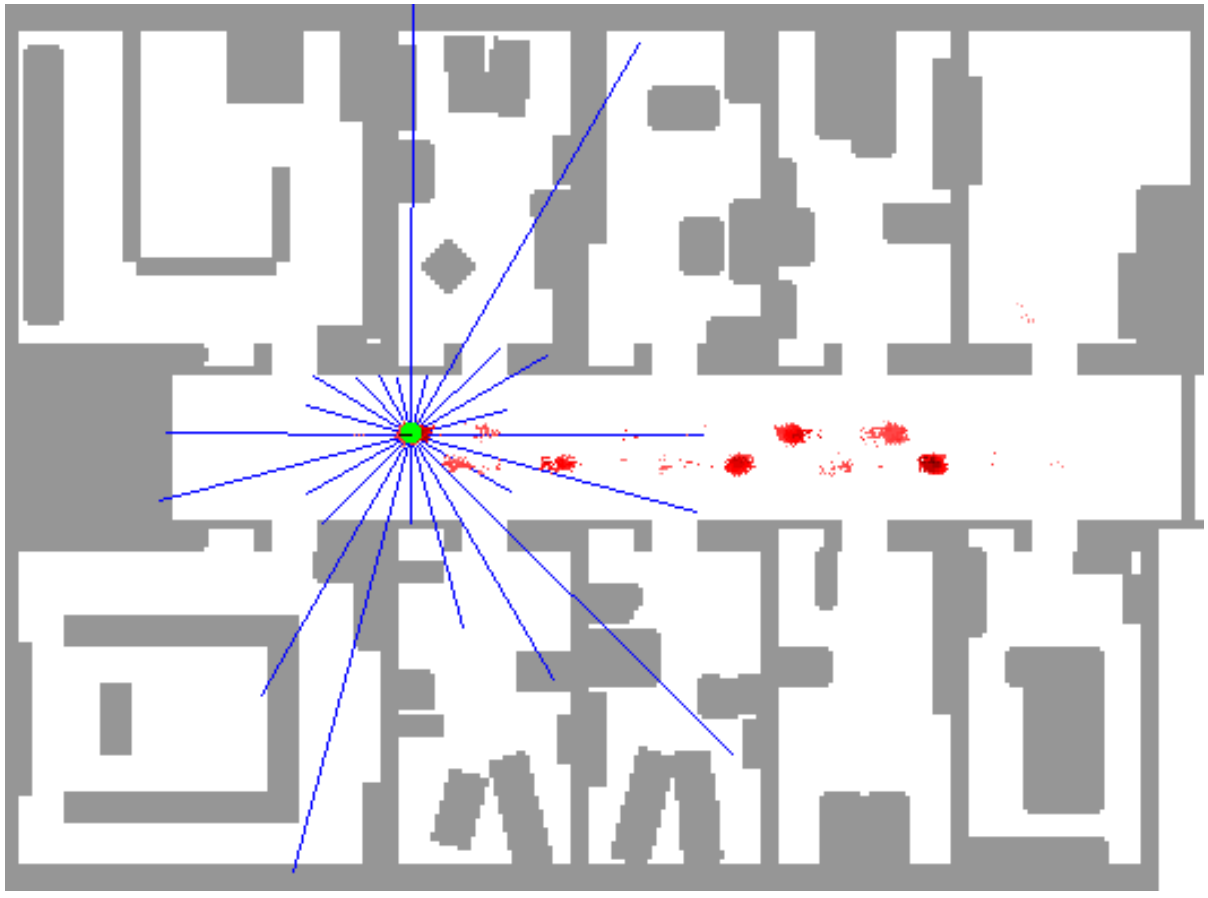
Monte Carlo Localisation (MCL)

- In MCL, a cloud of weighted particles represents the uncertain position of a robot.

