Sensor fusion, and simultaneous localisation and mapping (SLAM)

Autonomous robots, TME290

Ola Benderius

ola.benderius@chalmers.se

Applied artificial intelligence
Vehicle engineering and autonomous systems
Mechanics and maritime sciences
Chalmers

Introduction

From last lecture

In the previous lecture you learned about global navigation, i.e. localisation, path-planning, and path following. As we also said, a single sensor does not typically help us to achieve robust localisation.

GNSS with sensors and internal model

However, we can involve both (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

GNSS with sensors and internal model

However, we can involve both (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally

GNSS with sensors and internal model

However, we can involve both (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally
- The onboard sensors can keep track of the movement between GNSS updates

GNSS with sensors and internal model

However, we can involve both (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally
- The onboard sensors can keep track of the movement between GNSS updates
- The internal model removes noise since it contains information about possible movement

The Kalman filter

Kalman filter (KF)

The Kalman filter formalises the merging of data as mathematical expressions, and is referred to as an *optimal estimator*.

Kalman filter (KF)

The Kalman filter formalises the merging of data as mathematical expressions, and is referred to as an *optimal estimator*.

A Kalman filter can handle data with different update frequencies.

Kalman filter (KF)

The Kalman filter formalises the merging of data as mathematical expressions, and is referred to as an *optimal estimator*.

A Kalman filter can handle data with different update frequencies.

A Kalman filter (KF) handles linear process and sensor models, but the *extended* Kalman (EKF) filter handles non-linear models.

Kalman filter (KF)

The Kalman filter formalises the merging of data as mathematical expressions, and is referred to as an *optimal estimator*.

A Kalman filter can handle data with different update frequencies.

A Kalman filter (KF) handles linear process and sensor models, but the *extended* Kalman (EKF) filter handles non-linear models.

- Noise is assumed to be of normal distribution
- The word *filter* refers to noise filtering

Kalman filter (KF), concepts

Some concepts are important

State

The value of our state variables (e.g. position, velocity) at a specific time.

- *Estimated state*: the best estimated state values
- *True state*: the actual real-world state values

Kalman filter (KF), concepts

Some concepts are important

State

The value of our state variables (e.g. position, velocity) at a specific time.

- *Estimated state*: the best estimated state values
- *True state*: the actual real-world state values

Prediction

An estimate of our future state based on the internal model.

Kalman filter (KF), concepts

Some concepts are important

State

The value of our state variables (e.g. position, velocity) at a specific time.

- *Estimated state*: the best estimated state values
- *True state*: the actual real-world state values

Prediction

An estimate of our future state based on the internal model.

Observation

An estimate of our state based on sensors.

Kalman filter (KF), running

A Kalman filter is run iteratively. In each time step, a *prediction* is first done based on a process model. Then, in the same time step, this prediction is *updated* based on sensor readings and previous experience on sensor accuracy.

- The filter is typically run throughout the run-time (live) of the robot

Kalman filter (KF), models

A Kalman filter involves two models

Process model

Describes the transition from one state to the next. Can depend on *process input*. A good example in this context: The kinematics model of differential steering.

