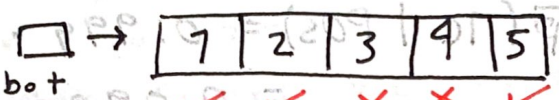
AI for robotics — Udacity

Localization — Monte Carlo

When you do localization using ML or DL we analyze the probability of "having x or y features in our map". We may describe it with a uniform maximum confusion function... and we update it when we explore.



✓: clear
×: obstacle

pHit: (0-1)
pMiss: (0-1)

We sense the cells and we assign prob. to each cell. The sensing helps update the p that there is an object at any of the cells.

Motion

We may have exact or inexact motion.
In exact motion we only shift the spaces or features based on the movement of the robot.
In inexact motion we assign a probability to landing in a cell that is near, but was not the target.

Having a function for sense( ) and one for move( ), entropy will decrease when we sense and increase when we move.

- Entropy = $\sum(-p \times \log(p))$

In general, localization can be defined by the following parts:

- Belief → Probability
- Sense → Product, followed by normalization
- Move → Convolution

Probability

$0 \leq P(X) \leq 1$, $\sum P(x_i) = 1$

For measurements, and where x = grid cell and z = measurement. $p(x_i|z) = \frac{P(z|x_i)\ P(x_i)}{P(z)}$

AI for robotics

Example use of Baye's Theorem (Cancer Test):

$P(C) = 0.001$

$P(\neg C) = 0.999$

$P(Pos \mid C) = 0.8$

$P(Pos \mid \neg C) = 0.1$

$P(C \mid Pos) = ?$

Answer:

$\overline{P}(C \mid Pos) = 0.001 \cdot 0.8 = 0.0008$

$\overline{P}(\neg C \mid Pos) = 0.999 \cdot 0.1 = 0.0999$

$\alpha = 0.1007$

$P(C \mid Pos) = \frac{0.0008}{0.1007} = 0.0079$

Total probability — Motion

$$P(X_i^t) = \sum_j P(X_j^{t-1}) \cdot P(X_i \mid X_j)$$ {Theorem of Total Probability}

t = time

i = grid cell

PHit and PMiss

These values represent the probabilities of the sensor(s) being wrong. In the real world they are assigned by computing the noise the sensor would experience and may be set experimentally.

Tracking — Kalman Filters

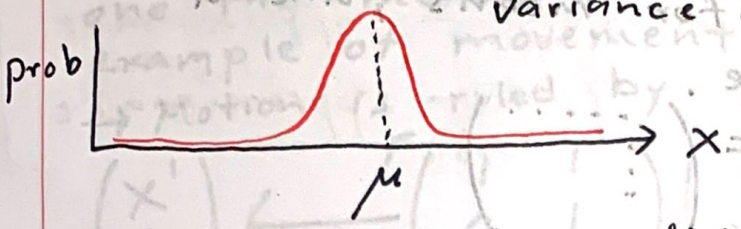
They give a uni-modal, continuous approximation of the state, as opposed to the discrete multi-modal Monte Carlo localization technique.

In Monte Carlo localization the space may be as follows:

{where the space is a histogram of probabilities}

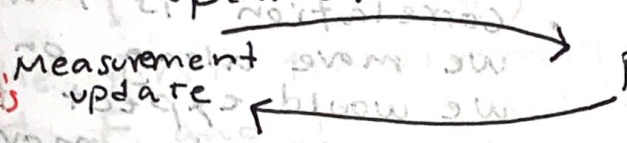
AI for robotics

In Kalman filters we have a Gaussian distribution.



$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\cdot\frac{(x-\mu)^2}{\sigma^2}}$$

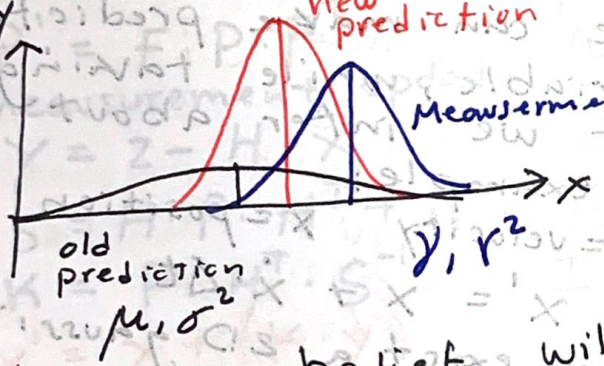
Similar to the Monte Carlo approach, in Kalman filters we have a measurement and motion update.