Two ways of thinking about how MCL works:

- A Bayesian probabilistic filter.
- 'Survival of the fittest', like a genetic algorithm. After motion prediction, the particles are in a set of random positions which should span the possible positions into which the robot might have really moved. When a measurement is made from sonar, the likelihood each particle is assigned, according to how well it fits the measurement, is a 'fitness' score. Normalising and resampling then allow strong particles to reproduce (multiple copies are made of them), while weak particles die out.

MCL Successful Large-Scale Implementation



(Dieter Fox *et al.*1999, using sonar. See animated gif at `http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/montecarlolocalization.gif` .)

The Particle Distribution

- A particle is a point estimate $\mathbf{x}_i$ of the state (position) of the robot with a weight w_i.

$$\mathbf{x}_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}$$

- The full particle set is:

$$\{\mathbf{x}_i, w_i\} \ ,$$

  for $i = 1$ to N. A typical value might be $N = 100$.

- All weights should add up to 1. If so, the distribution is said to be *normalised*:

$$\sum_{i=1}^{N} w_i = 1 \ .$$

- Interpretation: the probability that the robot is within any region (multi-D volume) of the state space is the sum of all of the weights of particles within that region.

Continuous vs. Global Localisation

- Continuous localisation is a tracking problem: given a good estimate of where the robot was at the last time-step and some new measurements, estimate its new position.
- Global localisation is often called the 'kidnapped robot problem': the environment is known, but the robot's position within it is completely uncertain.

MCL can be used to tackle both of these problems; the only difference is in how we initialise the particle set.

Continuous vs. Global Localisation

- In continuous localisation we usually assume that we start from a perfectly known robot position: set the state of all particles to the same value. Set all weights to be equal. The result is a 'spike', completely certain point estimate of the robot's location.

$$\mathbf{x}_1 = \mathbf{x}_2 = \ldots = \mathbf{x}_N = \mathbf{x}_{init} \quad ; \quad w_1 = w_2 = \ldots w_N = 1/N$$

- In global localisation we start from knowledge only that the robot is somewhere within a certain region. The state of each particle should be sampled randomly from all of the possible positions within that region. Set all weights to be equal.

$$\mathbf{x}_i = Random \quad ; \quad w_1 = w_2 = \ldots w_N = 1/N$$

- With only one sonar sensor global localisation is very difficult, so in the practical we will attempt continuous localisation, initialising all particles to the same location.

Inferring an Estimate and Position-Based Navigation

- At any point, our uncertain estimate of the location of the robot is represented by the whole particle set.
- If we need to, we can make a point estimate of the current position and orientation of the robot by taking the mean of all of the particles:

$$\bar{\mathbf{x}} = \sum_{i=1}^{N} w_i \mathbf{x}_i \ .$$

- i.e. the means of the x, y and θ components are all calculated individually and stored in $\bar{\mathbf{x}}$.
- The robot could use this for instance to plan A $\rightarrow$ B waypoint navigation.

Steps in MCL/Particle Filter

These steps are repeated every time the robot moves a little and makes measurements:

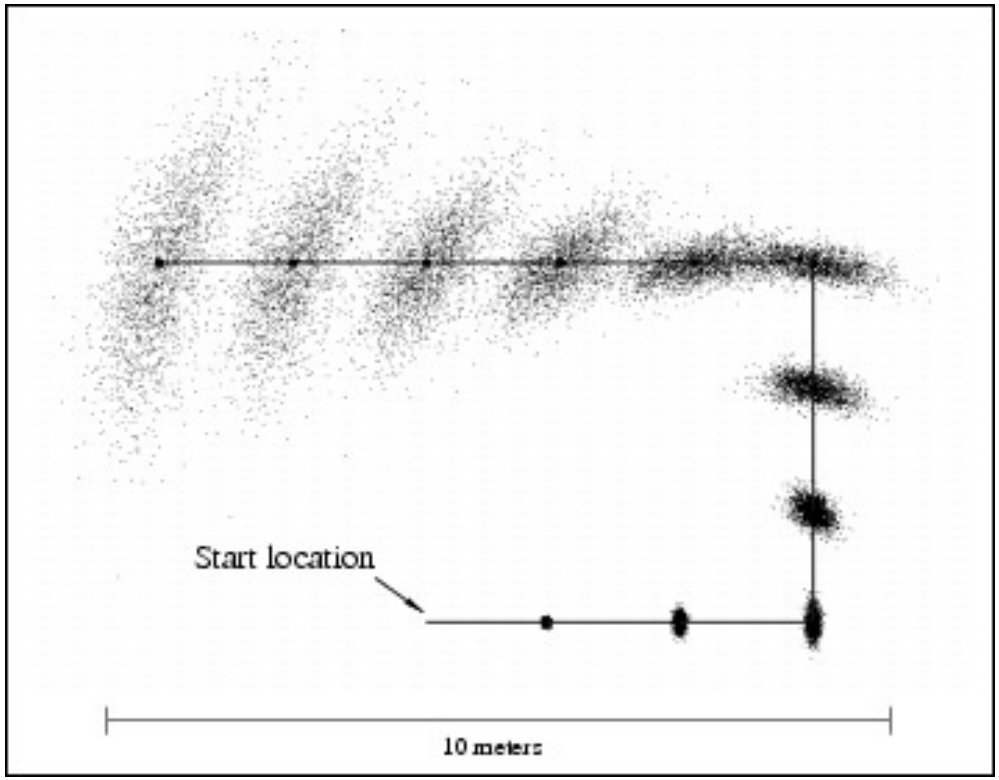
1. Motion Prediction based on Odometry
2. Measurement Update based on Outward Looking Sensors (Sonar)
3. Normalisation
4. Resampling

Motion Prediction

- See details from the last lecture.
- Watch out for angular wrap-around — i.e. make sure that θ values are always in the range $-\pi$ to π.
- Solve this by simply adding or subtracting 2π to any particles which go out of this range.

Displaying a Particle Set

- We can visualise the particle set by plotting the x and y coordinates as a set of dots; more difficult to visualise the θ angular distribution (perhaps with arrows?) — but we can get the main idea just from the linear components.



- This example shows a robot only using repeated motion prediction — the particles get more and more spread out.

Measurement Updates

- A measurement update consists of applying Bayes Rule to each particle; remember:

$$P(\mathbf{X}|\mathbf{Z}) = \frac{P(\mathbf{Z}|\mathbf{X})P(\mathbf{X})}{P(\mathbf{Z})}$$

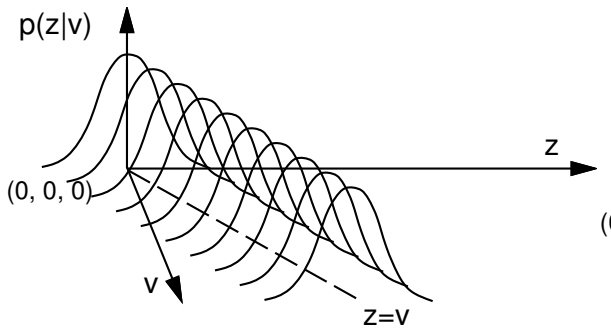
- So when we achieve a measurement z, we update the weight of each particle as follows:

$$w_{i(new)} = P(z|\mathbf{x}_i) \times w_i \ ,$$

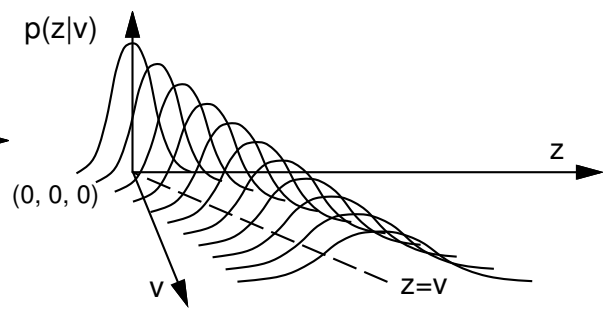
  remembering that the denominator in Bayes' rule is a constant factor which we do not need to calculate because it will later be removed by normalisation.
- $P(z|\mathbf{x}_i)$ is the *likelihood* of particle i; the probability of getting measurement z given that $\mathbf{x}_i$ represents the true state.