Kalman filter (KF), models

A Kalman filter involves two models

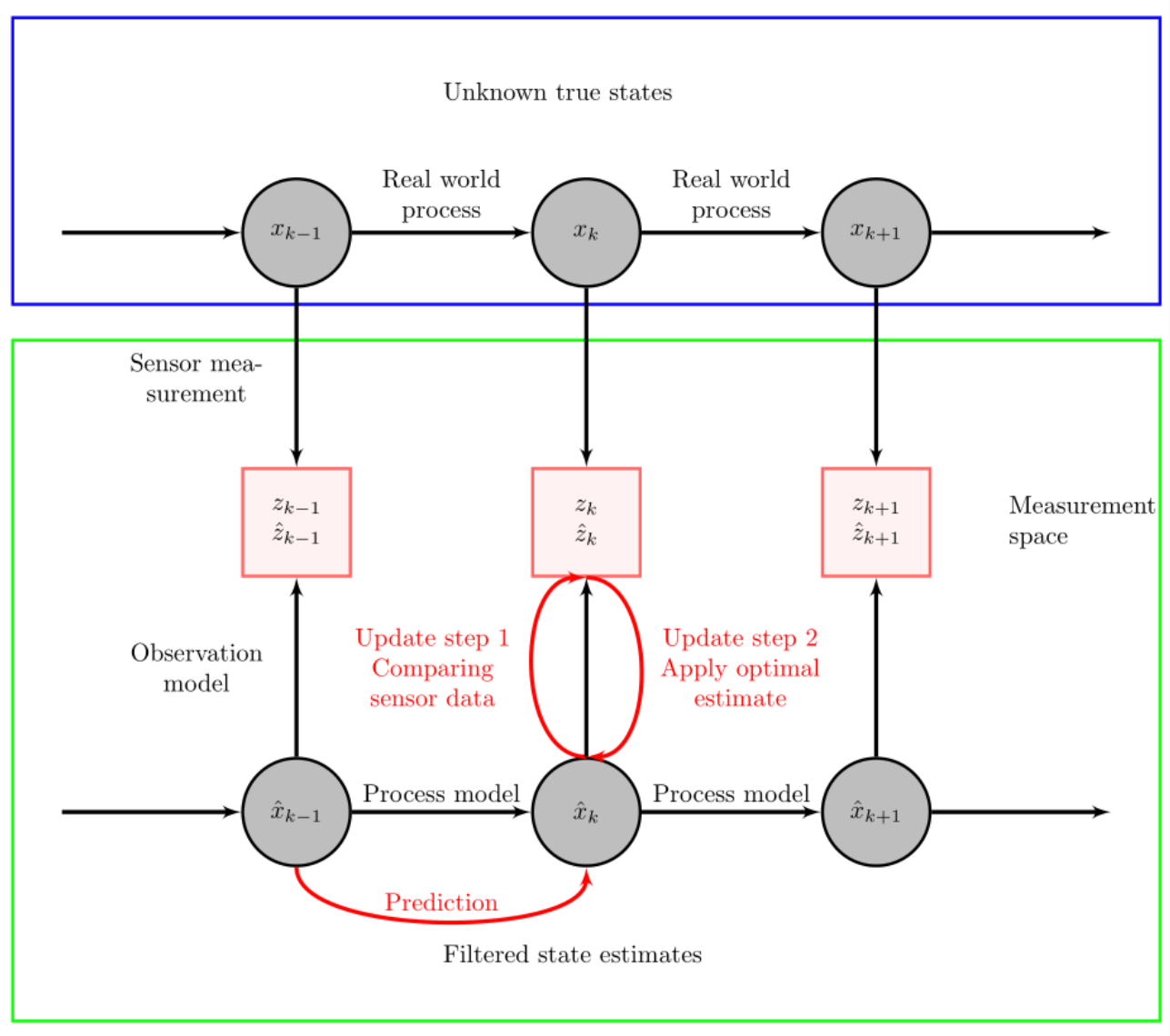
Process model

Describes the transition from one state to the next. Can depend on *process input*. A good example in this context: The kinematics model of differential steering.

