JC3504 Robot Technology

Lecture 11: Localisation

Dr Xiao Li xiao.li@abdn.ac.uk

Dr Junfeng Gao junfeng.gao@abdn.ac.uk

Outline

Markov Localisation

The Bayes Filter

Particle Filter

Kalman Filter

- Kalman Filter 1-D Example
- Kalman Filter (Simplified)
- Kalman Filter (full-version)

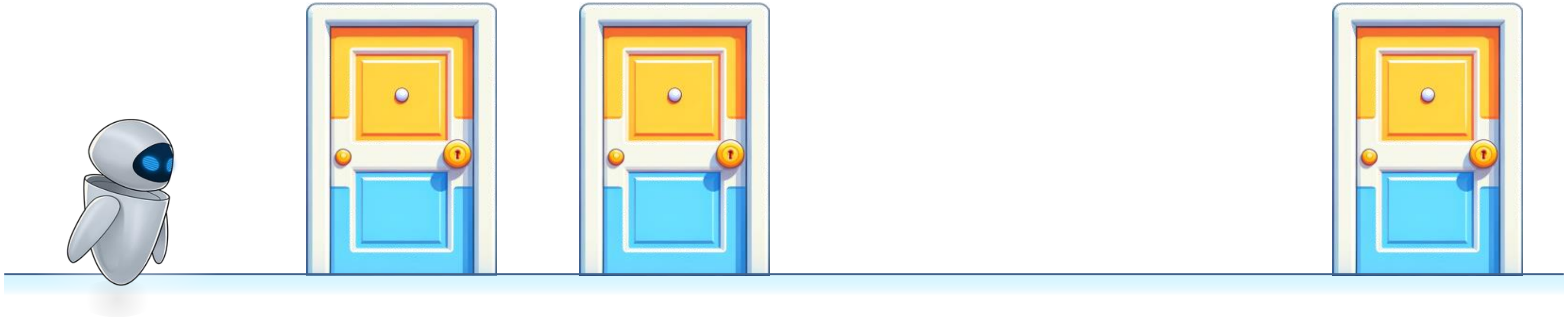
Localisation

Localisation in robotics involves the determination of a robot's position and orientation within its environment, utilising various sensors and algorithms to enable accurate navigation and interaction with surroundings.

The primary challenge of localisation in robotics lies in achieving accurate and robust position estimation amidst dynamic environments, sensor noise, and the inherent uncertainty of real-world settings.

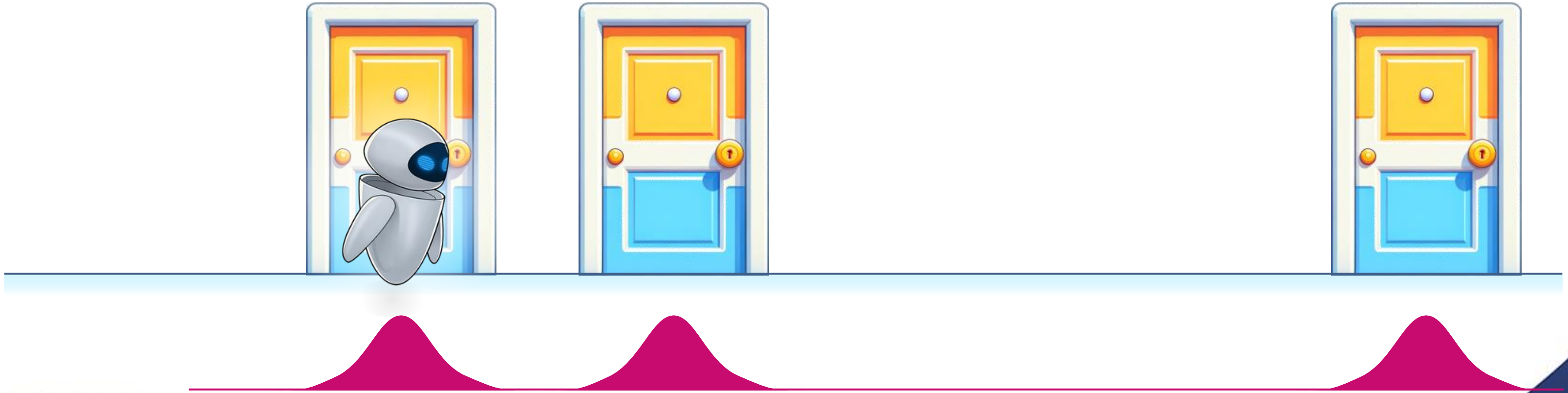
Motivating Example

Imagine a floor with three doors, two of which are closer together, and the third farther down the corridor. Imagine now that your robot is able to detect doors i.e. it is able to tell whether it is in front of a wall or in front of a door. Features like this can serve as a landmark for the robot.



Motivating Example

Given the map of the environment but no information about the robot's location, we can use landmarks to drastically reduce the space of possible locations once the robot has passed one of the doors. When a robot passes through a door, it cannot ascertain which door it has passed through, but it knows it is in front of a door; hence, its estimation of its own position is as follows.



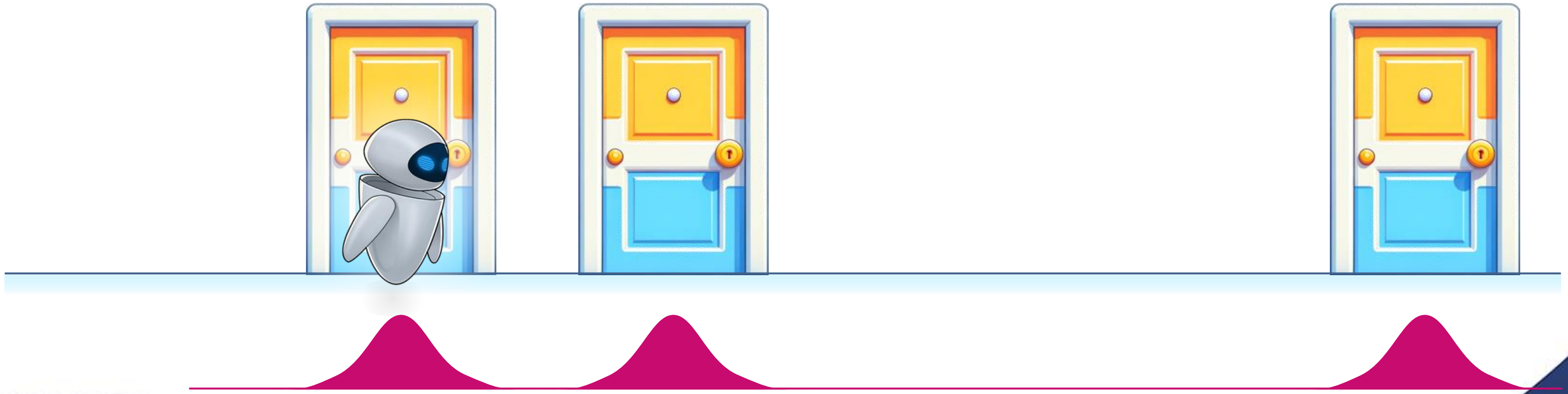
The location possibility distribution

UNIVERSITY OF ABERDEEN

Motivating Example

One way of representing this belief is to describe the robot's position with three Gaussian distributions, each centred in front of a door and its variance a function of the uncertainty with which the robot can detect a door's centre.

This is known as a multi-hypothesis belief, since we have a hypothesis stating that the robot can be in front of each door.

