# Sensors and Sensing
# Filtering

Todor Stoyanov

Centre for Applied Autonomous Sensor Systems
Örebro University
Sweden

# Outline

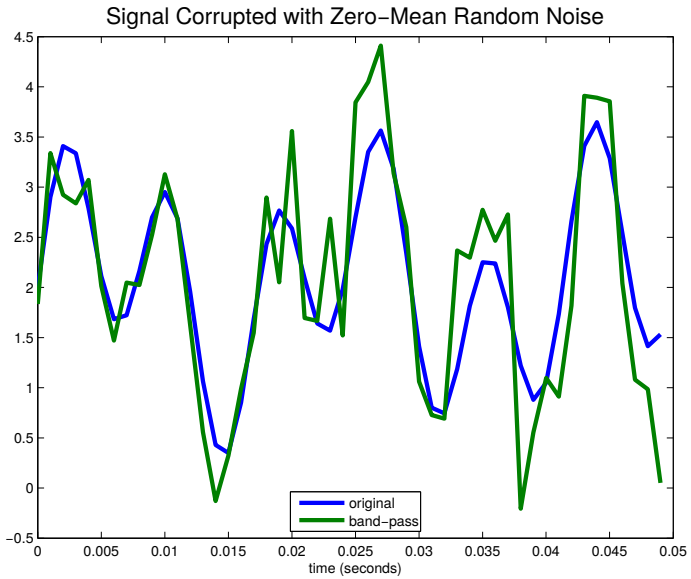
# Outline

# Signals

- Going back to definition from the first lecture:

$$y(t) = x(t) + \nu(t) + \eta(t) \quad (1)$$

- Several classical filters from signal processing can be applied to remove noise from y(t).



# Fourier Transform

- A standard way to think about time-variant signals is through their frequency-domain interpretation.
- Any signal can be approximated by a sum of sinusoidal waves with varying frequencies and amplitudes.

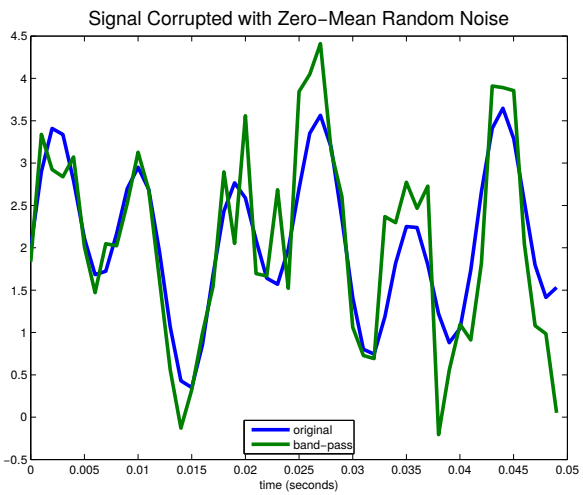
# Fourier Transform

- For a signal $f(t)$, it's Fourier transform $\hat{f}(\omega)$ is:

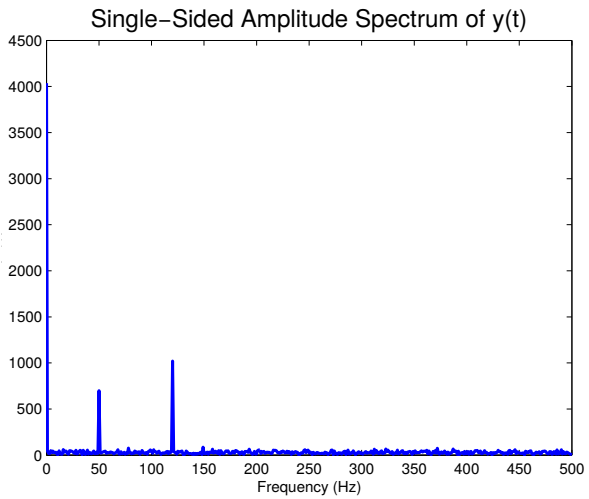
$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt \tag{2}$$

- To re-construct the original signal, we take the inverse Fourier transform:

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\omega) e^{2\pi i \omega t} d\omega \tag{3}$$



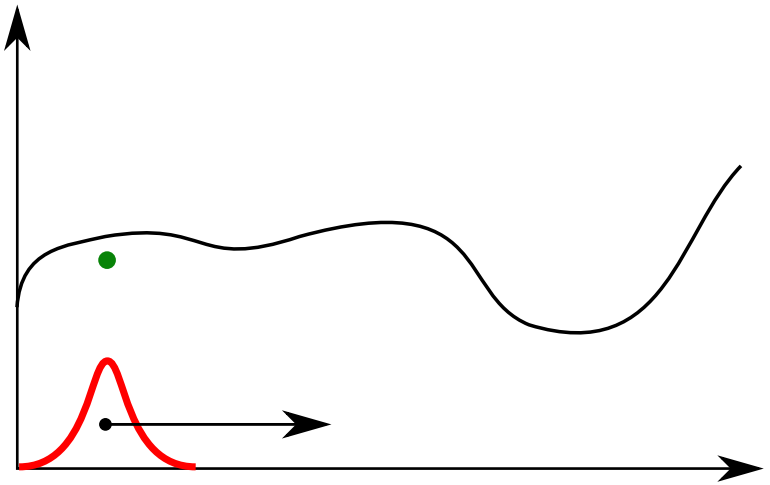

(a)




(b)

# Convolution

- Applying a filter to f(t) can be formulated through convolution.
- Given a filter signal (kernel) g(t), we slide it over f(t) and compute a filtered signal $f^*(t)$ as:

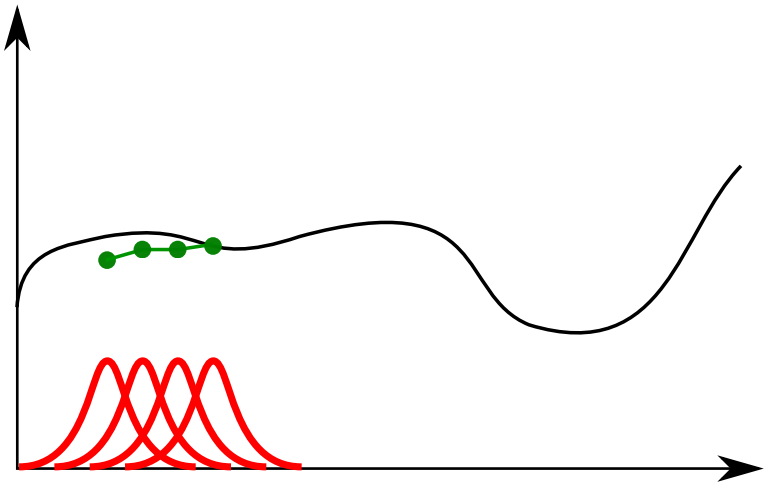


$$f^*(z) = f(t) * g(t) = \int_{-\infty}^{\infty} f(t)g(z-t)dt \quad (4)$$

- Convolution with an appropriately selected kernel g(t) can be used to filter out noise.

# Convolution

- Applying a filter to f(t) can be formulated through convolution.
- Given a filter signal (kernel) g(t), we slide it over f(t) and compute a filtered signal $f^*(t)$ as:



$$f^*(z) = f(t) * g(t) = \int_{-\infty}^{\infty} f(t)g(z-t)dt \quad (4)$$

- Convolution with an appropriately selected kernel g(t) can be used to filter out noise.

# Convolution Theorem

- The convolution theorem is a central tool for signal processing.
- Given two signals $f(t)$ and $g(t)$ and their Fourier transforms $F(\omega) = \mathcal{F}(f(t))$ and $G(\omega) = \mathcal{F}(g(t))$:

$$f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega)) \tag{5}$$

- Thus, convolution in the time domain is equivalent to multiplication in the frequency domain.
- Fourier transforms can be performed fast for digital signals using the FFT (Fast Fourier Transform) algorithm.
- Time-consuming convolution with a filter kernel is often implemented by FFT and it's inverse IFFT.

