Robotics

Lecture 6: Advanced Sensing: Place Recognition and Occupancy Mapping

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

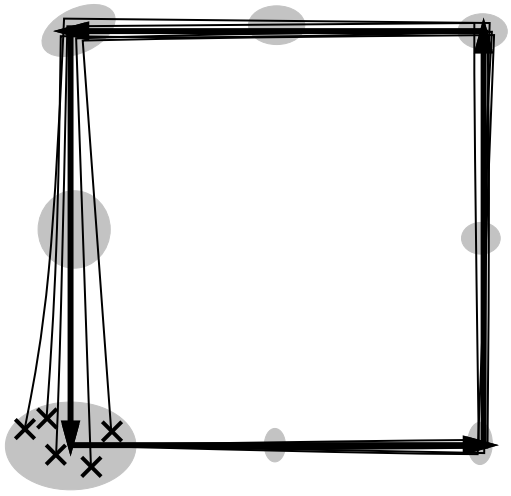
Andrew Davison
Department of Computing
Imperial College London

Review: Probabilistic Motion and Sensing Practical 3

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.
- Sonar sensor investigation.

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:

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

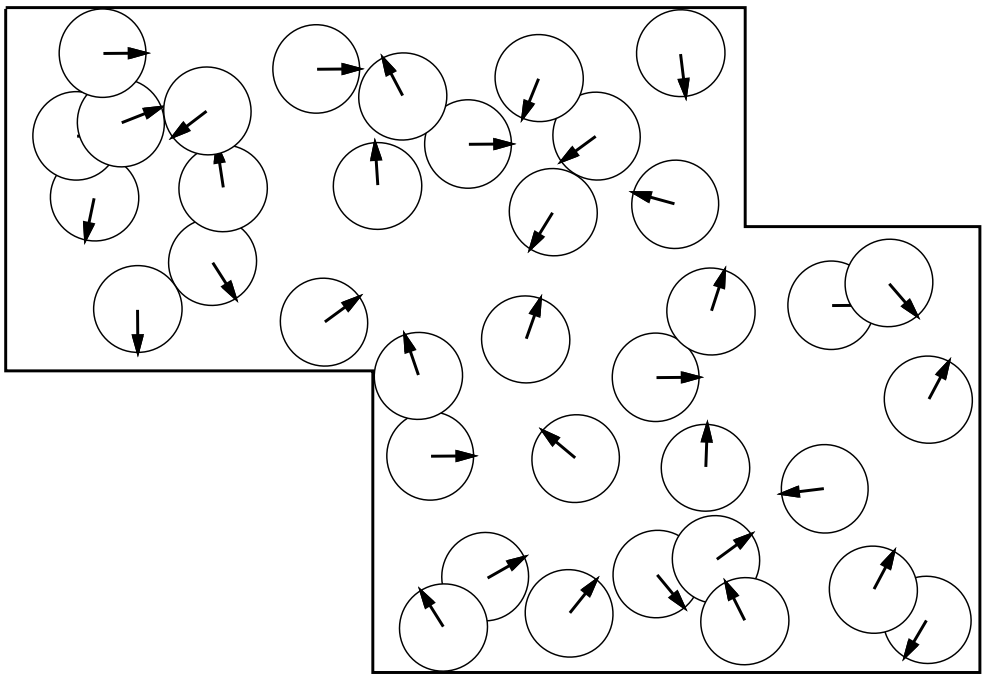
- Here e, f and g are zero mean *noise* terms — i.e. random numbers *sampled from* with a Gaussian distribution.
- The spreading out comes from sampling a different set of random numbers for each particle.

Review: Probabilistic Motion and Sensing Practical 3

- 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.