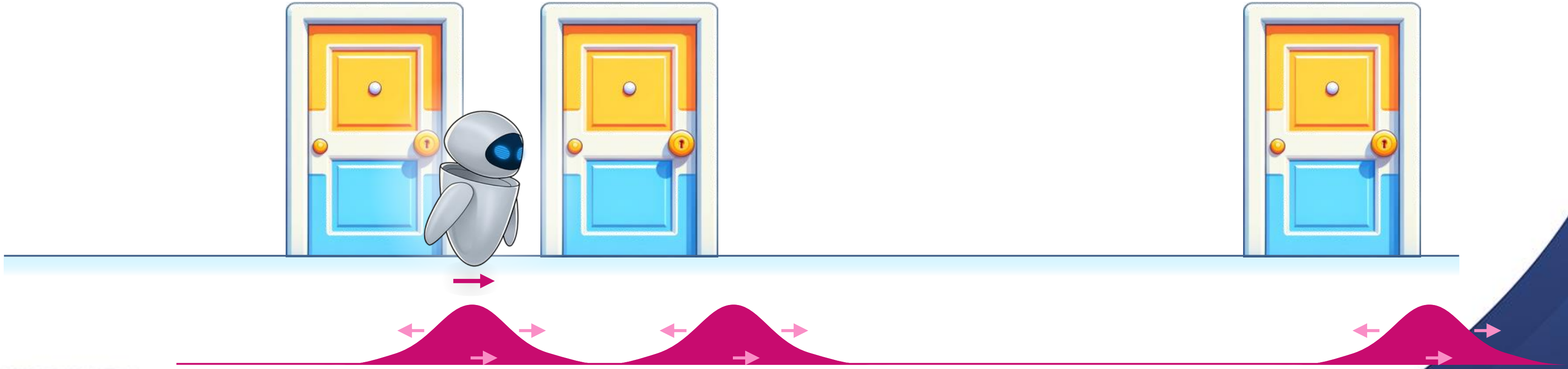


The location possibility distribution

Motivating Example

If the robot continues to move, according to the error propagation law:

1. The Gaussians describing the robot’s 3 possible locations will move with the robot.
2. The variance of each Gaussian will keep increasing with the distance the robot moves.

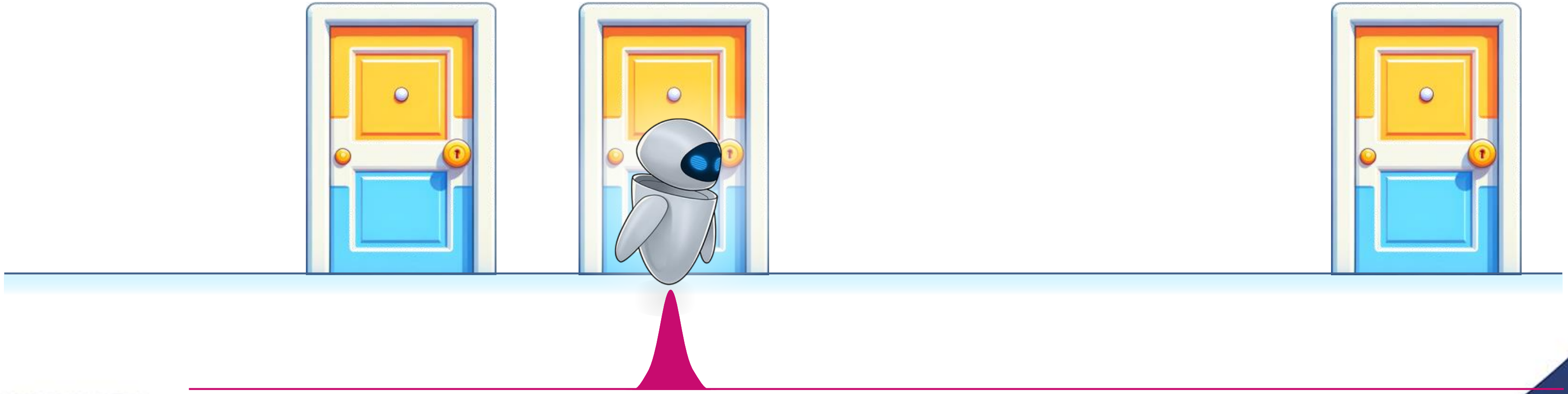


The location possibility distribution

Motivating Example

Assuming we trust our door detector much more than our odometry estimate, we can now remove all beliefs that do not coincide with a door.

Again assuming our door detector can detect the centre of a door with some accuracy, our location estimate's uncertainty is now only limited by that of the door detector.



The location possibility distribution

Markov Localisation

Calculating the probability to be at a certain location given the likelihood of certain observations is the same as any other conditional probability. There is a formal way to describe such situations: Bayes' Rule.

Initially, we have no knowledge of the position, so for the robot, the location follows a uniform distribution. We use $P(loc_0)$ represents the location distribution (we regard the space as a 1-dimention space).

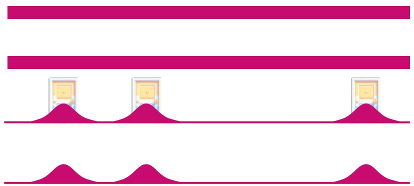
The location possibility distribution $(P(loc_0))$ is a uniform distribution)

Markov Localisation

Then, the sensor detects a door. So $P(loc_0)$ is updated with $P(loc_0|door)$. According to Bayes' Rule,

$$P(loc_0|door) = \frac{P(loc_0)P(door|loc_0)}{P(door)}$$

$$P(loc_0) =$$
$$P(door) =$$
$$P(door|loc_0) =$$
$$P(loc_0|door) =$$



The location possibility distribution

Markov Localisation

Then, the sensor detects a door. So $P(loc_0)$ is updated with $P(loc_0|door)$. According to Bayes' Rule,

$$P(loc_0|door) = \frac{P(loc_0)P(door|loc_0)}{P(door)}$$

prior probability

posterior probability

The location possibility distribution

Markov Localisation

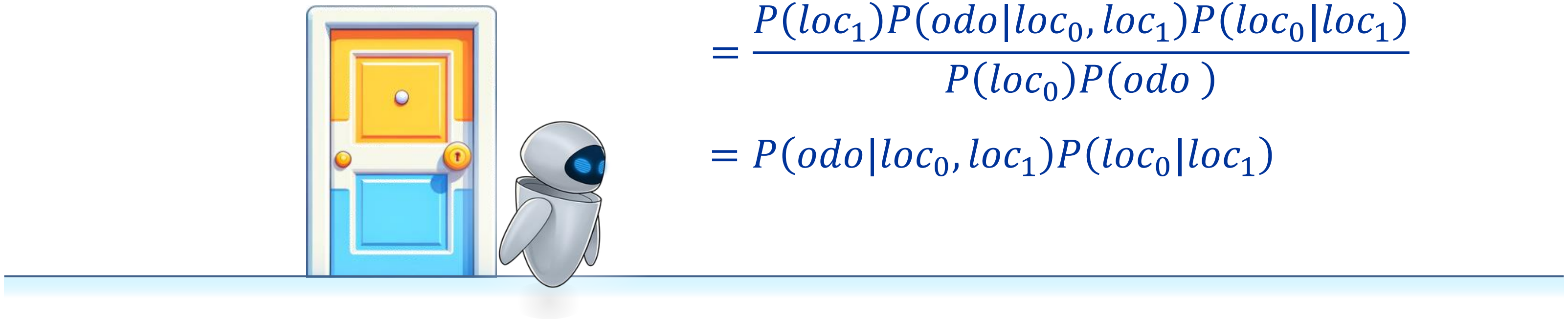
When the robot moves, $loc_0 \rightarrow loc_1$, the distance (odo) can be obtained from the odometer.

Assume we want to move to loc_1, but there is an error in the odometer. So,

$$P(loc_1|odo, loc_0) = \frac{P(loc_1)P(odo, loc_0|loc_1)}{P(odo, loc_0)}$$

$$= \frac{P(loc_1)P(odo|loc_0, loc_1)P(loc_0|loc_1)}{P(loc_0)P(odo\ )}$$

$$= P(odo|loc_0, loc_1)P(loc_0|loc_1)$$



The location possibility distribution

Markov Localisation

When the robot moves, $loc_0 \rightarrow loc_1$, the distance (odo) can be obtained from the odometer.