# Convolution Theorem

- The convolution theorem is a central tool for signal processing.
- Given two signals $f(t)$ and $g(t)$ and their Fourier transforms $F(\omega) = \mathcal{F}(f(t))$ and $G(\omega) = \mathcal{F}(g(t))$:

$$f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega)) \tag{5}$$

- Thus, convolution in the time domain is equivalent to multiplication in the frequency domain.
- Fourier transforms can be performed fast for digital signals using the FFT (Fast Fourier Transform) algorithm.
- Time-consuming convolution with a filter kernel is often implemented by FFT and it's inverse IFFT.

# Convolution Theorem

- The convolution theorem is a central tool for signal processing.
- Given two signals $f(t)$ and $g(t)$ and their Fourier transforms $F(\omega) = \mathcal{F}(f(t))$ and $G(\omega) = \mathcal{F}(g(t))$:

$$f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega)) \tag{5}$$

- Thus, convolution in the time domain is equivalent to multiplication in the frequency domain.
- Fourier transforms can be performed fast for digital signals using the FFT (Fast Fourier Transform) algorithm.
- Time-consuming convolution with a filter kernel is often implemented by FFT and it's inverse IFFT.

# Convolution Theorem

- The convolution theorem is a central tool for signal processing.
- Given two signals $f(t)$ and $g(t)$ and their Fourier transforms $F(\omega) = \mathcal{F}(f(t))$ and $G(\omega) = \mathcal{F}(g(t))$:

$$f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega)) \tag{5}$$

- Thus, convolution in the time domain is equivalent to multiplication in the frequency domain.
- Fourier transforms can be performed fast for digital signals using the FFT (Fast Fourier Transform) algorithm.
- Time-consuming convolution with a filter kernel is often implemented by FFT and it's inverse IFFT.

# Convolution Theorem

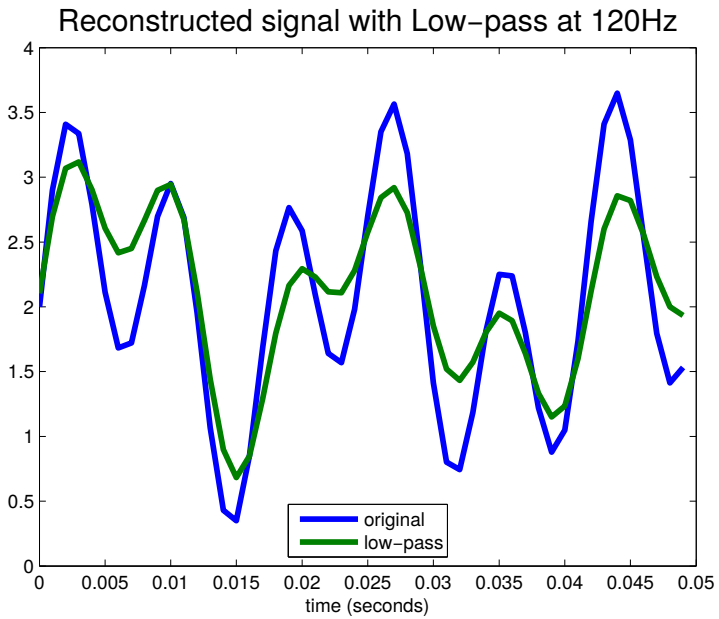
- The convolution theorem is a central tool for signal processing.
- Given two signals $f(t)$ and $g(t)$ and their Fourier transforms $F(\omega) = \mathcal{F}(f(t))$ and $G(\omega) = \mathcal{F}(g(t))$:

$$f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega)) \tag{5}$$

- Thus, convolution in the time domain is equivalent to multiplication in the frequency domain.
- Fourier transforms can be performed fast for digital signals using the FFT (Fast Fourier Transform) algorithm.
- Time-consuming convolution with a filter kernel is often implemented by FFT and it's inverse IFFT.

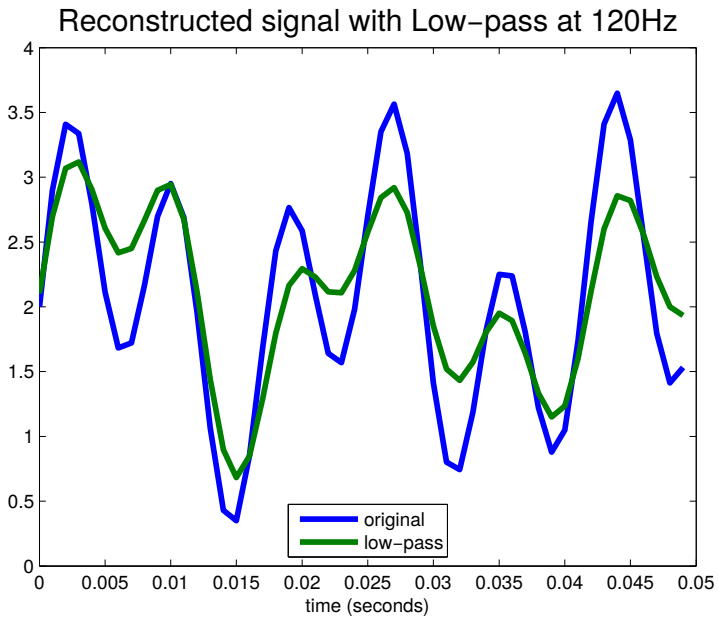
# Low-pass Filter

- Low-pass filters are derived from AC circuits (RC resistor-capacitor series circuit, output over C).
- Central idea: let through low-frequency signals and dampen high-frequency components.
- In frequency domain, ideal low-pass filter is a step (or ramp) function, sloping down.
- In time domain it is a sigmoid.
- High frequency components are often due to random noise.



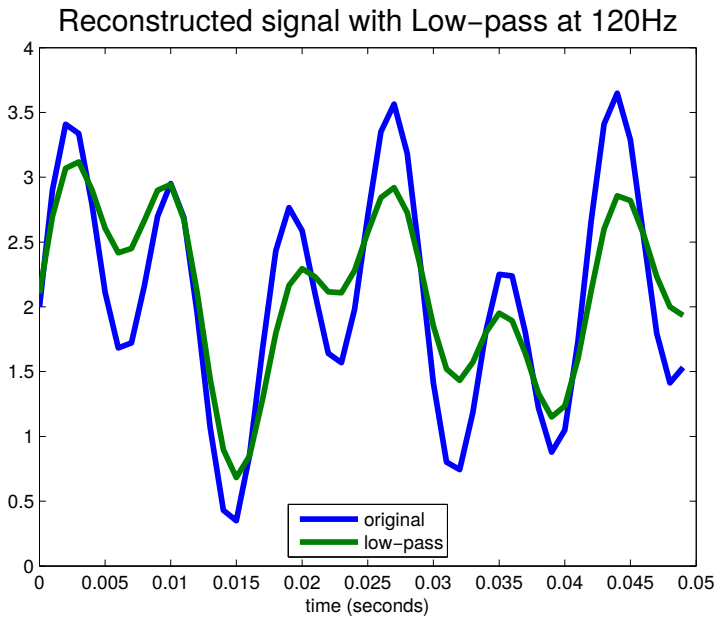
# Low-pass Filter

- Low-pass filters are derived from AC circuits (RC resistor-capacitor series circuit, output over C).
- Central idea: let through low-frequency signals and dampen high-frequency components.
- In frequency domain, ideal low-pass filter is a step (or ramp) function, sloping down.
- In time domain it is a sigmoid.
- High frequency components are often due to random noise.