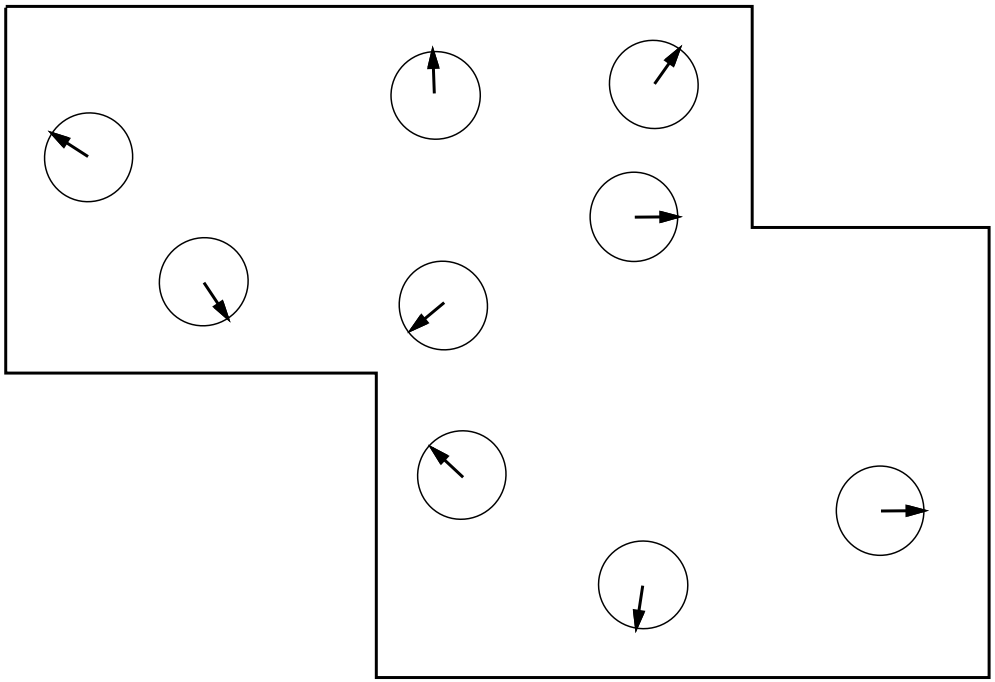
Global Localisation ('Kidnapped Robot') with Sonar

- In MCL, global localisation can be attempted by initialising a large number of particles randomly spread through the environment, and then running the filter normally. However, this requires many particles (computationally expensive) and it may take many movements and measurements to find the right location.
- We would expect better performance with more informative sensing: e.g. a ring of sonar sensors all making measurements at the same time rather than just one.



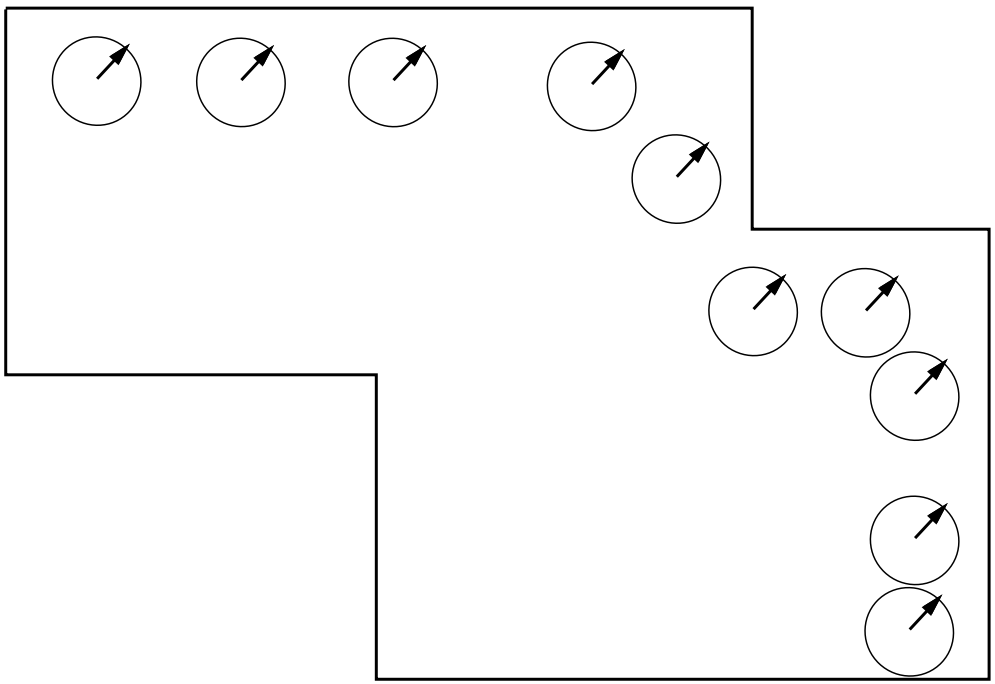
One Depth Measurement and Resampling

- After a measurement (e.g. sonar depth = 20cm), the weights of particles consistent with it will increase.
- Movement and further measurements are needed to lock down position, and ambiguities may still arise.