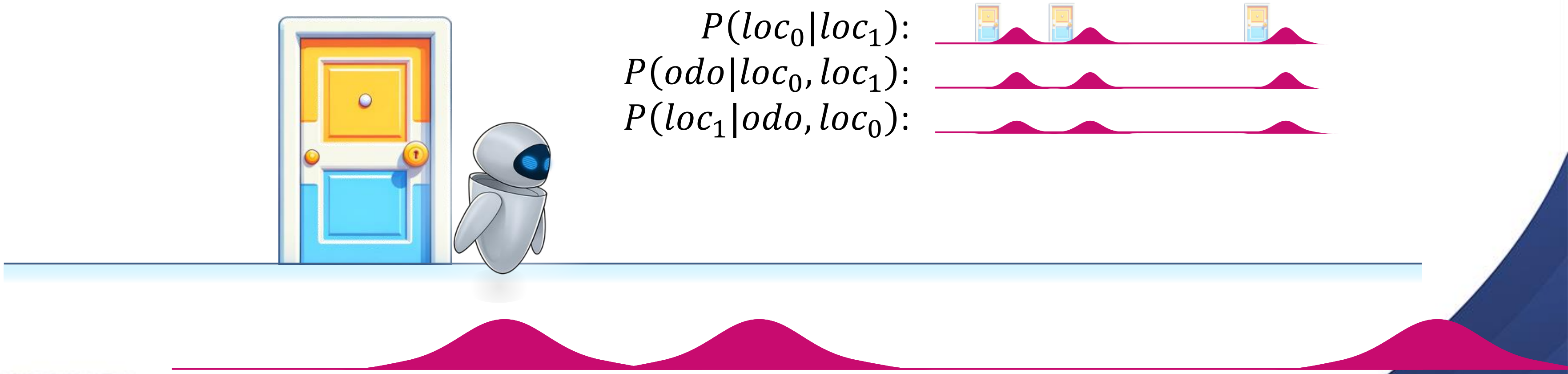
Assume we want to move to loc_1, but there is an error in the odometer. So,

$$P(loc_1|odo, loc_0) = P(odo|loc_0, loc_1)P(loc_0|loc_1)$$

$P(loc_0|loc_1)$:

$P(odo|loc_0, loc_1)$:

$P(loc_1|odo, loc_0)$:



The location possibility distribution

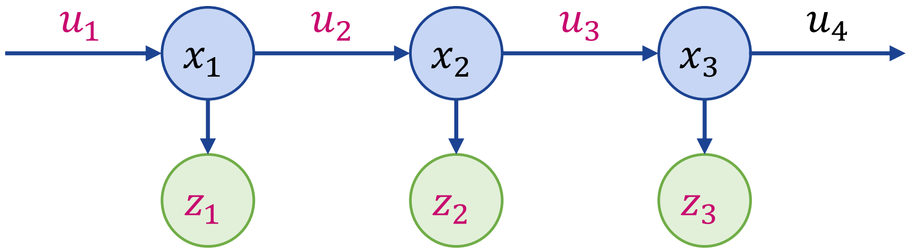
The Bayes Filter

The Bayes Filter

A Bayes Filter is a probabilistic tool for estimating the state of a system (e.g. robot location) over time, updating predictions with new evidence (e.g. sensor signal) using Bayesian inference.

To formalize our terms and notation, we will describe our robot's motion model as the distribution given by $P(x_i|x_{i-1}, u_i)$ that is, the probability of being in a particular state x_i given that we started in state x_{i-1} and executed action u_i.

We can describe our sensor model as being characterized by the distribution given by $P(z_i|x_i)$, namely the probability that we would see sensor observation z_i if we were in state x_i .



UNIVERSITY OF ABERDEEN

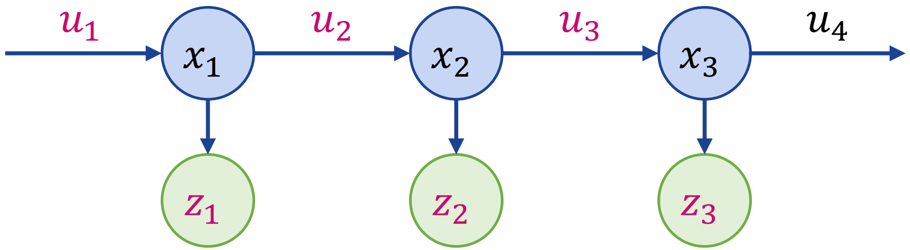
The Bayes Filter

Our goal with the Bayes filter will be to estimate our robot's state over time (x_t, where t indicates timestep) given a history of actions and observations (sensor measurements).

We compute the belief i.e. the posterior probability of our state estimate ($bel(x_t)$), given a history of actions ($u_1, \dots, u_t$) and sensor measurements ($z_1, \dots, z_t$) as:

$$bel(x_t) = P(x_t | u_1, \dots, u_t, z_1, \dots, z_t)$$

all the historical actions and observations



The Bayes Filter

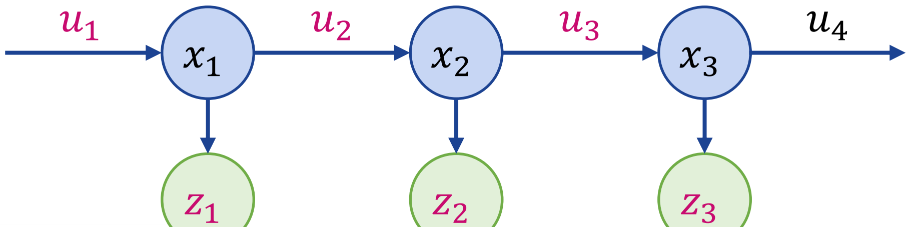
Always using all the historical data will lead to the consumption of increasing computational resources. Therefore, we want to calculate $bel(x_t)$ based on the information of x_t.

$$bel(x_t) = P(x_t|u_1, \dots, u_t, z_1, \dots, z_t)$$
$$= P(z_t|x_t, u_1, \dots, u_t, z_1, \dots, z_{t-1})P(x_t|u_1, \dots, u_t, z_1, \dots, z_{t-1})/P(z_t|u_1, \dots, u_t, z_1, \dots, z_{t-1})$$

We assume z_t only depends on x_t (Markov assumption), then:

- $P(z_t|u_1, \dots, u_t, z_1, \dots, z_{t-1})$ can be regarded as a constant, c.
- $P(z_t|x_t, u_1, \dots, u_t, z_1, \dots, z_{t-1}) = P(z_t|x_t)$

They give: $c \cdot bel(x_t) = P(z_t|x_t)P(x_t|u_1, \dots, u_t, z_1, \dots, z_{t-1})$



The Bayes Filter

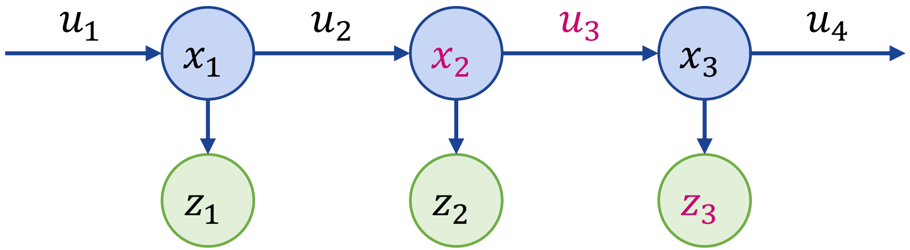
$$c \cdot bel(x_t) = P(z_t|x_t)P(x_t|u_1, \dots, u_t, z_1, \dots, z_{t-1})$$

$$c \cdot bel(x_t) = P(z_t|x_t)\sum_{x_{t-1} \in X_{t-1}} P(x_t|u_t, x_{t-1})P(x_{t-1}|u_1, \dots, u_t, z_1, \dots, z_{t-1})$$

$$c \cdot bel(x_t) = P(z_t|x_t)\sum_{x_{t-1} \in X_{t-1}} P(x_t|u_t, x_{t-1})P(x_{t-1}|u_1, \dots, u_{t-1}, z_1, \dots, z_{t-1})$$

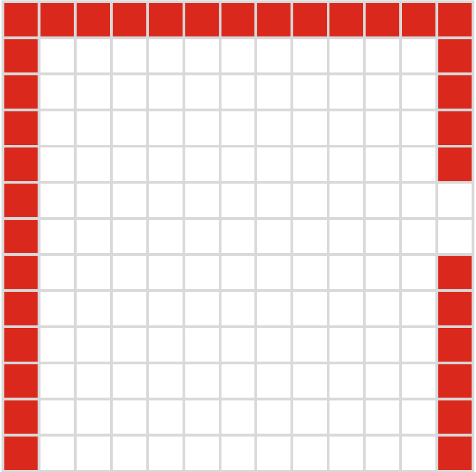
$$c \cdot bel(x_t) = P(z_t|x_t)\sum_{x_{t-1} \in X_{t-1}} P(x_t|u_t, x_{t-1})bel(x_{t-1})$$

This final equation is remarkable because it allows us to perform a belief update for a given state by incorporating a sensor measurement and/or a motion prediction based on an action we took.