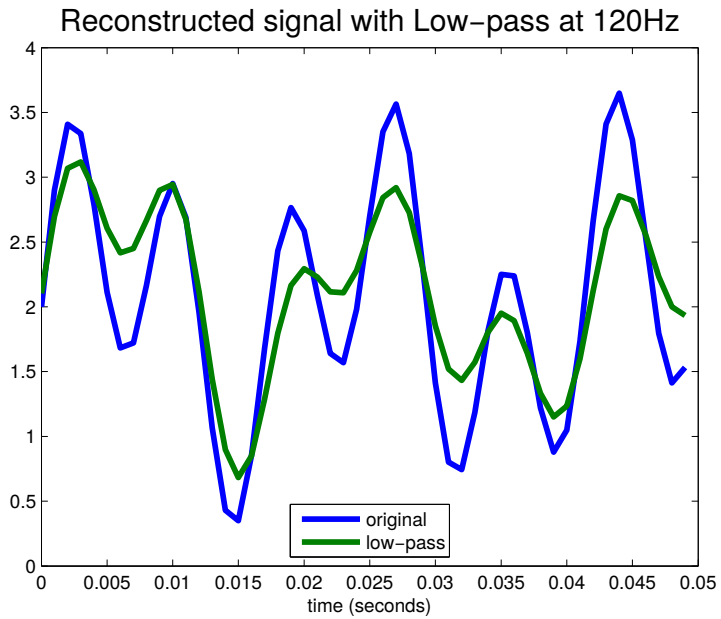
# Low-pass Filter

- Low-pass filters are derived from AC circuits (RC resistor-capacitor series circuit, output over C).
- Central idea: let through low-frequency signals and dampen high-frequency components.
- In frequency domain, ideal low-pass filter is a step (or ramp) function, sloping down.
- In time domain it is a sigmoid.
- High frequency components are often due to random noise.



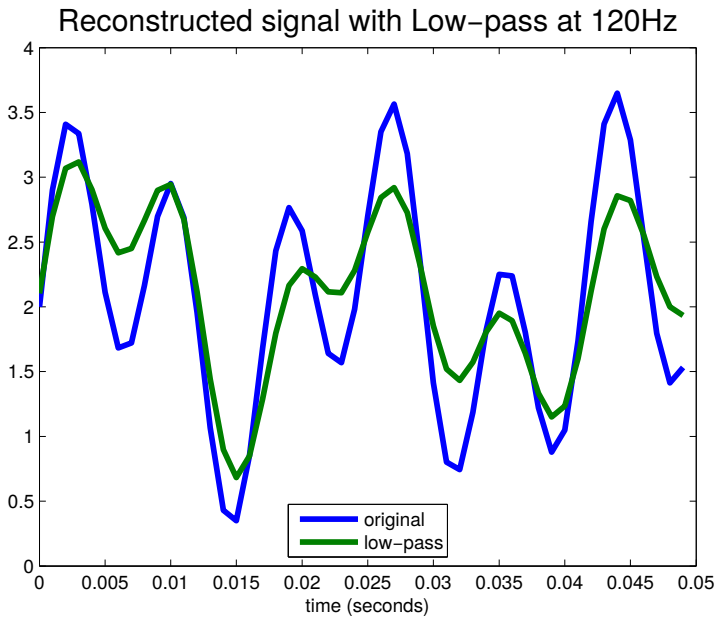
# Low-pass Filter

- Low-pass filters are derived from AC circuits (RC resistor-capacitor series circuit, output over C).
- Central idea: let through low-frequency signals and dampen high-frequency components.
- In frequency domain, ideal low-pass filter is a step (or ramp) function, sloping down.
- In time domain it is a sigmoid.
- High frequency components are often due to random noise.



# Low-pass Filter

- Low-pass filters are derived from AC circuits (RC resistor-capacitor series circuit, output over C).
- Central idea: let through low-frequency signals and dampen high-frequency components.
- In frequency domain, ideal low-pass filter is a step (or ramp) function, sloping down.
- In time domain it is a sigmoid.
- High frequency components are often due to random noise.



# High-pass Filter

- High-pass filters only let through high-frequency components.
- It is the dual of the low-pass (output over R in an RC circuit).
- Useful for eliminating baseline or detect high-frequency signals (e.g. edges in an image)
- In frequency domain it is also a step or ramp function, sloping up.

# High-pass Filter

- High-pass filters only let through high-frequency components.
- It is the dual of the low-pass (output over R in an RC circuit).
- Useful for eliminating baseline or detect high-frequency signals (e.g. edges in an image)
- In frequency domain it is also a step or ramp function, sloping up.

# High-pass Filter

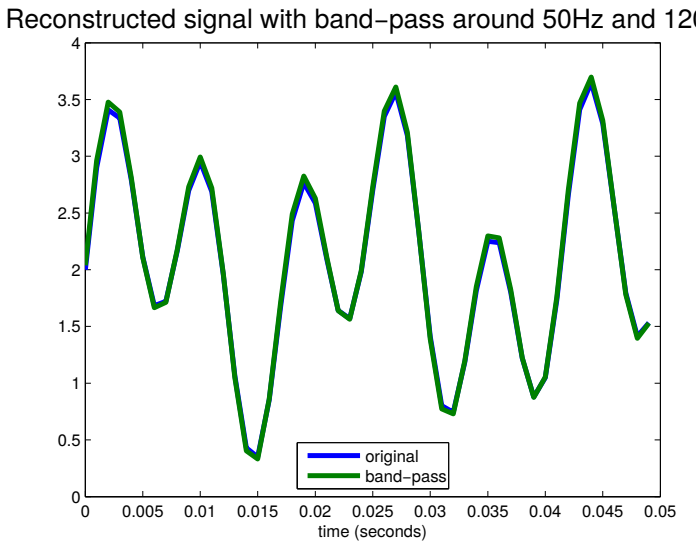
- High-pass filters only let through high-frequency components.
- It is the dual of the low-pass (output over R in an RC circuit).
- Useful for eliminating baseline or detect high-frequency signals (e.g. edges in an image)
- In frequency domain it is also a step or ramp function, sloping up.

# High-pass Filter

- High-pass filters only let through high-frequency components.
- It is the dual of the low-pass (output over R in an RC circuit).
- Useful for eliminating baseline or detect high-frequency signals (e.g. edges in an image)
- In frequency domain it is also a step or ramp function, sloping up.

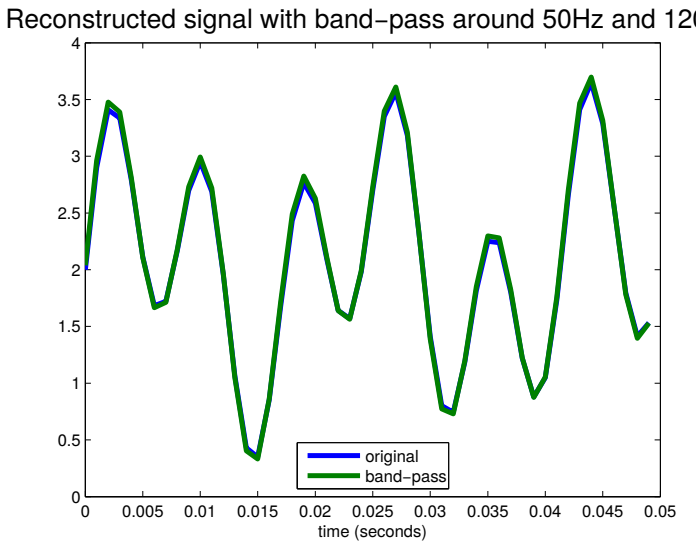
# Band-pass and Band-stop Filters

- By combining a low pass and a high pass filter, we can obtain a band pass or band reject filter.
- Main idea is to only let through a specific band of frequencies through.
- Useful when we want to reject a particular repetative noise pattern, or detect a particular signal.