Observation model

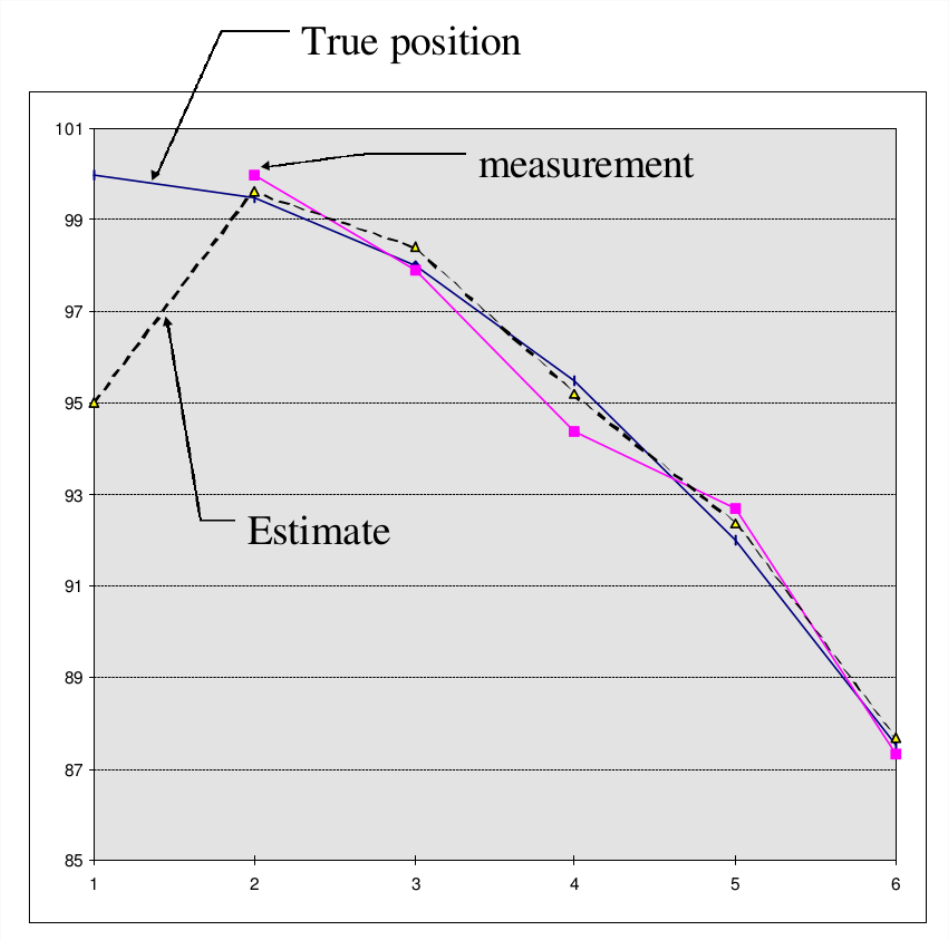
Describes how sensor readings correspond to the state variables. Contains calibration data, in terms of scaling as well as removal of *bias*.

Kalman filter (KF), overview



KF, example

Free fall



(from: `http://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf`)

KF, basics

(from `http://home.wlu.edu/~levys/kalman_tutorial`)

The state x depends on the process model a, and the *control signal* u, at time step k, as

$$x_k = ax_{k-1} + u_k \tag{1}$$

KF, basics

(from `http://home.wlu.edu/~levys/kalman_tutorial`)

The state x depends on the process model a, and the *control signal* u, at time step k, as

$$x_k = ax_{k-1} + u_k \tag{1}$$

The current observation of the system equals to the *actual* state with added sensor noise v

$$z_k = x_k + v_k \tag{2}$$

EKF, free fall example

Remember

$$x_k = ax_{k-1} + u_k \tag{3}$$

EKF, free fall example

Remember

$$x_k = a x_{k-1} + u_k \tag{3}$$

For the free fall example, our state is

$$x_k = \begin{pmatrix} \dot{y}_k \\ y_k \end{pmatrix} \tag{4}$$

where y is the distance over ground.

EKF, free fall example

Remember

$$x_k = a x_{k-1} + u_k \tag{3}$$

For the free fall example, our state is

$$x_k = \begin{pmatrix} \dot{y}_k \\ y_k \end{pmatrix} \tag{4}$$

where y is the distance over ground.

The complete process can then be described as

$$\begin{pmatrix} y_k \\ \dot{y}_k \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y_{k-1} \\ \dot{y}_{k-1} \end{pmatrix} + \begin{pmatrix} 0 \\ g\Delta t \end{pmatrix} \tag{5}$$

KF, observations

The goal is to get the state x also from the observation z, so

$$x_k = z_k - v_k \tag{6}$$

KF, observations

The goal is to get the state x also from the observation z, so

$$x_k = z_k - v_k \tag{6}$$

In the free fall example this could be

$$\begin{pmatrix} y_k \\ \dot{y}_k \end{pmatrix} = \begin{pmatrix} \text{altimeter} \\ 0 \end{pmatrix} - \begin{pmatrix} \text{altimeter noise} \\ 0 \end{pmatrix} \tag{7}$$

KF, noise estimation

The noise is however unpredictable, but it can be *estimated* as the trade-off between the previous estimate and the current observation

$$\hat{x}_k = \hat{x}_{k-1} + g_k(z_k - \hat{x}_{k-1}) = (1 - g_k)\hat{x}_{k-1} + g_k z_k \tag{8}$$

where $\hat{x}_k$ is the estimated state, and g_k is the *Kalman gain*.