Likelihood Functions

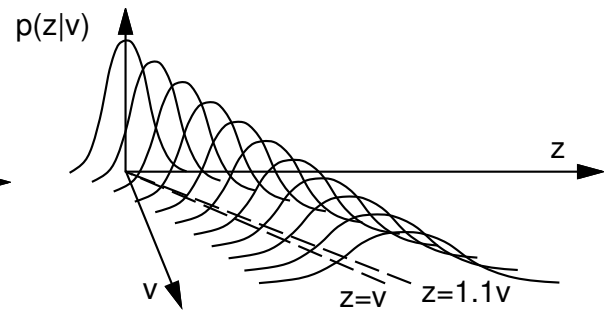
- A likelihood function fully describes a sensor's performance.
- Its form can be determined by repeated experiments with the sensor: make multiple many measurements at each of a set of precisely measured ground truth values, and calculate the statistics.
- $p(z|v)$ is a function of both measurement variables z and ground truth v and can be plotted as a probability surface. e.g. for a depth sensor:



Constant Uncertainty

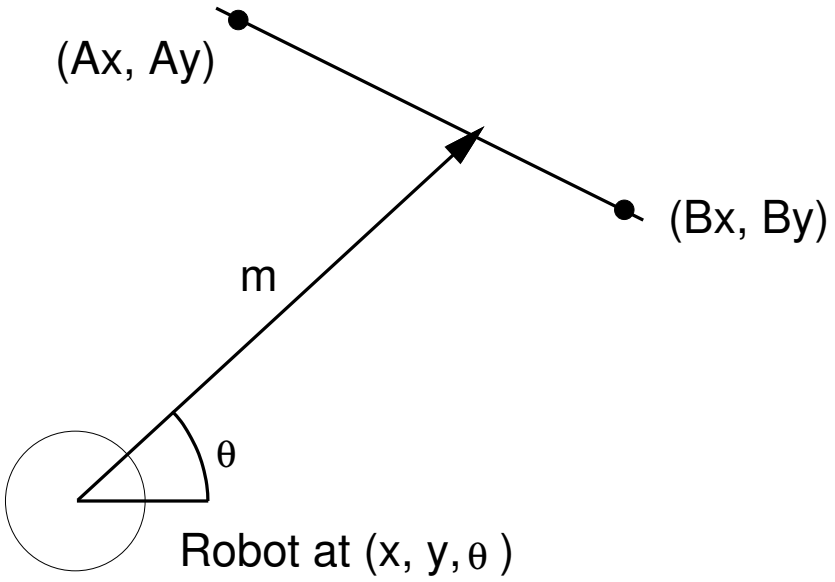


Growing



Systematic Error (Biased)

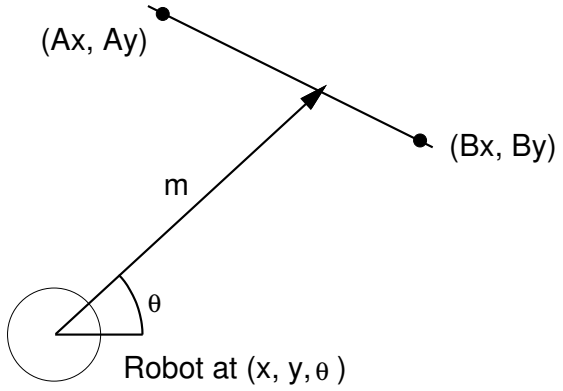
Measurement Update: How do we get the Ground Truth Value?