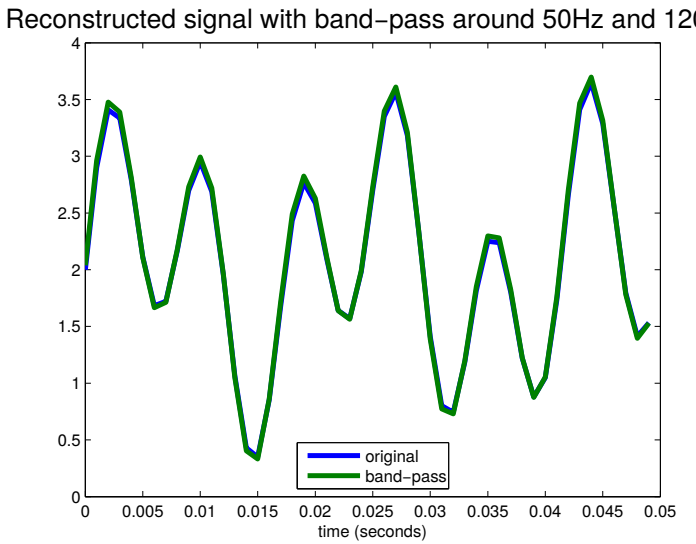
# Band-pass and Band-stop Filters

- By combining a low pass and a high pass filter, we can obtain a band pass or band reject filter.
- Main idea is to only let through a specific band of frequencies through.
- Useful when we want to reject a particular repetative noise pattern, or detect a particular signal.



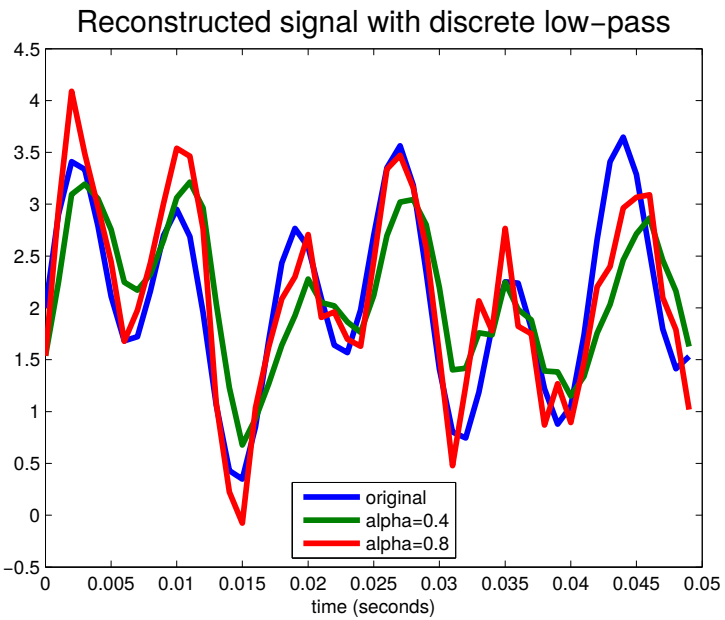
# Band-pass and Band-stop Filters

- By combining a low pass and a high pass filter, we can obtain a band pass or band reject filter.
- Main idea is to only let through a specific band of frequencies through.
- Useful when we want to reject a particular repetative noise pattern, or detect a particular signal.



# Sliding Window

- Time-domain approximations can be obtained by discrete convolution.
- If we restrict the kernel size, the operation is akin to sliding a window over the signal.
- e.g., a discrete approximation of a low pass filter is:




**Algorithm 1: Discrete low-pass filter**

$x_f(0) = x(0)$;
for $i \in 1 \cdots n$ do
  $x_f(i) = \alpha x(i) + (1 - \alpha) x_f(i-1)$;

# Outline

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{\mathrm{x}}_t, \mathrm{P}_t)$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{x}_t, P_t)$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{\mathrm{x}}_\mathrm{t}, \mathrm{P}_\mathrm{t})$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{\mathrm{x}}_t, \mathrm{P}_t)$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{\mathrm{x}}_t, \mathrm{P}_t)$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — what is it?

- The Kalman filter (KF) is one of the classical methods for fusing observations from different sensors for a more robus state estimate.
- Proposed by Rudolph Kalman in 1950s.
- It is a linear Gaussian filter.
- The assumptions are that the state variable can be modeled using a gaussian pdf $\mathcal{N}(\hat{\mathrm{x}}_t, \mathrm{P}_t)$.
- In addition, the KF assumes the state evolves as a linear function with Gaussian noise.
- This assumption is relaxed in the Extended KF (EKF), using linearization around the current estimate.

# Kalman filter — Formulation

- Given a state variable over time $x_t$, the next state is a linear function of the previous state and the controls $u_t$:

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, Q)$

- The probability of making an observation $z_t$ is also a linear function of $x_t$ with added Gaussian noise:

$$z_t = Hx_t + \delta_t$$

where $\delta_t \sim \mathcal{N}(0, R)$

# Kalman filter — Formulation

- Given a state variable over time $x_t$, the next state is a linear function of the previous state and the controls $u_t$:

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, Q)$

- The probability of making an observation $z_t$ is also a linear function of $x_t$ with added Gaussian noise:

$$z_t = Hx_t + \delta_t$$

where $\delta_t \sim \mathcal{N}(0, R)$

# Kalman filter — Formulation

- The KF assumes an initial state $x_0$, with a normal distribution of covariance $P_0$.
- We then predict the next state variables:

$$\hat{x}_t^- = A\hat{x}_{t-1} + Bu_t \tag{6}$$

$$P_t^- = AP_{t-1}A^T + Q \tag{7}$$

- The measurements $z_t$ are then used to correct the prediction:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{8}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \tag{9}$$

$$P_t = (I - K_t H)P_t^- \tag{10}$$

- $K_t$ is the Kalman gain.

# Kalman filter — Formulation

- The KF assumes an initial state $x_0$, with a normal distribution of covariance $P_0$.
- We then predict the next state variables:

$$\hat{x}_t^- = A\hat{x}_{t-1} + Bu_t \tag{6}$$

$$P_t^- = AP_{t-1}A^T + Q \tag{7}$$

- The measurements $z_t$ are then used to correct the prediction:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{8}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \tag{9}$$

$$P_t = (I - K_t H)P_t^- \tag{10}$$

- $K_t$ is the Kalman gain.

# Kalman filter — Formulation

- The KF assumes an initial state $x_0$, with a normal distribution of covariance $P_0$.
- We then predict the next state variables:

$$\hat{x}_t^- = A\hat{x}_{t-1} + Bu_t \tag{6}$$

$$P_t^- = AP_{t-1}A^T + Q \tag{7}$$

- The measurements $z_t$ are then used to correct the prediction:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{8}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \tag{9}$$

$$P_t = (I - K_t H)P_t^- \tag{10}$$

- $K_t$ is the Kalman gain.