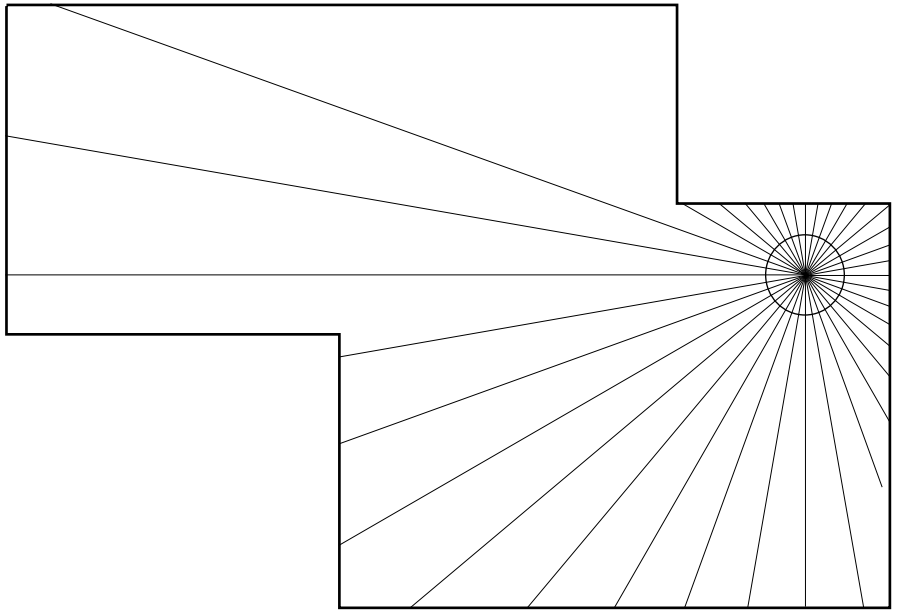
Using a Compass and Sonar Together

- Ambiguity is much reduced with extra measurements, such as from a compass.
- e.g. sonar depth = 20cm, compass bearing = 45°.