- Bayes Rule
- product
- Total probability
- convolution

Measurement update

To update our predictions we compute the new μ and σ for the distribution:



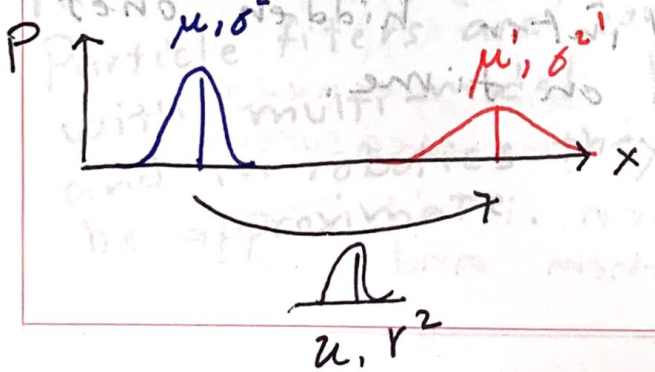
$$\mu' = \frac{r^2\mu + \sigma^2\nu}{r^2+\sigma^2} \quad \{\text{weighted sum}\}$$

$$\sigma'^2 = \frac{1}{\frac{1}{r^2}+\frac{1}{\sigma^2}} \quad \{\text{Harmonic Mean}\}$$

The new belief will be more certain than either the previous belief OR the measurement!

Motion update

When we move from point A to point B we will have lost some certainty, due to error in movement. This is represented by:



$$\mu' = \mu + u$$

$$\sigma'^2 = \sigma^2 + r^2$$

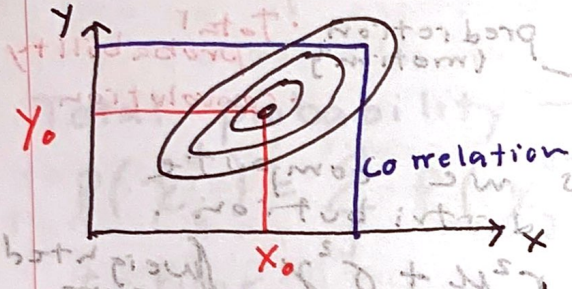
D 20 | M today | A

Multivariate Gaussians

In higher dimensional spaces ($S \geq \mathbb{R}^3$), we will have a vector for means, and a matrix for covariance.

$$\mu = \begin{pmatrix} \mu_0 \\ \vdots \\ \mu_0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \cdots \\ \vdots \\ \cdots \end{pmatrix}$$

where the Gaussian may look like:

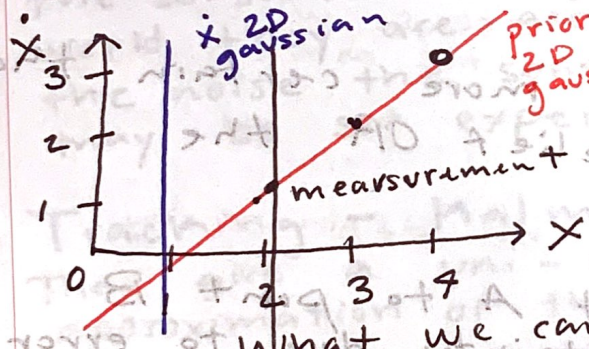


- correlation is that if we move along one dimension, we would expect the other dimension to move similarly. (along a curve of level of the surface or hyper-volume).

When we use higher dimensio Kalman filters we find that we can make predictions about an observable variable, while taking into consideration what we infer about another variable. For example:

$\dot{x}$ = velocity x = position

$$x' = x + \dot{x} \cdot \Delta t$$



- There exist a 2D gaussian that coorelates the measurements and the velocity (hidden state)

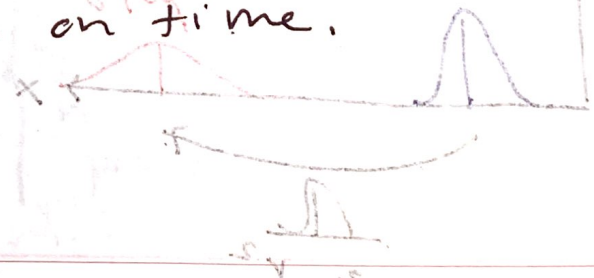
What we can understand is called:

Kalman filter states

there exist observable and hidden states.