# Kalman filter — Formulation

- The KF assumes an initial state $x_0$, with a normal distribution of covariance $P_0$.
- We then predict the next state variables:

$$\hat{x}_t^- = A\hat{x}_{t-1} + Bu_t \tag{6}$$

$$P_t^- = AP_{t-1}A^T + Q \tag{7}$$

- The measurements $z_t$ are then used to correct the prediction:

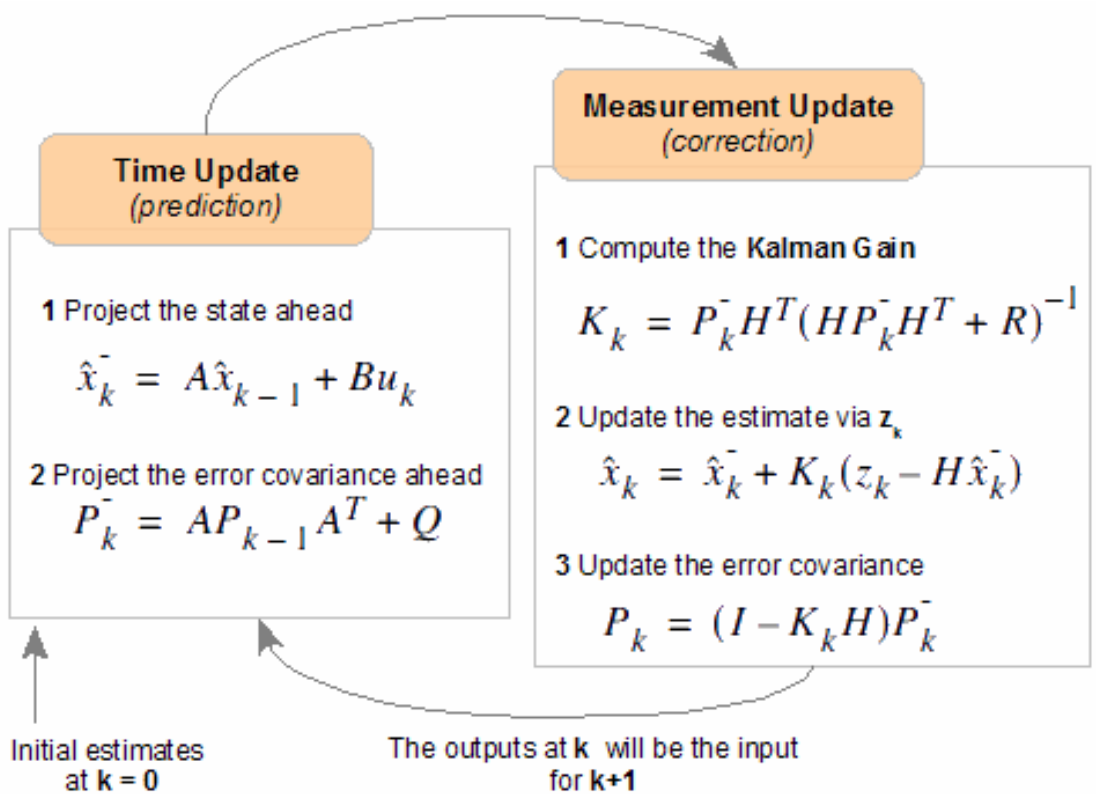
$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{8}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \tag{9}$$

$$P_t = (I - K_t H)P_t^- \tag{10}$$

- $K_t$ is the Kalman gain.

# Kalman filter — Formulation



# Kalman filter — Example

- Simple 1D example. Robot moving on a line, robots state $x_t$ is the position along the line in m. $P_t$ is the varaince (1D).
- Controls $u_t$ are 1D speeds (+/-) in m/s. For simplicity we take $\Delta t = 1s$. $A = B = 1$.
- Measurements are the distance to a landmark $l_1 = 1m$. Let $z'_t = d_1$.
- H relates measurements to state. In order to get a linear measurement model, we set $z_t = z'_t - l_1$. Then, $H = -1$.
- Set the variances of controls and sensors: $Q = 0.3$, $R = 0.1$.
- Let $x_0 = 2m$, $P_0 = 2$ be an uncertain initial position estimate.
- We observe controls $u_1 = 1, u_2 = 1, u_3 = -2$ and landmarks as $z_1 = -3.8, z_2 = -4.9, z_3 = -2.2$.

# Kalman filter — Example

- Simple 1D example. Robot moving on a line, robots state $x_t$ is the position along the line in m. $P_t$ is the varaince (1D).
- Controls $u_t$ are 1D speeds (+/-) in m/s. For simplicity we take $\Delta t = 1s$. $A = B = 1$.
- Measurements are the distance to a landmark $l_1 = 1m$. Let $z'_t = d_1$.
- H relates measurements to state. In order to get a linear measurement model, we set $z_t = z'_t - l_1$. Then, $H = -1$.
- Set the variances of controls and sensors: $Q = 0.3$, $R = 0.1$.
- Let $x_0 = 2m$, $P_0 = 2$ be an uncertain initial position estimate.
- We observe controls $u_1 = 1, u_2 = 1, u_3 = -2$ and landmarks as $z_1 = -3.8, z_2 = -4.9, z_3 = -2.2$.

# Kalman filter — Example

- Simple 1D example. Robot moving on a line, robots state $x_t$ is the position along the line in m. $P_t$ is the varaince (1D).
- Controls $u_t$ are 1D speeds (+/-) in m/s. For simplicity we take $\Delta t = 1s$. $A = B = 1$.
- Measurements are the distance to a landmark $l_1 = 1m$. Let $z'_t = d_1$.
- H relates measurements to state. In order to get a linear measurement model, we set $z_t = z'_t - l_1$. Then, $H = -1$.
- Set the variances of controls and sensors: $Q = 0.3$, $R = 0.1$.
- Let $x_0 = 2m$, $P_0 = 2$ be an uncertain initial position estimate.
- We observe controls $u_1 = 1, u_2 = 1, u_3 = -2$ and landmarks as $z_1 = -3.8, z_2 = -4.9, z_3 = -2.2$.

# Kalman filter — Example

- Simple 1D example. Robot moving on a line, robots state $x_t$ is the position along the line in m. $P_t$ is the varaince (1D).
- Controls $u_t$ are 1D speeds (+/-) in m/s. For simplicity we take $\Delta t = 1s$. $A = B = 1$.
- Measurements are the distance to a landmark $l_1 = 1m$. Let $z'_t = d_1$.
- H relates measurements to state. In order to get a linear measurement model, we set $z_t = z'_t - l_1$. Then, $H = -1$.
- Set the variances of controls and sensors: $Q = 0.3$, $R = 0.1$.
- Let $x_0 = 2m$, $P_0 = 2$ be an uncertain initial position estimate.
- We observe controls $u_1 = 1, u_2 = 1, u_3 = -2$ and landmarks as $z_1 = -3.8, z_2 = -4.9, z_3 = -2.2$.

# Kalman filter — Example

- Predict:

$$\hat{x}_t^- = 2 + 1 \tag{11}$$

$$P_t^- = 2 + 0.3 \tag{12}$$

- Correct:

$$K_t = 2.3 * -1 * (-1 * 2.3 * -1 + 0.1)^{-1} = -0.95 \tag{13}$$

$$\hat{x}_t = 3 - 0.95 * (-3.8 - (-1) * 3) = 3.7 \tag{14}$$

$$P_t = (1 - (-0.95 * -1))2.5 = 0.125 \tag{15}$$

- Variance drops substantially as we obtain more certain measurements.
- Same procedure for the next two observations.

# Kalman filter — Example

- Predict:

$$\hat{x}_t^- = 2 + 1 \tag{11}$$

$$P_t^- = 2 + 0.3 \tag{12}$$

- Correct:

$$K_t = 2.3 * -1 * (-1 * 2.3 * -1 + 0.1)^{-1} = -0.95 \tag{13}$$

$$\hat{x}_t = 3 - 0.95 * (-3.8 - (-1) * 3) = 3.7 \tag{14}$$

$$P_t = (1 - (-0.95 * -1))2.5 = 0.125 \tag{15}$$

- Variance drops substantially as we obtain more certain measurements.
- Same procedure for the next two observations.

# Kalman filter — Example

- Predict:

$$\hat{x}_t^- = 2 + 1 \tag{11}$$

$$P_t^- = 2 + 0.3 \tag{12}$$

- Correct:

$$K_t = 2.3 * -1 * (-1 * 2.3 * -1 + 0.1)^{-1} = -0.95 \tag{13}$$

$$\hat{x}_t = 3 - 0.95 * (-3.8 - (-1) * 3) = 3.7 \tag{14}$$

$$P_t = (1 - (-0.95 * -1))2.5 = 0.125 \tag{15}$$

- Variance drops substantially as we obtain more certain measurements.
- Same procedure for the next two observations.