- If robot is at pose (x, y, θ) then its forward distance to an infinite wall passing through (A_x, A_y) and (B_x, B_y) is:

$$m = \frac{(B_y - A_y)(A_x - x) - (B_x - A_x)(A_y - y)}{(B_y - A_y)\cos\theta - (B_x - A_x)\sin\theta} \ .$$

Measurement Update: Sonar



- The world coordinates at which the forward vector from the robot will meet the wall are:

$$\begin{pmatrix} x + m\cos\theta \\ y + m\sin\theta \end{pmatrix} .$$

  Using this you can check if the sonar should hit between the endpoint limits of the wall.
- If the sonar would independently hit several of these walls, obviously the closest is the one it will actually respond to.

Likelihood for Sonar Update

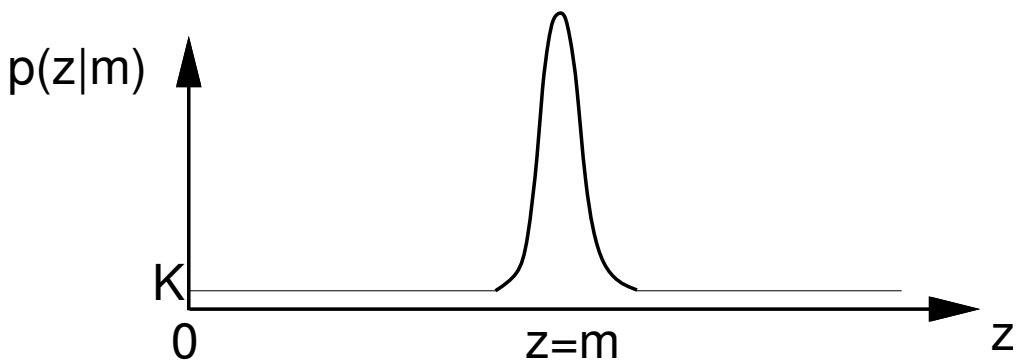
- The likelihood should depend on the difference $z - m$: if this is small, then the measurements validates a particle; if it is big it does not and weakens the particle.
- The further away the measurement is from the prediction, the less likely it is to occur. A Gaussian function is usually a good model for the mathematical form of this. The standard deviation σ_s is based on our model of how uncertain the sensor is, and may depend on z or may be constant.

$$p(z|m) \propto e^{\frac{-(z-m)^2}{2\sigma_s^2}}$$

- Note the difference between this and the motion prediction step. There we *sampled* randomly from a Gaussian distribution to move each particle by a slightly different amount. Here in the measurement update we just *read off* a value from a Gaussian function to obtain a likelihood for each particle.