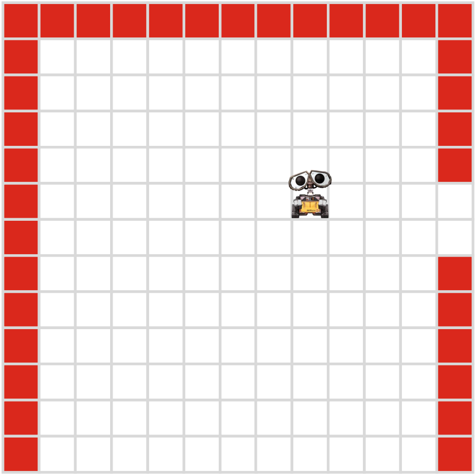
Example: Bayes Filter on a Grid

Instead of using a coarse topological map, we can model the environment as a fine-grained grid. We assume that the robot is able to detect walls around it with short-range sensors.

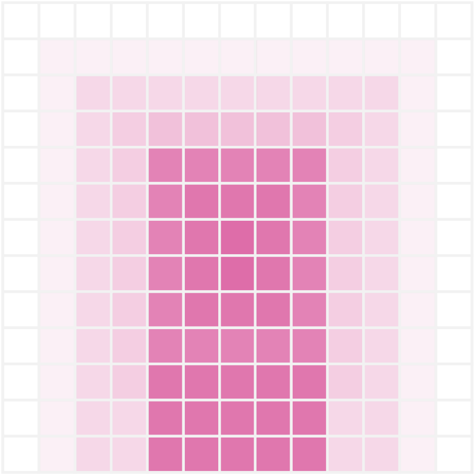


Example: Bayes Filter on a Grid

Initially, the robot does not see a wall and therefore could be almost anywhere.



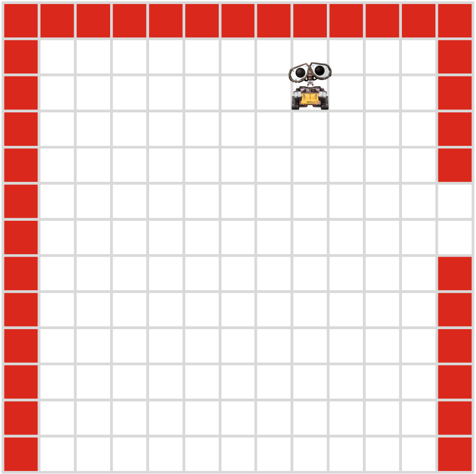
real position



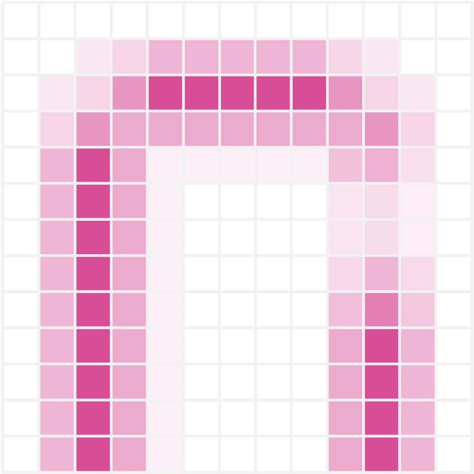
estimated position

Example: Bayes Filter on a Grid

The robot now moves. As soon as the robot encounters the wall, the perception update bumps up the likelihood to be higher in grid cells near walls.



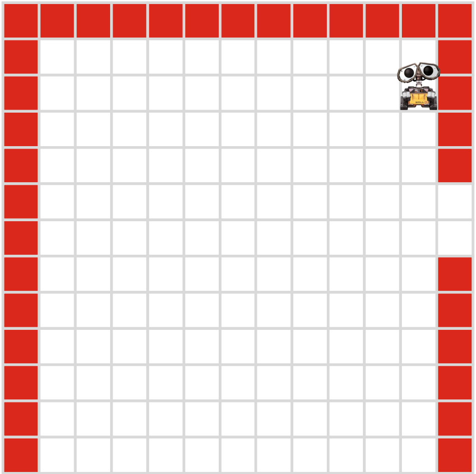
real position



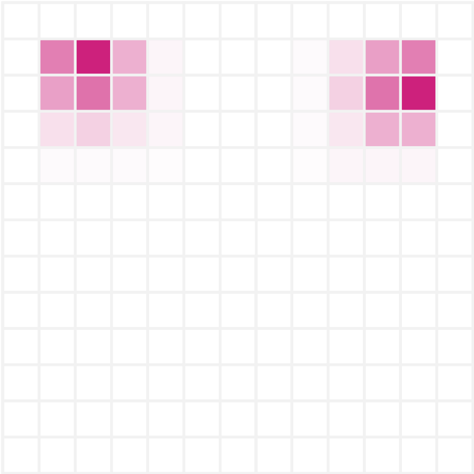
estimated position

Example: Bayes Filter on a Grid

The robot then performs a right turn and travels along the wall until hits the wall again.



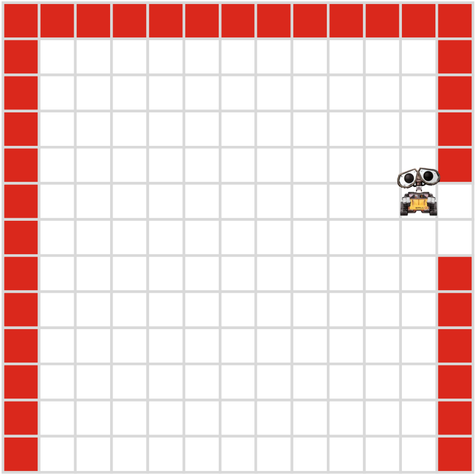
real position



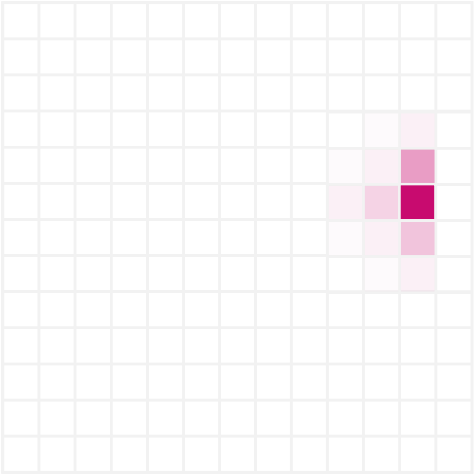
estimated position

Example: Bayes Filter on a Grid

The robot continuously travels along. when it reaches the gap in the wall, it is almost certain about its position.



real position



estimated position

Particle Filter

UNIVERSITY OF ABERDEEN

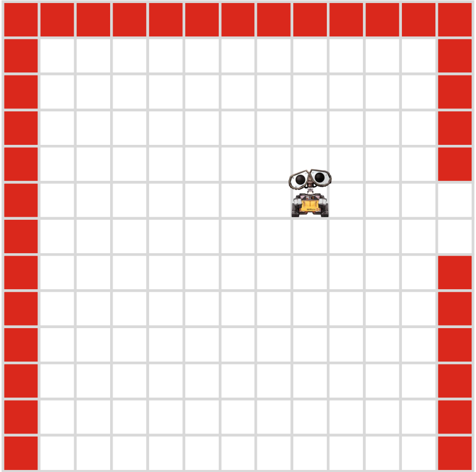
Particle Filter

Although grid-based Markov Localisation can provide compelling results, it can be computationally very expensive, particularly when the environment is large, and the resolution of the grid is small. This is in part due to the fact that we need to carry the probability to be at a certain location forward for every cell on the grid, regardless of how small this probability is.