Usually, from observation of observable states, we can predict/infer hidden ones.

- ex. predict velocity based on time.



AI for robotics

When we design Kalman filters we need two things one for the states and one the measurements. For the example of movement in 2D space:

→ Motion is ruled by $x' = x + \dot{x}$

$$\begin{pmatrix} x' \\ \dot{x}' \end{pmatrix} \xleftarrow{F} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \quad x' \leftarrow x + \dot{x}$$

$$z \xleftarrow{H} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \quad \dot{x}' \leftarrow \dot{x}$$

Where the first transformation matrix is called F and the transformation vector H. This is how the update is done in Kalman f.:

Prediction

$x' = Fx + u$

$P' = F \cdot P \cdot F^T$

measurement update

$Y = z - H \cdot x$

$S = H \cdot P \cdot H^T + R$

$K = P \cdot H^T \cdot S^{-1}$

$x' = x + (K \cdot y)$

$P' = (I - K \cdot H) \cdot P$

- x = estimate
- P = uncertainty covariance
- F = state transition matrix
- u = motion vector
- z = measurement
- H = measurement function
- R = measurement noise
- I = identity matrix

Notes: the "u" vector for motion is used when we may control the system, if not $u = 0$.

Particle filters

Particle fiters are continuous state space, with multi-modal belief, varying efficiency and in robotics they are considered to be approximate.

AI for robotics

Type	state space	Belief	Efficiency	In robotics
Histogram filters	Discrete	Muti-modal	exp.	approximate
Kalman filters	continuous	Uni...	x^2	approximate
Particle filters	continuous	Multi	?	approximate

Particle filters

The main idea is to have particles / points placed at random in the space, and have them represent a guess of where the robot is. Particles that are more consistent with the measurements "survive" and comprise the state prediction (this surviving or not is where the filter comes in).

Resampling

Resampling is what is done to replace and update the particles from one iteration to another. We have the following:

N particles $(\ldots,\ldots,\ldots)$ $(\ldots,\ldots,\ldots)$ $\vdots$ $(\ldots,\ldots,\ldots)$

weights: $w_1, w_2, \ldots, w_N$ — $W = \sum_i w_i$

normalized weight: $\alpha_1 = \frac{w_1}{W}$, $\alpha_2 = \frac{w_2}{W}$, $\ldots$, α_N — $\sum_i \alpha_i = 1$

N $\begin{bmatrix} \alpha_1 & \alpha_2 \\ \alpha_3 & \alpha_4 \\ \ldots & \end{bmatrix}$ — DRAW WITH REPLACEMENT $\Rightarrow$ N $\begin{bmatrix} p_2 & p_2 & p_5 \\ p_3 & p_4 & \end{bmatrix}$

We pick points at random, and because the points that best fit the measurements have higher weights, then it is more probable that we pick them and they 'survive'.

continues ...

continues

AI for robotics

It is important to note that even though orientation is not checked to determine the weights of each point or particle, it does matter, since the subsequent position depend on heading.

After all, the particle filter does:

Measurement updates

$$P(X|z) \overset{\text{resampling}}{\propto} P(z|X) \overset{\text{importance weights}}{} P(X) \overset{\text{particles}}{}$$

Motion updates

$$P(X') = \sum_{\text{samples}} P(X'|X) \overset{\text{sample}}{} P(X)$$

Note: A good heuristic to determine if a particle filter has enough / too many particles is to observe the weights. If they are high (non-normalized weights) you are probably tracking well and do not need many particles. the contrary is also true.

Motion planning

Search

A* is one of the most popular and efficient search algorithms. It uses an f-value instead of the normal g-values, while still performing the same kind of search.

$$f = g + h(x,y)$$

where $h(x,y)$ may be the function to calculate the Euclidean distance to the point.