# Kalman filter — Example

- Predict:

$$\hat{x}_t^- = 2 + 1 \tag{11}$$

$$P_t^- = 2 + 0.3 \tag{12}$$

- Correct:

$$K_t = 2.3 * -1 * (-1 * 2.3 * -1 + 0.1)^{-1} = -0.95 \tag{13}$$

$$\hat{x}_t = 3 - 0.95 * (-3.8 - (-1) * 3) = 3.7 \tag{14}$$

$$P_t = (1 - (-0.95 * -1))2.5 = 0.125 \tag{15}$$

- Variance drops substantially as we obtain more certain measurements.
- Same procedure for the next two observations.

# Extended Kalman Filter (EKF)

- The Kalman filter requires linear process and observation models.
- In most cases in robotics we have non-linear relationships (e.g., robot moving on a 2D plane)
- We re-define the process and observation models as:

$$x_t = f(x_{t-1}, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \delta_t$$

where f and h can be any non-linear differentiable functions.

- We will now linearize the functions f and h around the current state estimate $\hat{x}_t$

# Extended Kalman Filter (EKF)

- The Kalman filter requires linear process and observation models.
- In most cases in robotics we have non-linear relationships (e.g., robot moving on a 2D plane)
- We re-define the process and observation models as:

$$x_t = f(x_{t-1}, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \delta_t$$

where f and h can be any non-linear differentiable functions.

- We will now linearize the functions f and h around the current state estimate $\hat{x}_t$

# Extended Kalman Filter (EKF)

- The Kalman filter requires linear process and observation models.
- In most cases in robotics we have non-linear relationships (e. g., robot moving on a 2D plane)
- We re-define the process and observation models as:

$$x_t = f(x_{t-1}, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \delta_t$$

where f and h can be any non-linear differentiable functions.

- We will now linearize the functions f and h around the current state estimate $\hat{x}_t$

# Extended Kalman Filter (EKF)

- The Kalman filter requires linear process and observation models.
- In most cases in robotics we have non-linear relationships (e.g., robot moving on a 2D plane)
- We re-define the process and observation models as:

$$x_t = f(x_{t-1}, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \delta_t$$

where f and h can be any non-linear differentiable functions.

- We will now linearize the functions f and h around the current state estimate $\hat{x}_t$

# Extended Kalman Filter (EKF)

- Predict:

$$\hat{x}_t^- = f(\hat{x}_{t-1}, u_t) \tag{16}$$

$$P_t^- = FP_{t-1}F^T + Q \tag{17}$$

- Correct:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{18}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-)) \tag{19}$$

$$P_t = (I - K_t H)P_t^- \tag{20}$$

- Here we have used the non-linear process and observation functions f and h, as well as the Jacobians F and H:

$$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}}, \; H = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t^-} \tag{21}$$

# Extended Kalman Filter (EKF)

- Predict:

$$\hat{x}_t^- = f(\hat{x}_{t-1}, u_t) \tag{16}$$

$$P_t^- = FP_{t-1}F^T + Q \tag{17}$$

- Correct:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{18}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-)) \tag{19}$$

$$P_t = (I - K_t H)P_t^- \tag{20}$$

- Here we have used the non-linear process and observation functions f and h, as well as the Jacobians F and H:

$$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}}, \; H = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t^-} \tag{21}$$

# Extended Kalman Filter (EKF)

- Predict:

$$\hat{x}_t^- = f(\hat{x}_{t-1}, u_t) \tag{16}$$

$$P_t^- = FP_{t-1}F^T + Q \tag{17}$$

- Correct:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{18}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-)) \tag{19}$$

$$P_t = (I - K_t H)P_t^- \tag{20}$$

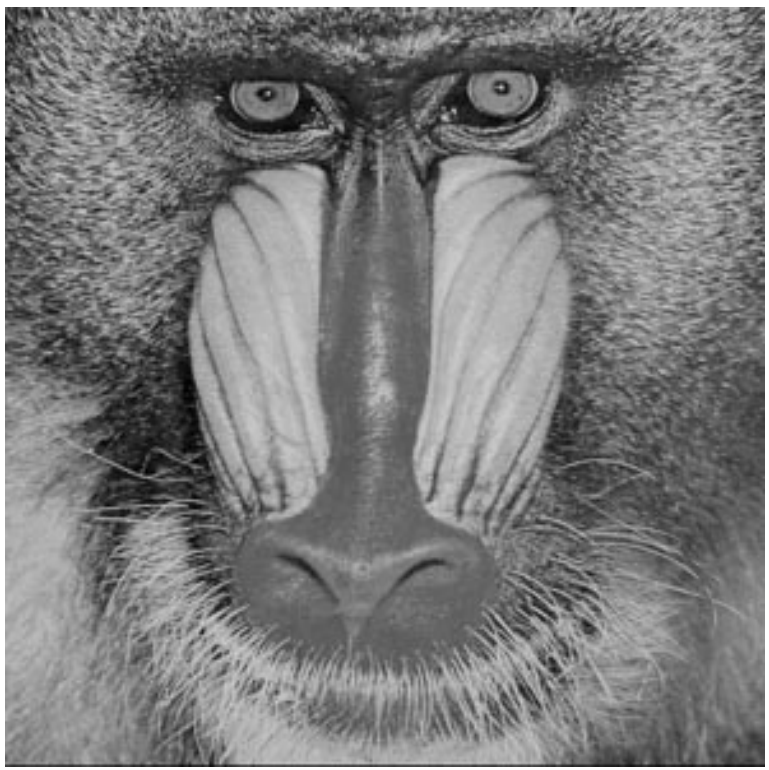
- Here we have used the non-linear process and observation functions f and h, as well as the Jacobians F and H:

$$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}}, \; H = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t^-} \tag{21}$$

# Outline

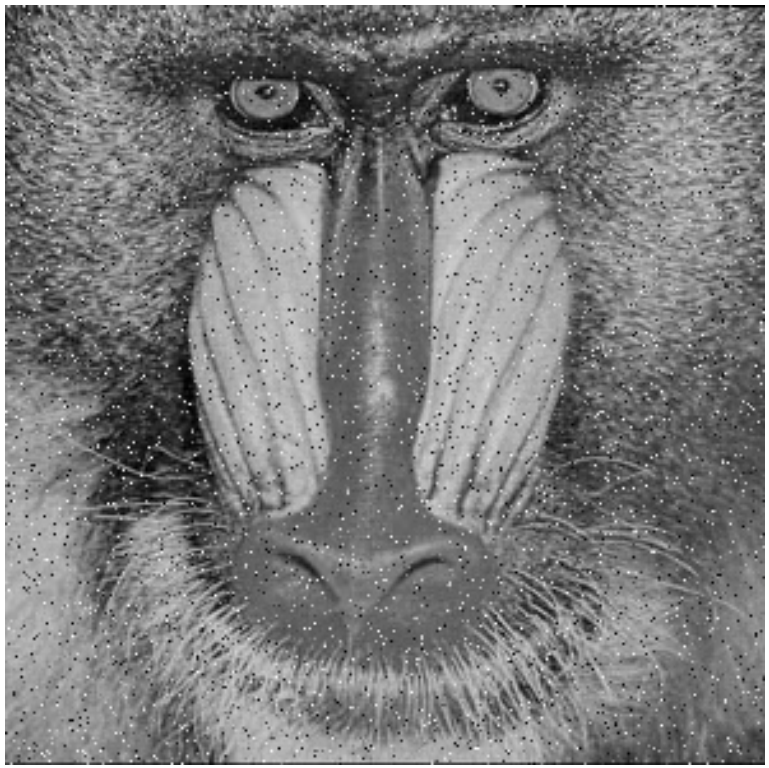
# Typical Noise in Images

- Apart from errors due to lens distortion, images usually corrupted by additional noise sources.
  - Additive Gaussian noise (independent per pixel)
  - Salt-and-pepper random noise
  - Multiplicative shot noise
- Images are often post-processed to filter out noise.