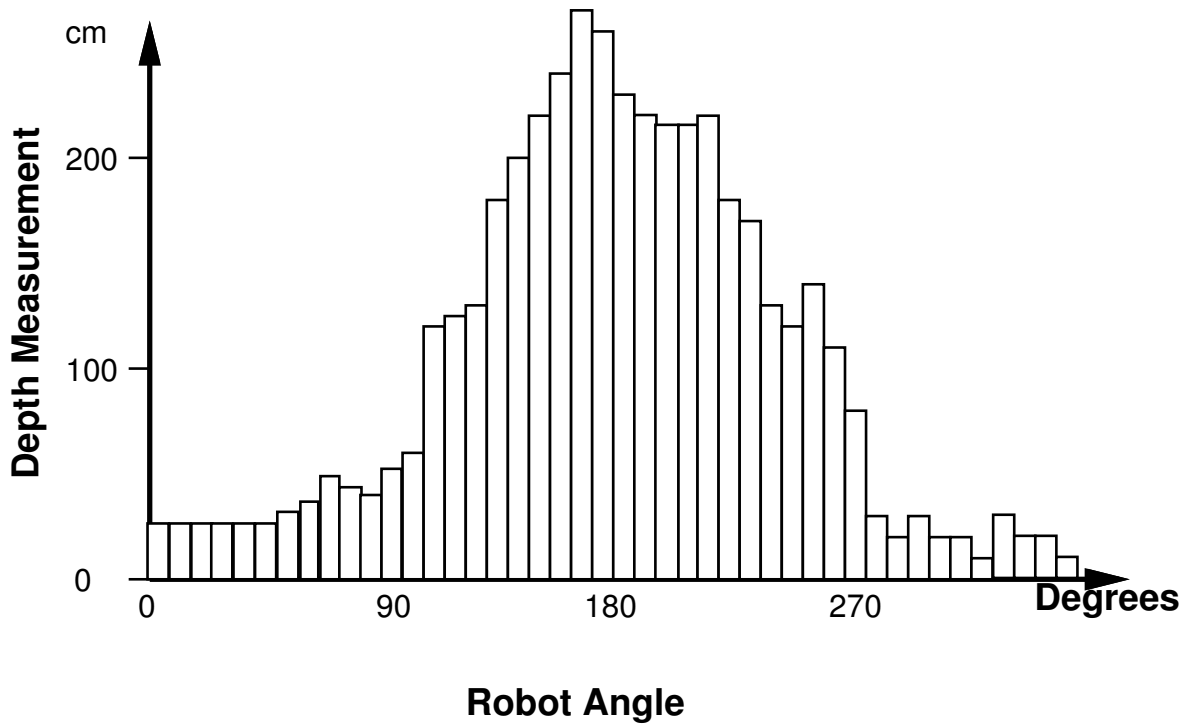
Global Localisation Via Recognition

- An alternative relocalisation technique involves making a lot of measurements at a number of chosen locations and *learning* their characteristics.
- This can be done without a prior map but needs *training*.
- The robot can only recognise the locations it has learned.
- For instance: at each location, spin the robot and take a regularly spaced set of sonar measurements (e.g. one per degree).



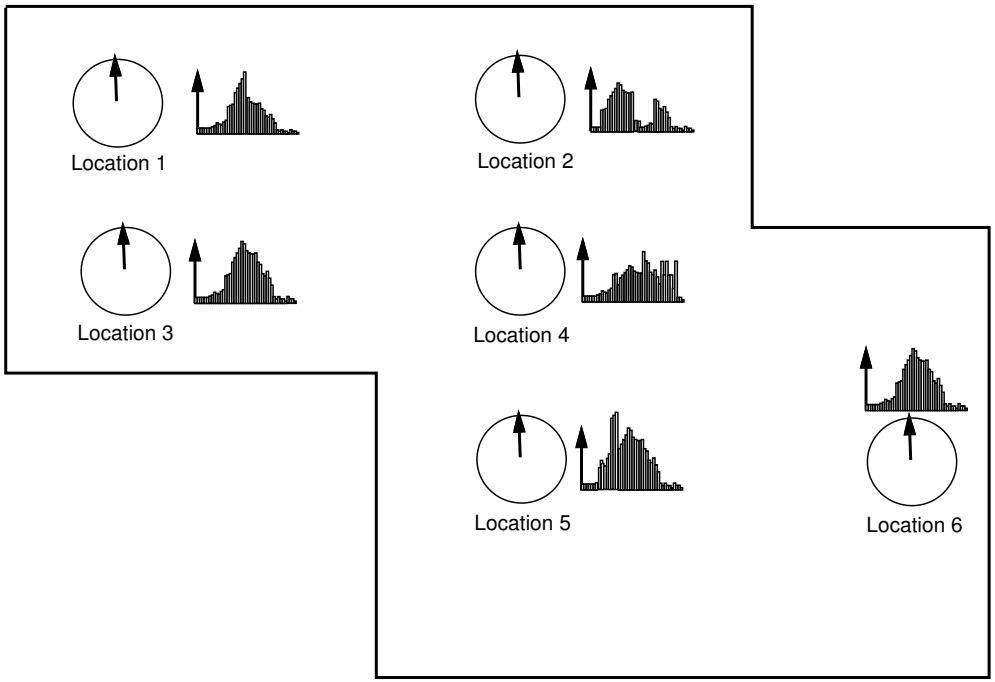
Measuring to Learn a Location

- First the robot must be placed in each target location to learn its appearance.
- The raw measurements are stored to describe the location: a place descriptor, or *signature*.



Place Recognition

- Afterwards, the robot is placed in one of the locations and it must take a set of measurements then decide which it is in.
- It must see which saved signature matches best to the new measurements by checking each in turn.



Place Recognition

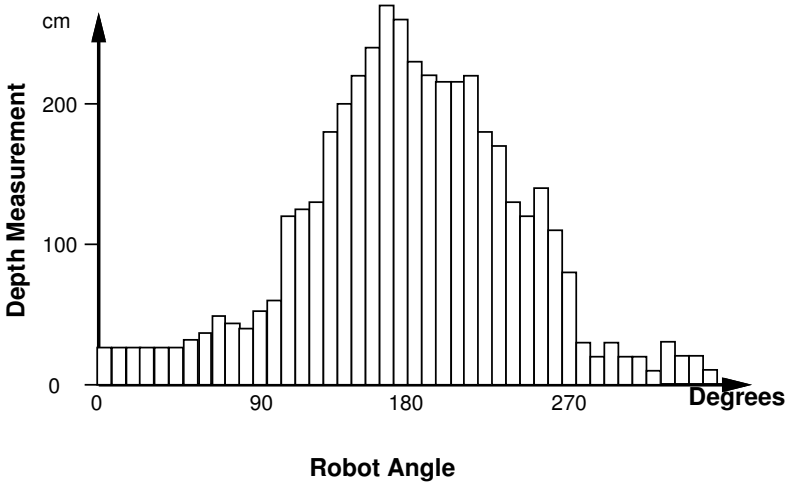
- Two histograms can be compared with a correlation test, measuring the sum of squared differences. Difference D_k between new measurement histogram $H_m(i)$ and saved signature histogram $H_k(i)$ is:

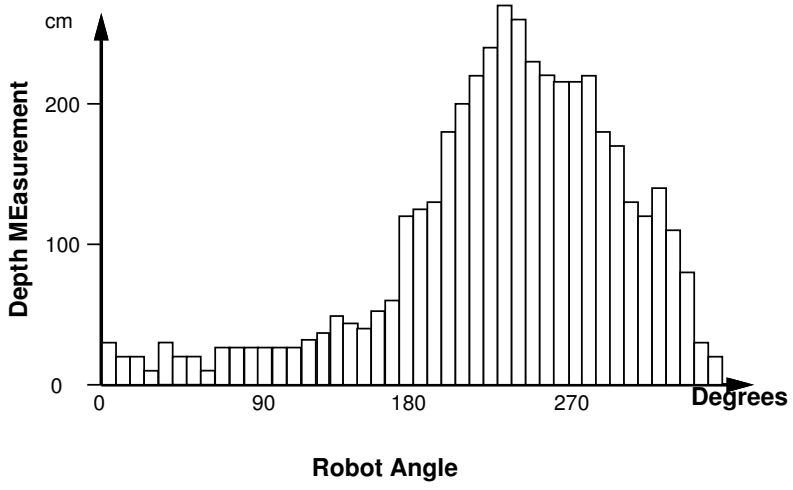
$$D_k = \sum_i \left(H_m(i) - H_k(i)\right)^2 \ .$$

- The saved location with lowest D_k is the most likely candidate, but we should also check that D_k is below a threshold in case the robot is in none of the known locations.

Estimating Orientation

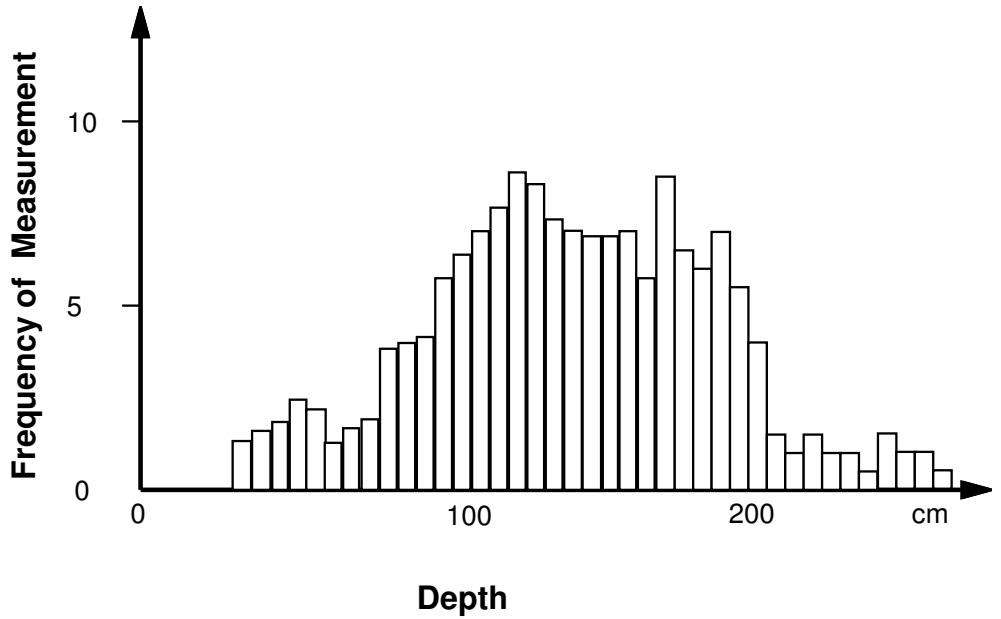
- If the test histogram and that from one of the saved locations can be brought into close agreement by only a shift, the robot is in the same place but rotated.
- The amount of shift to get the best agreement is a measurement of the rotation.