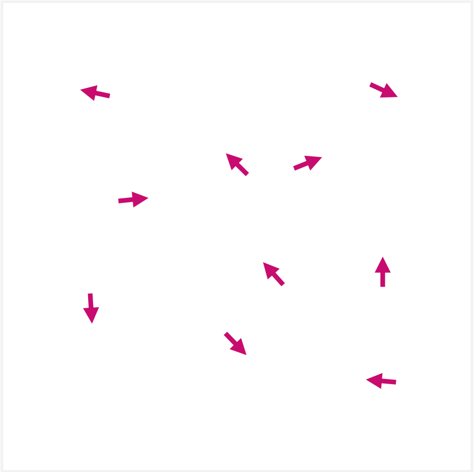
An elegant solution to this problem is the particle filter.

Particle Filter

Initially, we generate a set of particles, each represents a possible initial state of the robot.



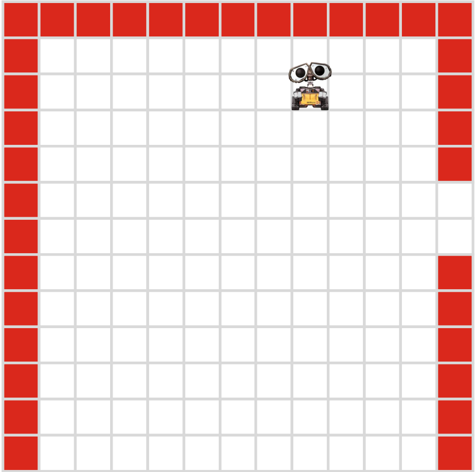
real position



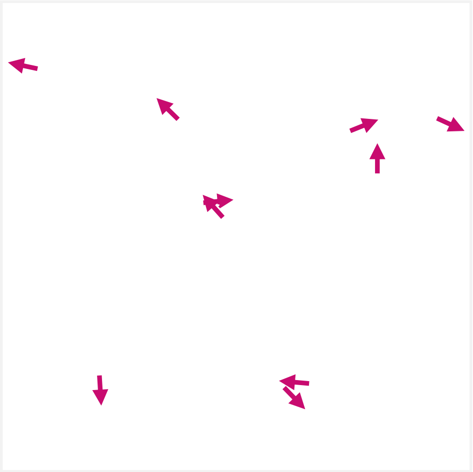
estimated positions and orientations

Particle Filter

Every time the robot moves, we will move each particle in the exact same way, but add noise to each movement much like we would observe on the real robot. Without a perception update, the particles will spread apart farther and farther.



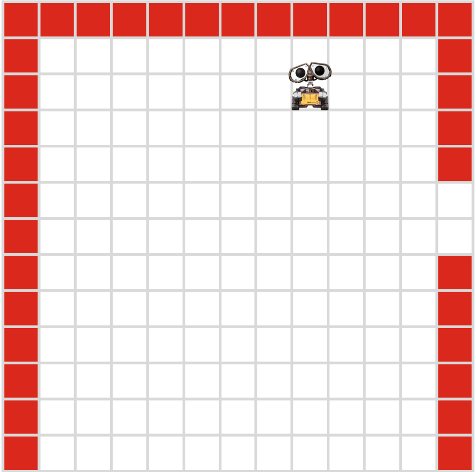
real position



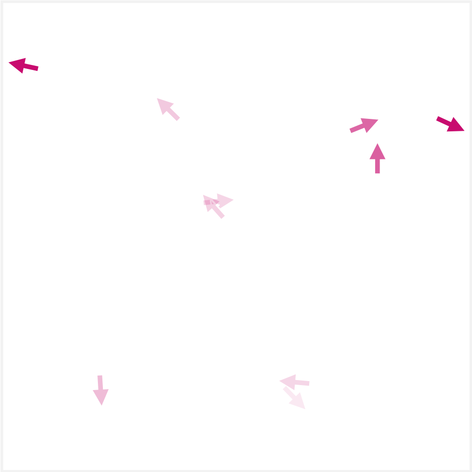
estimated positions and orientations

Particle Filter

When there is new observation data, each particle is given a weight, which reflects the degree to which the state of the particle is consistent with the observation data.



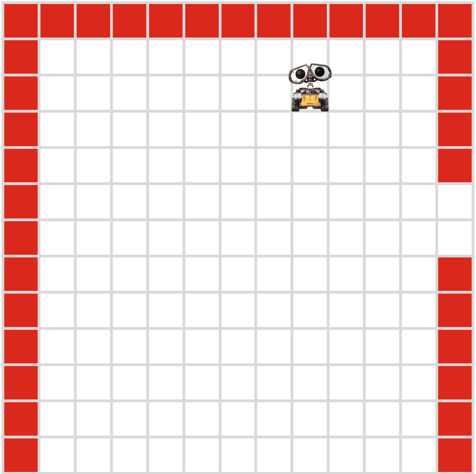
real position



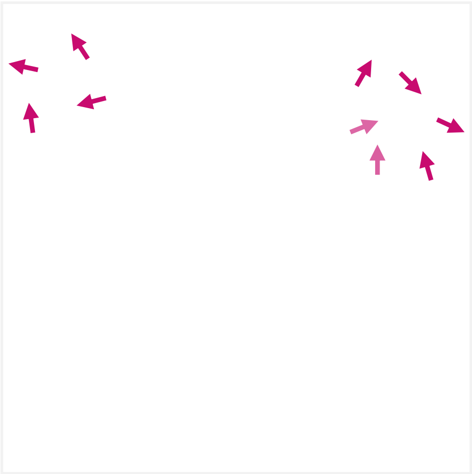
estimated positions and orientations

Particle Filter

Particles with low weights will be eliminated and particles with high weights will be copied.