KF, noise estimation

The noise is however unpredictable, but it can be *estimated* as the trade-off between the previous estimate and the current observation

$$\hat{x}_k = \hat{x}_{k-1} + g_k(z_k - \hat{x}_{k-1}) = (1 - g_k)\hat{x}_{k-1} + g_k z_k \tag{8}$$

where $\hat{x}_k$ is the estimated state, and g_k is the *Kalman gain*.

Test yourself

The filter will update g_k automatically, but take a moment to investigate how estimations are made when g_k varies between 0 and 1.

KF, Kalman gain

The gain is calculated as

$$g_k = \frac{p_{k-1}}{p_{k-1} + r} \tag{9}$$

$$p_k = (1 - g_k)p_{k-1} \tag{10}$$

where p is the *prediction error* (covariance) and r is the expected noise variance of the sensor.

KF, Kalman gain

The gain is calculated as

$$g_k = \frac{p_{k-1}}{p_{k-1} + r} \tag{9}$$

$$p_k = (1 - g_k)p_{k-1} \tag{10}$$

where p is the *prediction error* (covariance) and r is the expected noise variance of the sensor.

An initial value for p is selected by hand, but then it's updated automatically by the filter. In turn the Kalman *gain* is updated automatically.

KF, process estimation

That was the *observation*, and how it estimated the state.

KF, process estimation

That was the *observation*, and how it estimated the state.

The *process* model on the other hand will predict the state as

$$\hat{x}_k = a\hat{x}_{k-1} + u_k \tag{11}$$

$$p_k = a^2 p_{k-1} \tag{12}$$

Kalman filter, basic form

Model:

$$x_k = a x_{k-1} + u_k \tag{13}$$

$$z_k = x_k + v_k \tag{14}$$

For each time step

(save $\hat{x}_k$, g_k, and p_k between steps)

1. Predict:

$$\hat{x}_k = a \hat{x}_{k-1} + u_k \tag{15}$$

$$p_k = a^2 p_{k-1} \tag{16}$$

2. Update:

$$g_k = \frac{p_k}{p_k + r} \tag{17}$$

$$\hat{x}_k \leftarrow \hat{x}_k + g_k (z_k - \hat{x}_k) \tag{18}$$

$$p_k \leftarrow (1 - g_k) p_k \tag{19}$$

Extended Kalman filter

A Kalman filter assumes that the process and observation models are linear. If they are not, then we need to estimate the model derivatives around each relevant point in the state space (i.e. how does the process behave for small changes in state variables around the current state?).

- This can be done using Jacobians, and it's then more useful to express the filter in matrix form

Extended Kalman filter, matrix form

Model:

$$x_k = f(x_{k-1}, u_k) + w_k \tag{20}$$

$$z_k = h(x_k) + v_k \tag{21}$$

For each time step

1. Predict:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_k) \tag{22}$$

$$P_k = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1} \tag{23}$$

2. Update:

$$G_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1} \tag{24}$$

$$\hat{x}_k \leftarrow \hat{x}_k + G_k(z_k - h(\hat{x}_k)) \tag{25}$$

$$P_k \leftarrow (I - G_k H_k) P_k \tag{26}$$

EKF, handling non-linear models (the extended part)

In the complete form there are four important changes, the non-linear functions $f(x_k, u_k)$ and $h(x_k)$, as well as the Jacobians F_k and H_k.

EKF, handling non-linear models (the extended part)

In the complete form there are four important changes, the non-linear functions $f(x_k, u_k)$ and $h(x_k)$, as well as the Jacobians F_k and H_k.

The Jacobians work as estimations of the non-linear functions around the relevant function area (inserting the current values of the state variables).

EKF, process error

A good thing with the Kalman filter, is that it automatically calculates the process error (covariance) and stores it in P.

- The covariance is calculated for each combination of state variables
- Additionally, this gives information about the performance of the filter

EKF, process error

A good thing with the Kalman filter, is that it automatically calculates the process error (covariance) and stores it in P.