Robust Likelihood for Sonar Update

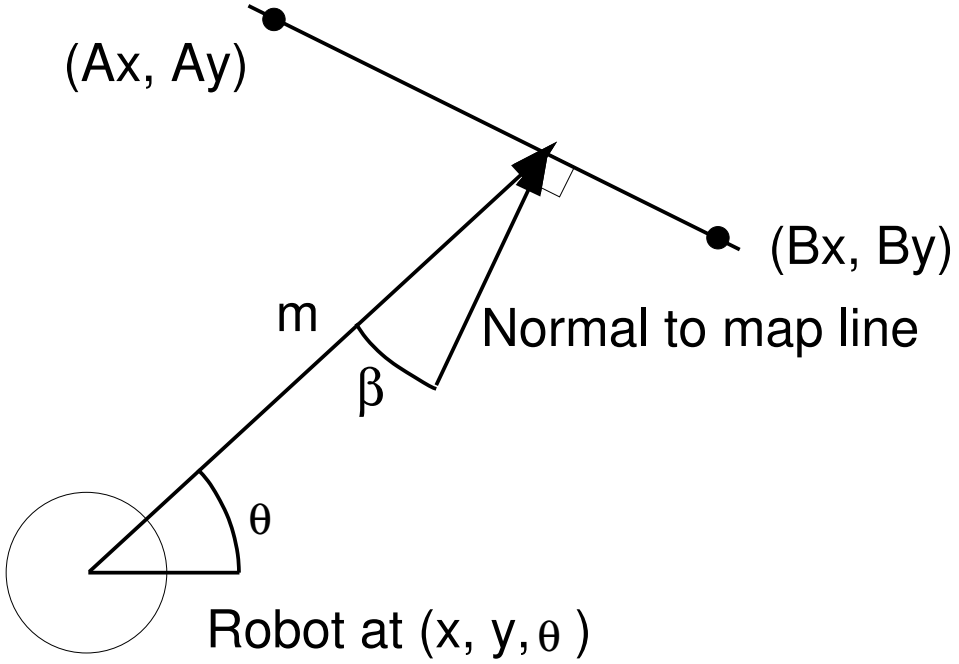
- A *robust* likehood function models the fact that real sensors sometimes report 'garbage' values which are not close to ground truth. Robust functions have 'heavy tails'. This can be achieved most simply by adding a constant to the likelihood function. The meaning of this is that there is some constant probability that the sensor will return a garbage value, uniformly distributed across the range of the sensor.



$$p(z|m) \propto e^{\frac{-(z-m)^2}{2\sigma_s^2}} + K$$

- The effect of a robust likelihood function in MCL is that the filter is less aggressive in 'killing off' particles which are far from agreeing with measurements. An occasional garbage measurement will not lead to the sudden death of all of the particles in good positions.

Likelihood for Sonar Update



- Also relevant may be the angle between the sonar direction and the normal to the wall. If this is too great, probably the sonar will not give a sensible reading and you should ignore it.

$$\beta = cos^{-1}\left(\frac{\cos\theta(A_y - B_y) + \sin\theta(B_x - A_x)}{\sqrt{(A_y - B_y)^2 + (B_x - A_x)^2}}\right)$$

Normalisation

- The weights of all particles should be scaled so that they again add up to 1.
- Calculate $\sum_{i=1}^{N} w_i$ and divide all existing weights by this:

$$w_{i(new)} = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Resampling

- Resampling consists of generating a new set of N particles which all have equal weights $1/N$, but whose spatial distribution now reflects the probability density.
- To generate each of the N new particles, we copy the state of one of the previous set of particles with probability according to that particle's weight.
- This is best achieved by generating the *cumulative probability distribution* of the particles, generating a random number between 0 and 1 and then picking the particle whose cumulative probability this intersects.
- Note that for efficiency it is possible to skip the normalisation step and resample directly from an unnormalised distribution.

Another Measurement Possibility: Compass Sensor

A digital compass gives a robot the ability to estimate its rotation *without drift*.

- Having a digital compass and a single sonar sensor puts us in a position close to that of having a full sonar ring.
- In principle, the robot could stop and rotate on the spot every so often to simulate a sonar ring.
- We could simply monitor the compass as the robot continuously moves and feed this into our filter.
- But we won't try it today, because the NXT compasses don't work very well in the lab.

Measurement Update: Compass Sensor

- Compass measures bearing β relative to north. What is $P(\beta|\mathbf{x}_i)$?
- Clearly the likelihood only depends on the θ part of $\mathbf{x}_i$.
- If the compass is working perfectly, then:

$$\beta = \gamma - \theta \ ,$$

  where γ is the magnetic bearing of the x coordiante axis of frame W.
- So we should assess the uncertainty in the compass, and set a likelihood which depends on the difference between β and $\gamma - \theta$; e.g. Gaussian $e^{\frac{-(\beta-(\gamma-\theta))^2}{2\sigma_c^2}}$
- Particles far from the right orientation will get low weights.