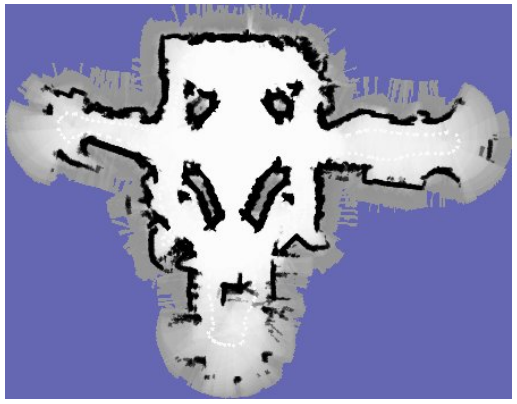


Depth Measurement Histogram

- Optionally (to save the computational cost of always trying every shift), we can build a signature which is *invariant* to robot rotation, such as a histogram of occurences of certain depth measurements.
- Matching tests can then be carried out on this directly.
- Once the correct location has been found, the shifting procedure to find the robot's orientation need only be carried out for that one location.

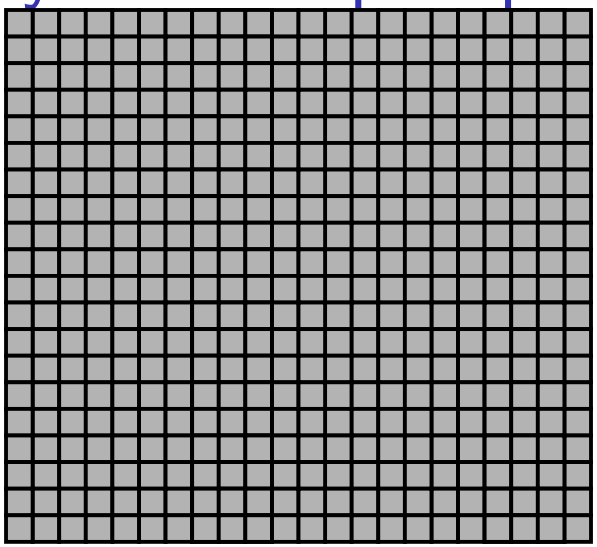


Probabilistic Occupancy Grid Mapping



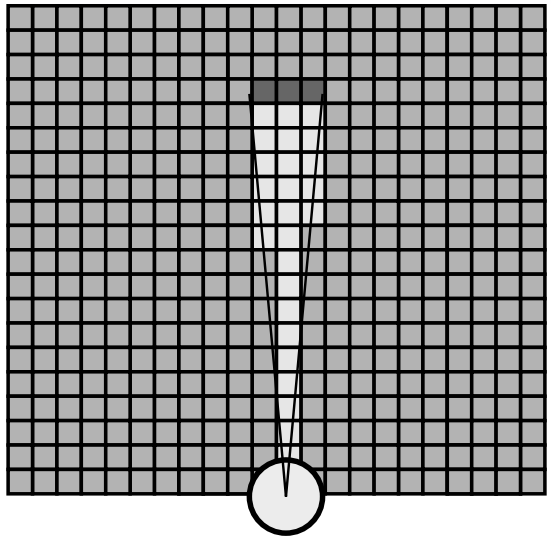
- The second method we will look at today is a probabilistic algorithm for *mapping* in the case that a robot's localisation is known. The goal is to infer which parts of the environment around a robot are navigable free-space, and which parts contain obstacles.
- In Occupancy Mapping, rather than building a parametric map of the positions of walls we use a regular grid representation.
- Occupancy grids accumulate the uncertain information from sensors like sonar to solidify towards precise maps.