real position

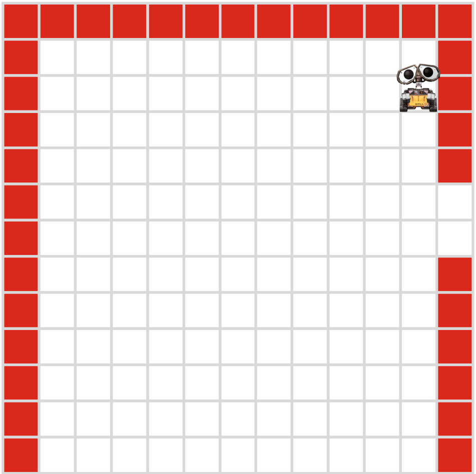


estimated positions and orientations

UNIVERSITY OF ABERDEEN

Particle Filter

We repeat this step

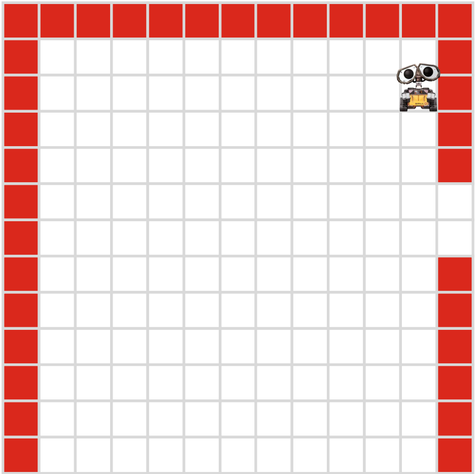


real position

estimated positions and orientations

Particle Filter

We repeat this step

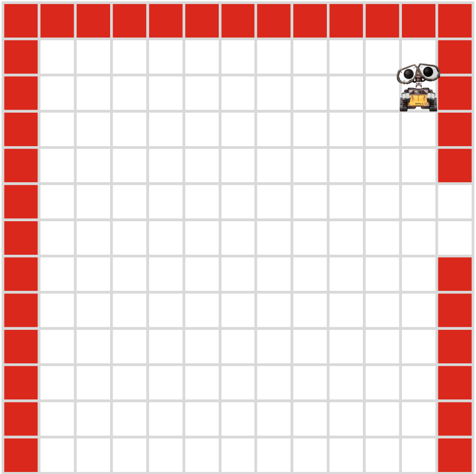


real position

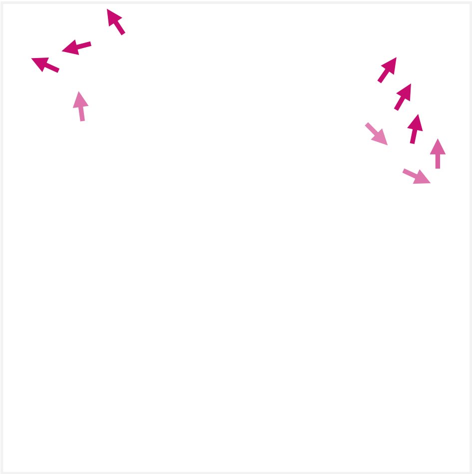
estimated positions and orientations

Particle Filter

We repeat this step



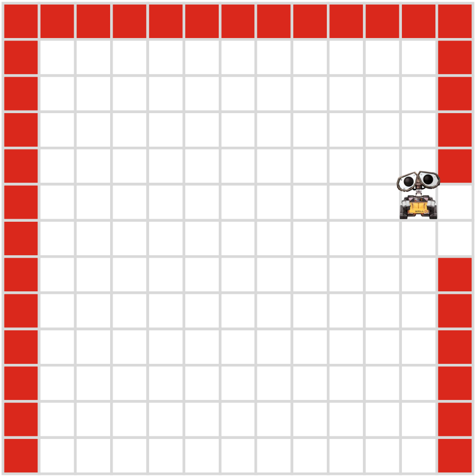
real position



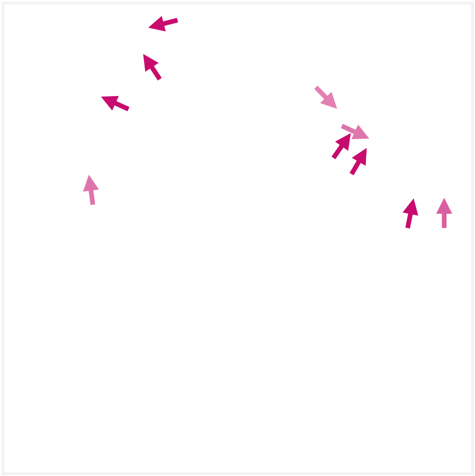
estimated positions and orientations

Particle Filter

We repeat this step



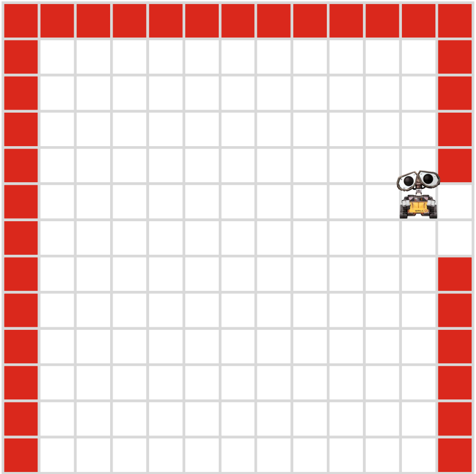
real position



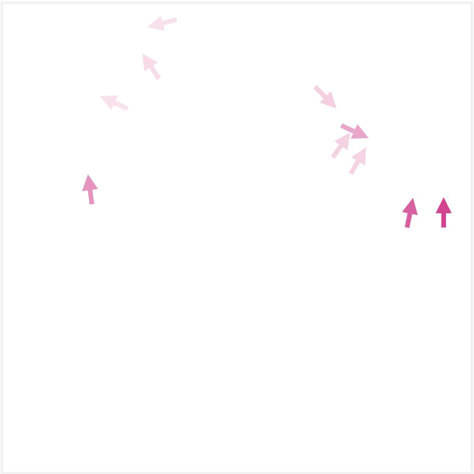
estimated positions and orientations

Particle Filter

We repeat this step



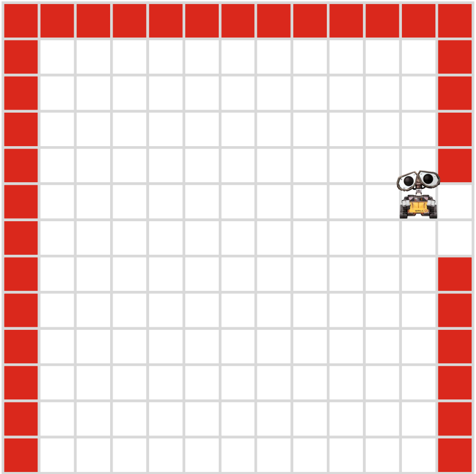
real position



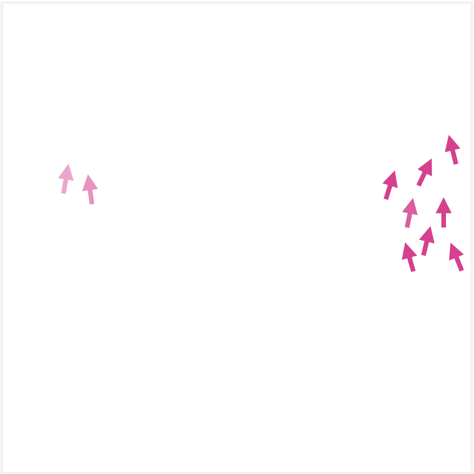
estimated positions and orientations

Particle Filter

After a few more observations, most examples will be located near the correct location of the robot.



real position



estimated positions and orientations

Kalman Filter

Kalman Filter

Kalman Filter is an efficient recursive filtering algorithm used for state estimation and control of linear systems.

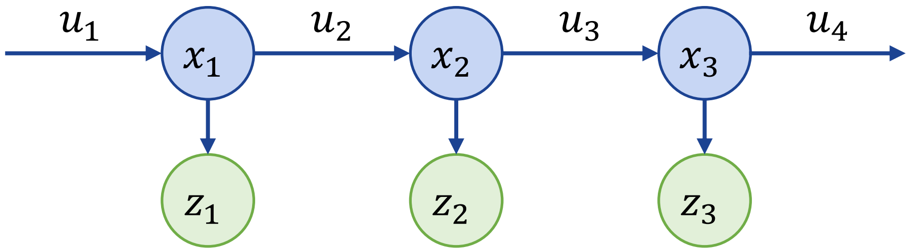
The Kalman filter uses a dynamic model of the system, combined with past states and current observations, to predict the current state and update the prediction to more accurately reflect the real situation.

Kalman Filter

The action prediction step looks as follows:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_k)$$

Here, $f()$ denote the Kalman Filter predicting process, $\hat{x}_{k-1}$ denotes the previous state, and u_k is the control input.