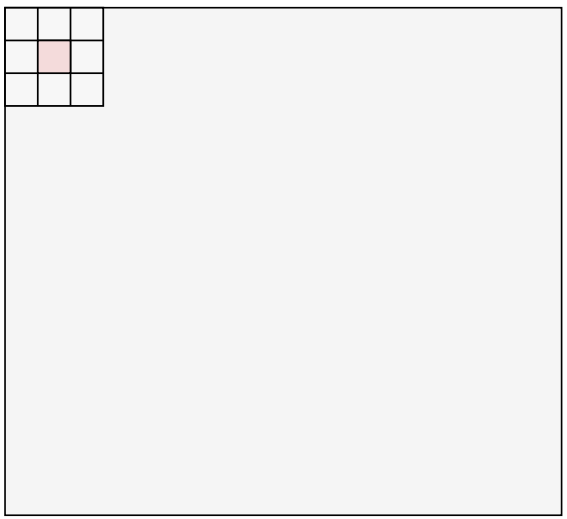
# Typical Noise in Images

- Apart from errors due to lens distortion, images usually corrupted by additional noise sources.
  - Additive Gaussian noise (independent per pixel)
  - Salt-and-pepper random noise
  - Multiplicative shot noise
- Images are often post-processed to filter out noise.



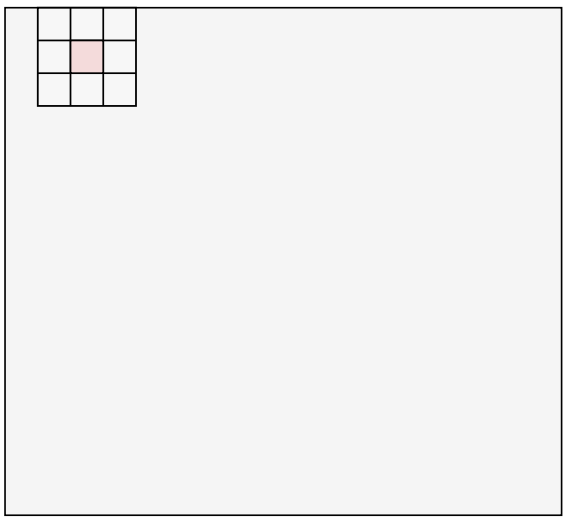
# Filter Kernel Convolution

- Spatial domain image filters work by convolution of a filter kernel (or mask).
  - As in 1D case: slide kernel over input.
  - Each element of the mask contains a weight
  - The value of the filtered pixel $p(x, y) = \sum^{i,j} w_{i,j} p_{x+i,y+j}$
  - Special care should be taken at the borders.



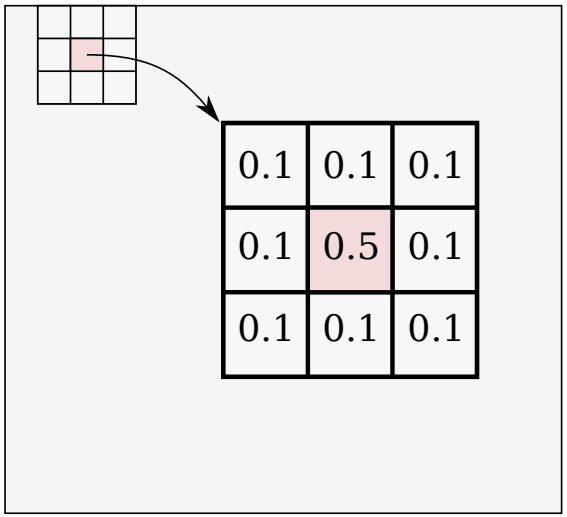
# Filter Kernel Convolution

- Spatial domain image filters work by convolution of a filter kernel (or mask).
- As in 1D case: slide kernel over input.
- Each element of the mask contains a weight
- The value of the filtered pixel $p(x, y) = \sum^{i,j} w_{i,j} p_{x+i,y+j}$
- Special care should be taken at the borders.



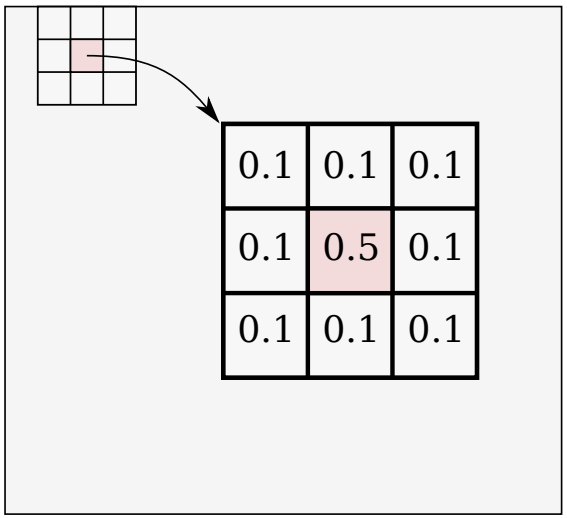
# Filter Kernel Convolution

- Spatial domain image filters work by convolution of a filter kernel (or mask).
- As in 1D case: slide kernel over input.
- Each element of the mask contains a weight
- The value of the filtered pixel $p(x, y) = \sum^{i,j} w_{i,j} p_{x+i,y+j}$
- Special care should be taken at the borders.



# Filter Kernel Convolution

- Spatial domain image filters work by convolution of a filter kernel (or mask).
- As in 1D case: slide kernel over input.
- Each element of the mask contains a weight
- The value of the filtered pixel $p(x, y) = \sum^{i,j} w_{i,j} p_{x+i,y+j}$
- Special care should be taken at the borders.
  - shrink output image by half kernel size
  - pad input image with zeros
  - pad with copies of border pixels



# Mean filter

- Mean (average, box) filter places an equal weight for all elements of the kernel.

$$W(i,j) = \frac{1}{n_i n_j} \tag{22}$$

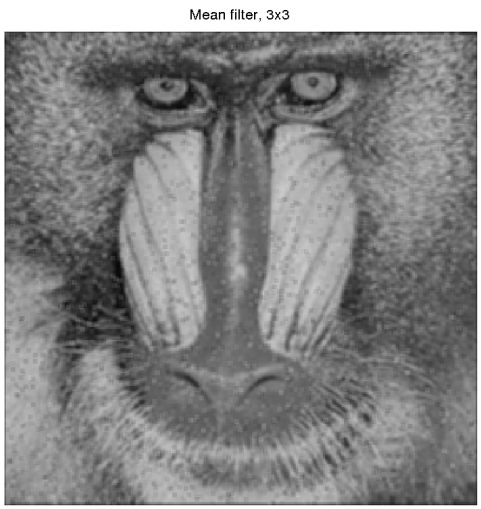
- where the kernel has size $n_\times n_j$
- Averaging blurs out both noise and details in the image.
- Generates defects, e.g. ringing, axis-aligned streaks.