- The covariance is calculated for each combination of state variables
- Additionally, this gives information about the performance of the filter

Reminder

The covariance between two distributions translates to their correlation:

$$\mathrm{cov}(X, Y) = \frac{\sum(x_i - \bar{X})(y_i - \bar{Y})}{n - 1}$$

(a single value that describes how well the distributions match)

EKF, sensor fusion example

$$z_k = h(x_k) + v_k \tag{27}$$

EKF, sensor fusion example

$$z_k = h(x_k) + v_k \tag{27}$$

Could be

$$\begin{pmatrix} \text{magnetometer} \\ \text{gnss} \\ \text{gyroscope} \\ \text{odometer} \end{pmatrix} = \begin{pmatrix} 0 & h_0 \\ h_1 & h_2 \\ 0 & h_3 \\ h_4 & 0 \end{pmatrix} \begin{pmatrix} \text{position} \\ \text{heading} \end{pmatrix} + \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{28}$$

EKF, sensor fusion example

$$z_k = h(x_k) + v_k \tag{27}$$

Could be

$$\begin{pmatrix} \text{magnetometer} \\ \text{gnss} \\ \text{gyroscope} \\ \text{odometer} \end{pmatrix} = \begin{pmatrix} 0 & h_0 \\ h_1 & h_2 \\ 0 & h_3 \\ h_4 & 0 \end{pmatrix} \begin{pmatrix} \text{position} \\ \text{heading} \end{pmatrix} + \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{28}$$

The observation model h maps sensor data into the state variables, it also includes any calibration data for a sensor.

Extended Kalman filter, summary

The extended Kalman filter can estimate the true state based on (1) observed data over time, (2) a process model, and (3) an observation model. Assuming that all noise is of normal distribution, the estimation is optimal.

- Using this method, the *localisation* problem can be solved

Simultaneous localisation and mapping

Local navigation, SLAM

Localisation can also be done by using a map, even when the map is created on the fly.

- The collection name of these functions are SLAM, *simultaneous localisation and mapping*

Local navigation, SLAM

Localisation can also be done by using a map, even when the map is created on the fly.

- The collection name of these functions are SLAM, *simultaneous localisation and mapping*
- The work around this concept started at the 1986 IEEE Robotics and Automation Conference

Local navigation, SLAM

Localisation can also be done by using a map, even when the map is created on the fly.

- The collection name of these functions are SLAM, *simultaneous localisation and mapping*
- The work around this concept started at the 1986 IEEE Robotics and Automation Conference
- There have been very many different versions since then

Local navigation, SLAM

Localisation can also be done by using a map, even when the map is created on the fly.

- The collection name of these functions are SLAM, *simultaneous localisation and mapping*
- The work around this concept started at the 1986 IEEE Robotics and Automation Conference
- There have been very many different versions since then
- Early versions include the EKF-SLAM and FastSLAM

Local navigation, SLAM

Localisation can also be done by using a map, even when the map is created on the fly.

- The collection name of these functions are SLAM, *simultaneous localisation and mapping*
- The work around this concept started at the 1986 IEEE Robotics and Automation Conference
- There have been very many different versions since then
- Early versions include the EKF-SLAM and FastSLAM
- The research is still very active, and there have been recent breakthroughs

Recent high performing vision-based SLAM

- ORB-SLAM2, 2016, University of Zaragoza, Spain
  https://www.youtube.com/watch?v=ufvPS5wJAx0
- Direct Sparse Odometry (DSO), 2016, Technical University of Munich, Germany
  https://www.youtube.com/watch?v=C6-xwSOOdqQ

Neurological SLAM

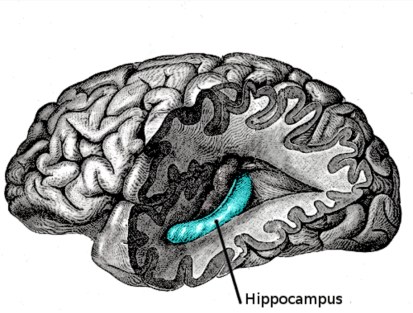
SLAM-like features can also be found in nature:

In 2004, three researchers May-Britt Moser, Edvard I. Moser, and John O'Keefe shared the Nobel prize for the discovery that the brain region *hippocampus* contains SLAM-like functions.