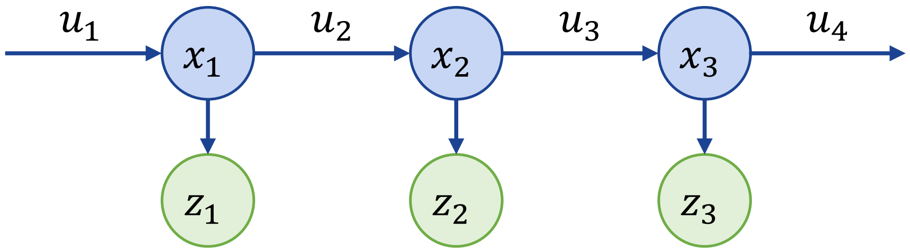
Kalman Filter **1-D** Example

We only consider the x position of a robot (denoted by x_k), so u_k denotes the forward distance, and z_k denotes the odometer reading.

Given $\hat{x}_{k-1}$ (the prediction for x_{k-1}), u_k and z_k, Kalman Filter will find $\hat{x}_k$.

If there is no error/noise, we would have

$$\hat{x}_k \equiv x_k \equiv z_k = \hat{x}_{k-1} + u_k$$



Kalman Filter **1-D** Example

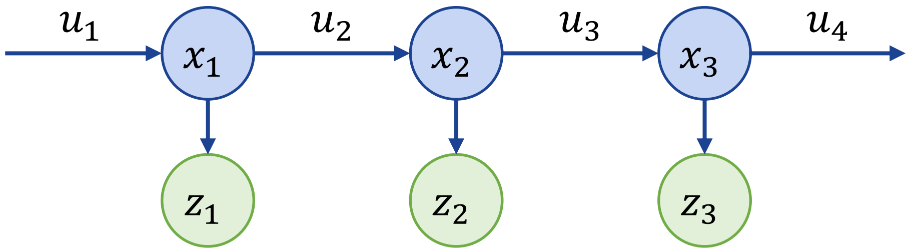
Here, we want to consider the generalised situation

1. The state of the robot (i.e. x) may change because of itself e.g. The robot will always turn around automatically, so we use f to denote the state change.
2. x and u may have different dimensions e.g. x in *km*, u in *m*. we use b to unify them.

They give:

$$\hat{x}_k \equiv x_k \equiv z_k = f \cdot \hat{x}_{k-1} + b \cdot u_k$$

If there is no error/noise, the prediction is very simple.



Kalman Filter **1-D** Example

When we consider the errors, $\hat{x}_k$, x_k, and z_k will differ, so that only

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$

We temporally use $\hat{x}_{k-}$ denotes $\hat{x}_k$ since we haven't taken z_k into account yet.

Because the errors exist, Kalman Filter also predict the variance of $\hat{x}_k$, denoted by p_k.

$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Because $p_{k-1} = \frac{1}{n}\sum_{i=0}^{n}\left(\hat{x}_{k-1,i} - \overline{\hat{x}_{k-1}}\right)^2$, so

$$\frac{1}{n}\sum_{i=0}^{n}\left(f \cdot \hat{x}_{k-1,i} - f \cdot \overline{\hat{x}_{k-1}}\right)^2 = \frac{1}{n}\sum_{i=0}^{n}\left(f \cdot \left(\hat{x}_{k-1,i} - \overline{\hat{x}_{k-1}}\right)\right)^2$$

$$= \frac{1}{n}\sum_{i=0}^{n} f^2\left(\hat{x}_{k-1,i} - \overline{\hat{x}_{k-1}}\right)^2 = f^2 \cdot \frac{1}{n}\sum_{i=0}^{n}\left(\hat{x}_{k-1,i} - \overline{\hat{x}_{k-1}}\right)^2 = f^2 \cdot p_{k-1}$$

Kalman Filter **1-D** Example

When we consider the errors, $\hat{x}_k$, x_k, and z_k will differ, so that only

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$

We temporally use $\hat{x}_{k-}$ denotes $\hat{x}_k$ since we haven't taken z_k into account yet.

Because the errors exist, Kalman Filter also predict the variance of $\hat{x}_k$, denoted by p_k.

$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

q_k donotes the process noise. It encapsulates how much uncertainty we believe exists in the dynamics of the system being modelled, between state transitions.

Again, we temporally use p_{k-} denotes p_k since we haven't taken z_k into account yet.

UNIVERSITY OF ABERDEEN

Kalman Filter **1-D** Example

Now, we want to use z_k to correct $\hat{x}_{k-}$. When the values of z_k and $\hat{x}_{k-}$ are different, we need to balance who to believe more. Kalman, therefore, introduces a rate parameter λ_k to deal with the balance, such that:

$$\hat{x}_k = \lambda_k z_k + (1 - \lambda_k)\hat{x}_{k-} = \hat{x}_{k-} + \lambda_k(z_k - \hat{x}_{k-})$$

λ_k represents the degree of credibility of $\hat{x}_{k-}$, which is estimated from the variance of $\hat{x}_{k-}$.

If the variance of $\hat{x}_{k-}$ is large (i.e. p_{k-} is large), the degree should be low, vice versa.

1495 UNIVERSITY OF ABERDEEN

Kalman Filter **1-D** Example

Kalman use the following formulation to calculate λ_k, where r_k is the intensity of noise :

$$\lambda_k = \frac{p_{k-}}{p_{k-} + r_k}$$

r_k is the measurement noise. It represents the uncertainty or the "noise" in the measurements observed from the system.

Because $\hat{x}_k = \lambda_k z_k + (1 - \lambda_k)\hat{x}_{k-}$, the term $(1 - \lambda_k)$ reduces the uncertainty inherent in $\hat{x}_{k-}$ when it passed to $\hat{x}_k$, thereby also reducing the variance of $\hat{x}_k$:

$$p_k = (1 - \lambda_k)p_{k-}$$

Kalman Filter **1-D** Example

In summary, given $\hat{x}_{k-1}$, u_k, and z_k, Kalman Filter can be regards as a two-step process.

1. Prediction step predicts the next state $\hat{x}_{k-}$ as well as the variation p_{k-}.
2. Updating step corrects the prediction according to the observation z_k.

So far, we introduced the Kalman Filter for single state variable (e.g. x in the example).

State $k-1$

$\hat{x}_{k-1}$

p_{k-1}

State k

Prediction

u_k

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$

$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Updating

z_k

$$\lambda_k = p_{k-} \,/\, (p_{k-} + r_k)$$

$$\hat{x}_k = \hat{x}_{k-} + \lambda_k (z_k - \hat{x}_{k-})$$

$$p_k = (1 - \lambda_k) p_{k-}$$

$\hat{x}_k$

p_k

State $k+1$

Kalman Filter (Simplified)

If there are multiple state variables, e.g., position x and velocity v, and the variables are not independent of each other, we need to consider their covariance.

NB: If the variables are independent, we just need to calculate them separately.

The covariance is represented as matrix, so the formulars of the 1-D case need to be revised.

Kalman Filter (Simplified)

First, $\hat{x}_{k-1}$ will be a vector.

$$\hat{x}_{k-1} = \begin{bmatrix} x_{k-1} \\ v_{k-1} \end{bmatrix}$$

u_k may also include multiple variables, so a vector again.

Therefore, f and b become matrix, to be clear, we replace them with F and B .

$$F: \{State\} \rightarrow \{State\}$$
$$B: \{Control\} \rightarrow \{State\}$$

They give,

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$

For example,

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \Longrightarrow \begin{cases} x_k = x_{k-1} + v_{k-1} \\ v_k = v_{k-1} \end{cases}$$

1-D example

Prediction

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$
$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Updating

$$\lambda_k = p_{k-} \,/\, (p_{k-} + r_k)$$
$$\hat{x}_k = \hat{x}_{k-} + \lambda_k(z_k - \hat{x}_{k-})$$
$$p_k = (1 - \lambda_k)p_{k-}$$

Kalman Filter (Simplified)

p_{k-1} becomes the covariance matrix P_{k-1}.

Because of the covariance

$$\Sigma = cov(X) = \mathrm{E}\left((\mathrm{E}(X) - X)(X - \mathrm{E}(X))^T\right)$$
$$cov(FX) = F\Sigma F^T$$

It gives:

$$P_{k-} = FP_{k-1}F^T + Q_k$$

Q_k is the matrix version noises.