Occupancy Grid Map Representation



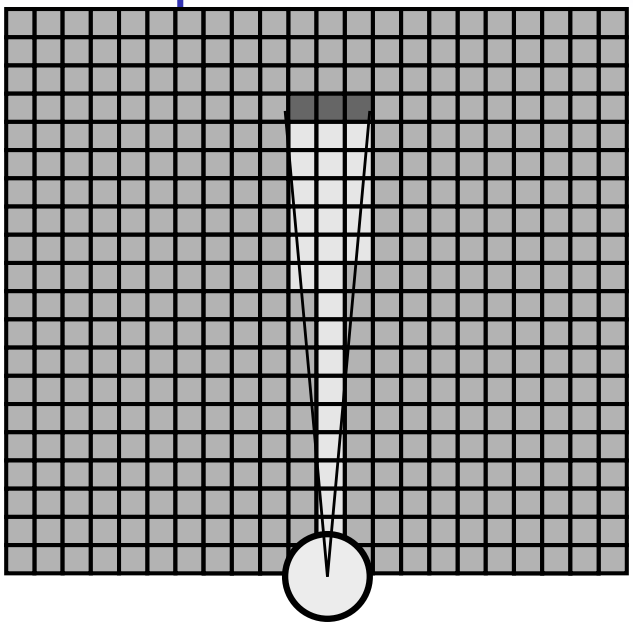
- We define an area on the ground we would like to map, and choose a square grid cell size.
- For each cell i, we store and update a *probability of occupancy* $P(O_i)$ that it is occupied by an obstacle.
- $P(E_i)$ is the corresponding probability that the cell is empty, where $P(O_i) + P(E_i) = 1$.
- We initialise the occupancy probabilities for unexplored space to a constant prior value; for instance 0.5
- Occupancy maps are often visualised with a greyscale value for each cell: from black for $P(O_i) = 1$ to white for $P(O_i) = 0$; intermediate values are shades of grey.

Update after Sonar Measurement



- For each cell we want to update the probability of occupancy to take account of a new sonar measurement Z.
- Suppose that the sonar reports a depth $Z = d$. This provides evidence that cells around distance d in front of the robot are more likely to be occupied. But also, that cells in front of the robot *at depths less than d* are more likely to be empty.
- A sonar beam is not a perfect ray but has a width (e.g. 10–15° as it spreads out and we can take account of this as shown.
- For each cell, we must test if it lies within the beam given the robot's position. We do not learn anything about cells beyond the beam width or beyond the measured depth.

Bayesian Update of Occupancy



Technically, for each cell, we apply Bayes rule to get a posterior probability for each cell.

$$P(O_i|Z) = \frac{P(Z|O_i)P(O_i)}{P(Z)}$$

As in MCL, we could avoid calculating $P(Z)$ by also calculating $P(E_i|Z)$ and normalising since we know $P(O_i|Z) + P(E_i|Z) = 1$.

$$P(E_i|Z) = \frac{P(Z|E_i)P(E_i)}{P(Z)}$$

Log Odds Representation

If we take the ratio of the two Bayes rule expressions on the previous slide:

$$\left(\frac{P(O_i|Z)}{P(E_i|Z)}\right) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * \left(\frac{P(O_i)}{P(E_i)}\right)$$

We can use the odds notation: $o(A) = \frac{P(A)}{P(\bar{A})}$.

$$o(O_i|Z) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * o(O_i)$$

Taking logs:

$$\ln o(O_i|Z) = \ln \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) + \ln o(O_i)$$

So in this form, for each cell we store $\ln o(O_i)$ and update it additively. Cells with probability 0.5 of occupancy will have log odds 0; positive log odds means probability > 0.5; negative log odds means probability < 0.5. Also we normally cap log odds at certain positive and negative limits.

Likelihood Model of Sensor

- We need models for the likelihood function which models sensor performance. We can consider directly the ratio of likelihoods we need to update log odds.
- Log odds update $U = \ln \frac{P(Z|O_i)}{P(Z|E_i)}$: for each cell $P(Z|O_i)$ is the probability of obtaining the sensor value we did given that the cell is occupied; $P(Z|E_i)$ is the probability of obtaining that value given that the cell is empty.
- For cells within the sonar beam but closer than the measured depth $Z = d$: $\frac{P(Z|O_i)}{P(Z|E_i)}$ is less than 1; we can choose a constant negative value for U.
- For cells within the sonar at around the measured depth $Z = d$: $\frac{P(Z|O_i)}{P(Z|E_i)}$ is greater than 1; we can choose a constant positive value for U.
- Note that we are somewhat oversimplifying in occupancy grids in assuming the the probabilities of occupancy for each cell are independent.

Occupancy Grid Mapping

- Over time and robot motion, measurements accumulate and the occupancy map converges towards white and black indicating definite knowledge.
- Sonar occupancy mapping example at `http://www.youtube.com/watch?v=aEeS8hDnnYg`
- An occupancy map's probability values must be thresholded if a decision is to be made about where the robot can actually drive.
- Large scale occupancy maps are very memory intensive; and subject to drift due to localisation uncertainty. We will look at methods to resolve this next week.