Neurological SLAM

SLAM-like features can also be found in nature:

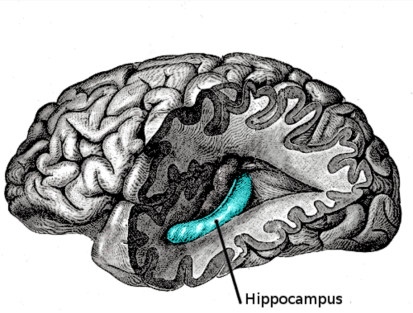
In 2004, three researchers May-Britt Moser, Edvard I. Moser, and John O'Keefe shared the Nobel prize for the discovery that the brain region *hippocampus* contains SLAM-like functions.



Neurological SLAM

SLAM-like features can also be found in nature:

In 2004, three researchers May-Britt Moser, Edvard I. Moser, and John O'Keefe shared the Nobel prize for the discovery that the brain region *hippocampus* contains SLAM-like functions.



- There is an implementation of bio-inspired SLAM called OpenRatSLAM (2013)
  `https://www.youtube.com/watch?v=p8mf0SZKneY`

Vision-based SLAM

A single monocular camera can normally not see any depth information. How does it work?

Vision-based SLAM

A single monocular camera can normally not see any depth information. How does it work?

SLAM is used, and the camera position over time is estimated based on features in the scene (landmarks). Then, as we have several images, all taken from different known positions, we can simulate stereo vision!

EKF-SLAM

EKF-SLAM, the school book example

Before we learned about the extended Kalman filter (EKF), then merging a GNSS signal, wheel encoders, IMU, and a internal (process) model.

EKF-SLAM, the school book example

Before we learned about the extended Kalman filter (EKF), then merging a GNSS signal, wheel encoders, IMU, and a internal (process) model.

This time we want to remove the requirement of GNSS, and instead use previously observed landmarks (no global information).

EKF-SLAM, the school book example

Before we learned about the extended Kalman filter (EKF), then merging a GNSS signal, wheel encoders, IMU, and a internal (process) model.

This time we want to remove the requirement of GNSS, and instead use previously observed landmarks (no global information).

Additionally, SLAM will even allow us to remove the requirement of a global map. However, such map can be used for online calibration.

EKF-SLAM, basics

The algorithm consists of three steps

- Update the current state estimate using (for example) wheel encoders
- Update the estimated state from re-observing landmarks
- Add new landmarks to the current state

EKF-SLAM, update state estimate

The first step is, as previously:

- The state x_k of the robot contains (x, y, φ)
- The control of the robot results in Δx, Δy, and $\Delta \varphi$

EKF-SLAM, re-observing landmarks

The second step:

- The estimate of the current position is compared to where landmarks *should be*
- The difference is referred to as the *innovation*
- The amount of uncertainty of each observed landmark is updated
- If re-observing a landmark *where it is supposed to be*, decreases the uncertainty

EKF-SLAM, add landmarks (update map)

- Uses information about the current position
- And relation between the new and old landmarks

EKF-SLAM, extended state

- In addition to the robot position (x, y, φ), the state x_k also includes the position of each of the n landmarks $(x_i,\ y_i)$
- Therefore, the state vector will increase in size over time (size $3 + 2 * n$)

EKF-SLAM, covariance matrix

- As learned before, the covariance matrix P_k keeps information about *data accuracy*

Reminder

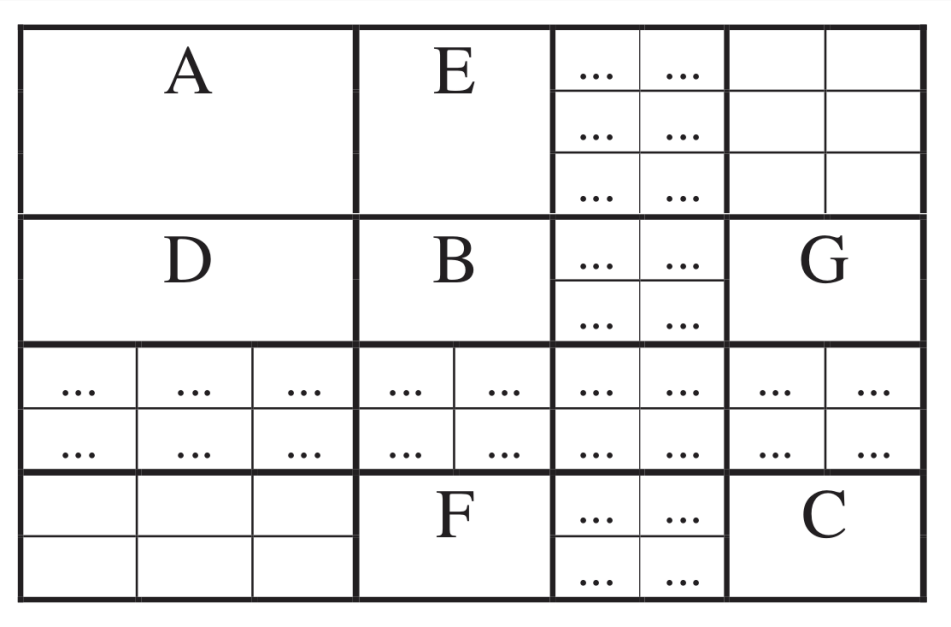
The covariance between two distributions translates to their correlation:

$$\mathrm{cov}(X, Y) = \frac{\sum(x_i - \bar{X})(y_i - \bar{Y})}{n - 1}$$

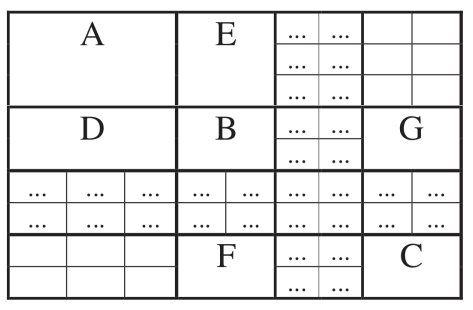
(a single value that describes how well the distributions match)

EKF-SLAM, covariance matrix

- By using the state definition on the previous slide, the covariance matrix will look like:

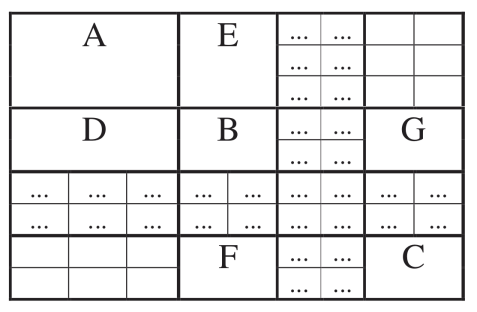


EKF-SLAM, covariance matrix composition



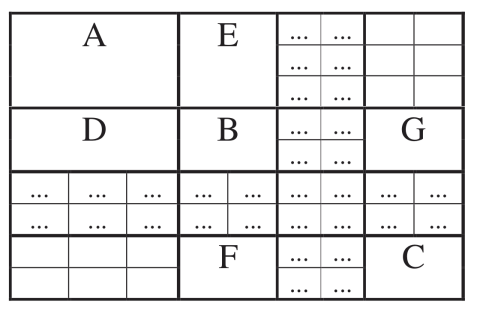
- A (3x3) is the covariance on robot position (x, y, φ)
- B (2x2) is the covariance of the first landmark
- C (2x2) is the covariance of landmark n
- D (2x3) is between the robot state and first landmark
- E (3x2) is between first landmark and robot state ($E = D^T$)
- F (2x2) is between the last and first landmarks
- G (2x2) is between the first and last landmarks ($G = F^T$)

EKF-SLAM, covariance matrix growth



How will the covariance matrix P_k grow when a new landmark is observed?

EKF-SLAM, covariance matrix growth



How will the covariance matrix P_k grow when a new landmark is observed?

Check

How are the values of the covariance matrix P_k selected?

EKF-SLAM, the Kalman gain

- The Kalman gain G_k contains the system's *trust* for each observation
- Remember: The Kalman gain is calculated as

$$G_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1} \tag{29}$$

where H is the Jacobian of the observation model and R is the systematic error of the observation

EKF-SLAM, the Kalman gain

- The Kalman gain G_k contains the system's *trust* for each observation
- Remember: The Kalman gain is calculated as

$$G_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1} \tag{29}$$

  where H is the Jacobian of the observation model and R is the systematic error of the observation
- Therefore, the Kalman gain is calculated from the covariance matrix automatically (great!)