1-D example

Prediction

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$
$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Updating

$$\lambda_k = p_{k-} \,/\, (p_{k-} + r_k)$$
$$\hat{x}_k = \hat{x}_{k-} + \lambda_k(z_k - \hat{x}_{k-})$$
$$p_k = (1 - \lambda_k)p_{k-}$$

Kalman Filter (Simplified)

λ_k becomes a matrix Λ_k, because

$$\Lambda_k = \frac{P_{k-}}{(P_{k-} + R_k)} = P_{k-}(P_{k-} + R_k)^{-1}$$

R_k is the matrix version noises.

Therefore,

$$\hat{x}_k = \hat{x}_{k-} + \Lambda_k(z_k - \hat{x}_{k-})$$

$$P_k = (I - \Lambda_k)P_{k-}$$

1-D example

Prediction

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$

$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Updating

$$\lambda_k = p_{k-} \,/\, (p_{k-} + r_k)$$

$$\hat{x}_k = \hat{x}_{k-} + \lambda_k(z_k - \hat{x}_{k-})$$

$$p_k = (1 - \lambda_k)p_{k-}$$

Kalman Filter (Simplified)

Then we got the matrix version of Kalman Filter

Prediction

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$

$$P_{k-} = FP_{k-1}F^T + Q_k$$

Updating

$$\Lambda_k = P_{k-}(P_{k-} + R_k)^{-1}$$

$$\hat{x}_k = \hat{x}_{k-} + \Lambda_k(z_k - \hat{x}_{k-})$$

$$P_k = (I - \Lambda_k)P_{k-}$$

Kalman Filter (full-version)

There is one thing remaining. So far, we suppose that z_k directly reflexes all the variables of x_k, so we can produce $z_k - \hat{x}_{k-}$. However, in the real-life cases, z_k may contain patricidal data of $\hat{x}_{k-}$ (e.g. $\hat{x}_{k-}$ contains velocity, but z_k not).

The full-version Kalman Filter use one more matrix H_k (called the measurement matrix) to project vectors in $\{State\}$ space to $\{Measur\}$ space.

So that $\hat{x}_k = \hat{x}_{k-} + \Lambda_k(z_k - \hat{x}_{k-})$ becomes:

$$H_k\hat{x}_k = H_k\hat{x}_{k-} + \Lambda'_k(z_k - H_k\hat{x}_{k-})$$

Here, Λ_k is also replaced by Λ'_k, because now, $(z_k - H_k\hat{x}_{k-})$ is in $\{Measur\}$ no longer in $\{State\}$.

NB: we cannot define a matrix which maps $\{Measur\}$ to $\{State\}$, because $\{State\}$ includes more dimension than $\{Measur\}$.

1495 UNIVERSITY OF ABERDEEN

Kalman Filter (full-version)

In $\Lambda_k = P_{k-}(P_{k-} + R_k)^{-1}$, $\Lambda_k: \{State\} \rightarrow \{State\}$, but $\Lambda'_k: \{Measur\} \rightarrow \{Measur\}$. Recall that P_{k-} is the covariance matrix, so we can use H_k to project P_{k-} to $\{State\}$ by

$$H_k P_{k-} H_k^T$$

It gives:

$$\Lambda'_k = H_k P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1}$$

NB: R_k is measurement noise covariance matrix, so it's already for $\{Measur\}$.

Then, we plug Λ'_k in $H_k \hat{x}_k = H_k \hat{x}_{k-} + \Lambda'_k (z_k - H_k \hat{x}_{k-})$:

$$H_k \hat{x}_k = H_k \hat{x}_{k-} + H_k P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1} (z_k - H_k \hat{x}_{k-})$$

By applying a pseudoinverse of H_k, we can cancel the three H_k. It gives:

$$\hat{x}_k = \hat{x}_{k-} + P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1} (z_k - H_k \hat{x}_{k-})$$

Kalman Filter (full-version)

We can use the same way to replace Λ_k with Λ'_k from $P_k = (I - \Lambda_k)P_{k-}$:

$$P_k = (I - \Lambda_k)P_{k-} = p_{k-} - \Lambda_k P_{k-}$$

$$H_k P_k H_k^T = H_k P_{k-} H_k^T - \Lambda'_k H_k P_{k-} H_k^T$$

$$H_k P_k H_k^T = H_k p_{k-} H_k^T - H_k P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1} H_k P_{k-} H_k^T$$

Then we cancel the left H_k and right H_k^T of each term:

$$P_k = P_{k-} - P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1} H_k P_{k-} = \left(I - P_{k-} H_k^T \left(H_k P_{k-} H_k^T + R_k\right)^{-1} H_k\right) P_{k-}$$

Kalman Filter (full-version)

Finally, the full-version with considering the measurement matrix:

Prediction

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$
$$P_{k-} = FP_{k-1}F^T + Q_k$$

Updating

$$\hat{x}_k = \hat{x}_{k-} + P_{k-}\, H_k^T \left(H_k\, P_{k-}\, H_k^T + R_k\right)^{-1} (z_k - H_k\, \hat{x}_{k-})$$
$$P_k = \left(I - P_{k-}H_k^T\left(H_kP_{k-}H_k^T + R_k\right)^{-1}H_k\right)P_{k-}$$

Usually, $P_{k-}\, H_k^T \left(H_k\, P_{k-}\, H_k^T + R_k\right)^{-1}$ is called Kalman gain denoted by K_k. So,

Kalman Filter (full-version)

Finally, the full-version with considering the measurement matrix:

Prediction

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$

$$P_{k-} = FP_{k-1}F^T + Q_k$$

Updating

$$K_k = P_{k-}\, H_k^T \left(H_k\, P_{k-}\, H_k^T + R_k\right)^{-1}$$

$$\hat{x}_k = \hat{x}_{k-} + K_k(z_k - H_k\, \hat{x}_{k-})$$

$$P_k = (I - K_k H_k)P_{k-}$$

Kalman Filter

In summary, we started from the 1-D case, then, derived the 1-D case to n-D version. Finally, we introduced the measurement matrix to obtain the common version of Kalman Filter.

1-D version

Prediction

$$\hat{x}_{k-} = f \cdot \hat{x}_{k-1} + b \cdot u_k$$
$$p_{k-} = f^2 \cdot p_{k-1} + q_k$$

Updating

$$\lambda_k = p_{k-} \,/\, (p_{k-} + r_k)$$
$$\hat{x}_k = \hat{x}_{k-} + \lambda_k(z_k - \hat{x}_{k-})$$
$$p_k = (1 - \lambda_k)p_{k-}$$

n-D version

Prediction

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$
$$P_{k-} = FP_{k-1}F^T + Q_k$$

Updating

$$\Lambda_k = P_{k-}(P_{k-} + R_k)^{-1}$$
$$\hat{x}_k = \hat{x}_{k-} + \Lambda_k(z_k - \hat{x}_{k-})$$
$$P_k = (I - \Lambda_k)p_{k-}$$

Kalman Filter

Prediction

$$\hat{x}_{k-} = F\hat{x}_{k-1} + Bu_k$$
$$P_{k-} = FP_{k-1}F^T + Q_k$$

Updating

$$K_k = P_{k-}\, H_k^T\left(H_k\, P_{k-}\, H_k^T + R_k\right)^{-1}$$
$$\hat{x}_k = \hat{x}_{k-} + K_k(z_k - H_k\, \hat{x}_{k-})$$
$$P_k = (I - K_k H_k)p_{k-}$$

Conclusion

The primary challenge of localisation in robotics lies in achieving accurate and robust position estimation amidst dynamic environments, sensor noise, and the inherent uncertainty of real-world settings.

The Bayes filter is a probabilistic framework for dynamically updating the estimate of a system's state by combining prior knowledge with observational evidence.

The Particle Filter approximates the posterior distribution of state variables using a set of random samples (particles) and weights.

The Kalman Filter is an optimal recursive data processing algorithm that provides estimates of the internal states of a linear dynamic system from series of noisy measurements.