EKF-SLAM, the observation model

- The observation model $h(x_k)$ reflects the relation between observations and state variables

EKF-SLAM, the observation model

- The observation model $h(x_k)$ reflects the relation between observations and state variables

The landmark observations can be related to the robot state via *range* and *bearing*

$$\begin{pmatrix} r \\ \beta \end{pmatrix} = \begin{pmatrix} \sqrt{(x_i - x)^2 + (y_i - y)^2} \\ \tan^{-1}\left(\frac{y_i - y}{x_i - x}\right) - \varphi \end{pmatrix} \tag{30}$$

where (x, y, φ) is the position of the robot and (x_i, y_i) is the position of the landmark

EKF-SLAM, example

For each time step (reminder)

1. Predict:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_k) \tag{31}$$

$$P_k = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1} \tag{32}$$

2. Update:

$$G_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1} \tag{33}$$

$$\hat{x}_k \leftarrow \hat{x}_k + G_k (z_k - h(\hat{x}_k)) \tag{34}$$

$$P_k \leftarrow (I - G_k H_k) P_k \tag{35}$$

EKF-SLAM, example eq. 1

$$\begin{pmatrix} x_k \\ y_k \\ \varphi_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \varphi_{k-1} \end{pmatrix} + \begin{pmatrix} \dot{x}_k \Delta t \\ \dot{y}_k \Delta t \\ \dot{\varphi}_k \Delta t \end{pmatrix} \tag{36}$$

Also consider the landmark states:

$$\begin{pmatrix} x_k \\ y_k \\ \varphi_k \\ \vdots \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \varphi_{k-1} \\ \vdots \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{x}_k \Delta t \\ \dot{y}_k \Delta t \\ \dot{\varphi}_k \Delta t \end{pmatrix} \tag{37}$$

The process model only affects the robot's motion (not the landmarks).

EKF-SLAM, example eq. 2

$$\hat{P}_k = F_k P_{k-1} F_k^T + Q_k \tag{38}$$

- Both the process model and the covariance matrix (discussed above) are of size (3+2n)x(3+2n)

EKF-SLAM, example eq. 2

$$\hat{P}_k = F_k P_{k-1} F_k^T + Q_k \tag{38}$$

- Both the process model and the covariance matrix (discussed above) are of size (3+2n)x(3+2n)

$$\begin{pmatrix} \frac{\delta x}{\delta x} & \frac{\delta x}{\delta y} & \frac{\delta x}{\delta \varphi} & 0 & \cdots & 0 \\ \frac{\delta y}{\delta x} & \frac{\delta y}{\delta y} & \frac{\delta y}{\delta \varphi} & 0 & \cdots & 0 \\ \frac{\delta \varphi}{\delta x} & \frac{\delta \varphi}{\delta y} & \frac{\delta \varphi}{\delta \varphi} & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{39}$$

EKF-SLAM, example step 3

- H_k (Jacobian of h) needs to be determined
- We need to use range and bearing (r, β)
- If a landmark i has not yet been observed:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \begin{pmatrix} r_{k,i} \cos(\beta_{k,i} + \varphi_k) \\ r_{k,i} \sin(\beta_{k,i} + \varphi_k) \end{pmatrix} \tag{40}$$

EKF-SLAM, example eq. 3

$$G_k = \hat{P}_k H_k^T (H_k \hat{P}_k H_k^T + R)^{-1} \tag{41}$$

EKF-SLAM, example eq. 3

$$G_k = \hat{P}_k H_k^T (H_k \hat{P}_k H_k^T + R)^{-1} \tag{41}$$

H_k is the Jacobian of the observation model. That is, how are the observations predicted to change as out state changes?

EKF-SLAM, example eq. 3

$$G_k = \hat{P}_k H_k^T (H_k \hat{P}_k H_k^T + R)^{-1} \tag{41}$$

H_k is the Jacobian of the observation model. That is, how are the observations predicted to change as out state changes?

G_k is the Kalman gains. That is, how do we scale between observations and our process model?

EKF-SLAM, example eqs. 4 and 5

The two final steps updates our state and covariance matrix, which are used in the next iteration of the filter.

$$\hat{x}_k \leftarrow \hat{x}_k + G_k(z_k - h(\hat{x}_k)) \tag{42}$$

$$P_k \leftarrow (I - G_k H_k) P_k \tag{43}$$