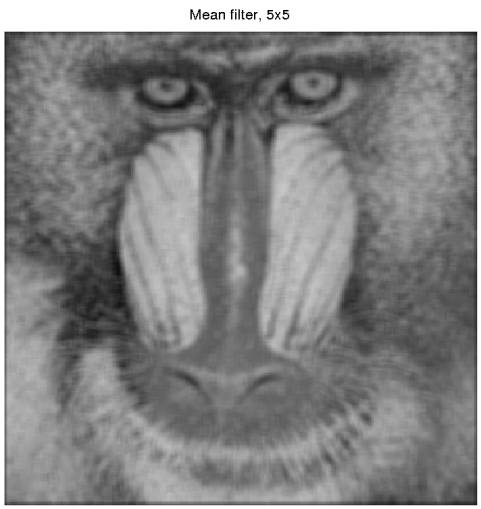
# Mean filter

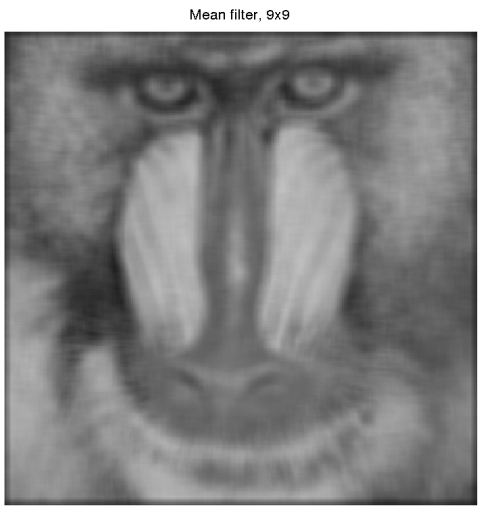
- Mean (average, box) filter places an equal weight for all elements of the kernel.

$$W(i,j) = \frac{1}{n_i n_j} \tag{22}$$

- where the kernel has size $n_\times n_j$
- Averaging blurs out both noise and details in the image.
- Generates defects, e.g. ringing, axis-aligned streaks.



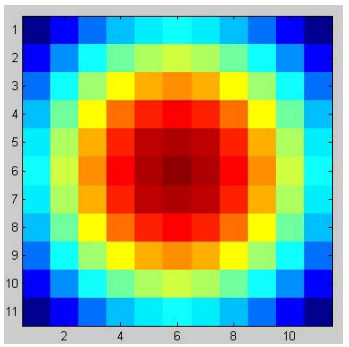



# Gaussian filter

- Uses a Gaussian distribution kernel.
- The isotropic kernel is formulated as:

$$W(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \qquad (23)$$

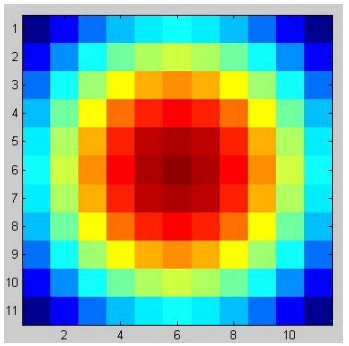



- Close pixels contribute more to the final result.
- Blurs and smoothens images.

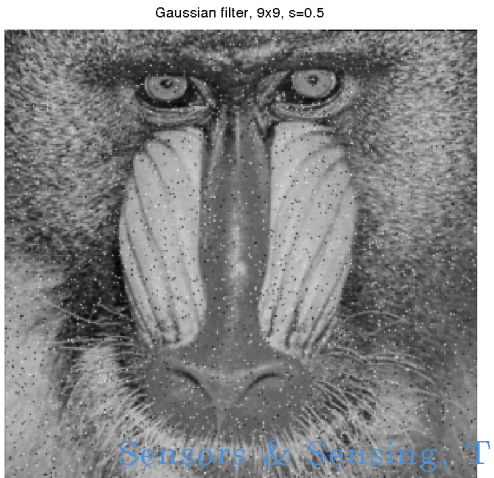
# Gaussian filter

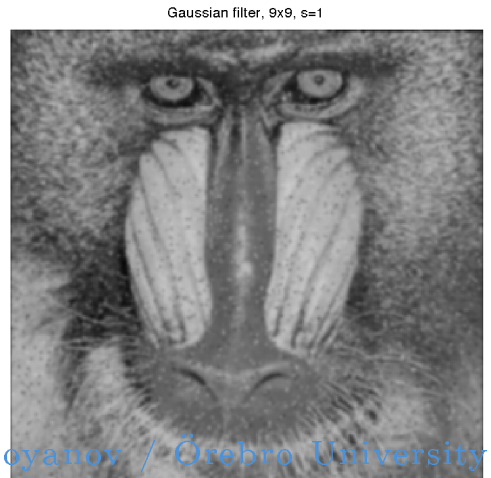
- Uses a Gaussian distribution kernel.
- The isotropic kernel is formulated as:

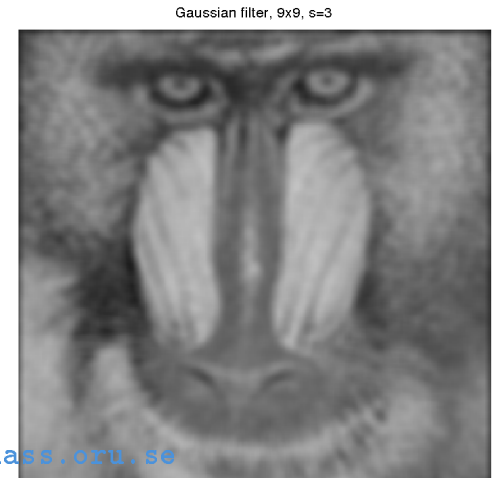
$$W(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \qquad (23)$$




- Close pixels contribute more to the final result.
- Blurs and smoothens images.



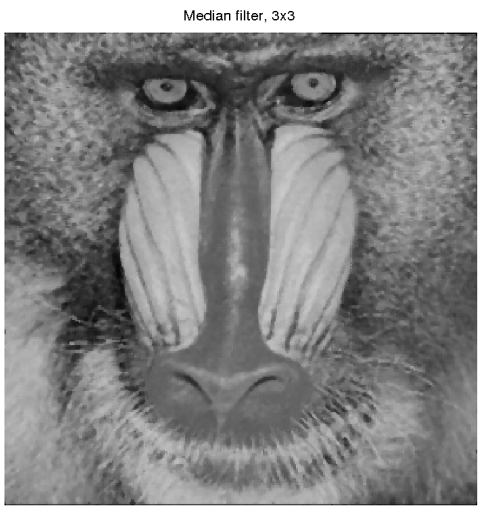



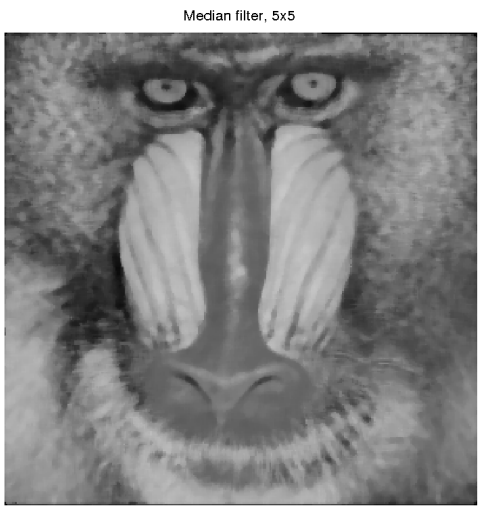
# Median filter

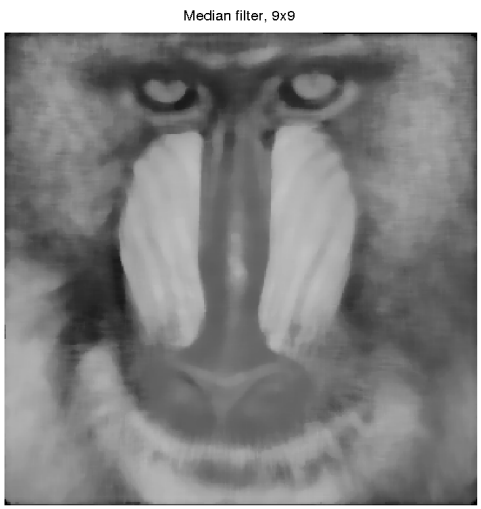
- Non-linear filter.
- Substitute pixel p with the median of all pixels inside the filter kernel.
- e.g. for a 3x3 filter, sort values and take the 5th largest as the median.
- Less blurry, very good for removing salt and pepper noise.

# Median filter

- Non-linear filter.
- Substitute pixel p with the median of all pixels inside the filter kernel.
- e.g. for a 3x3 filter, sort values and take the 5th largest as the median.
- Less blurry, very good for removing salt and pepper noise.



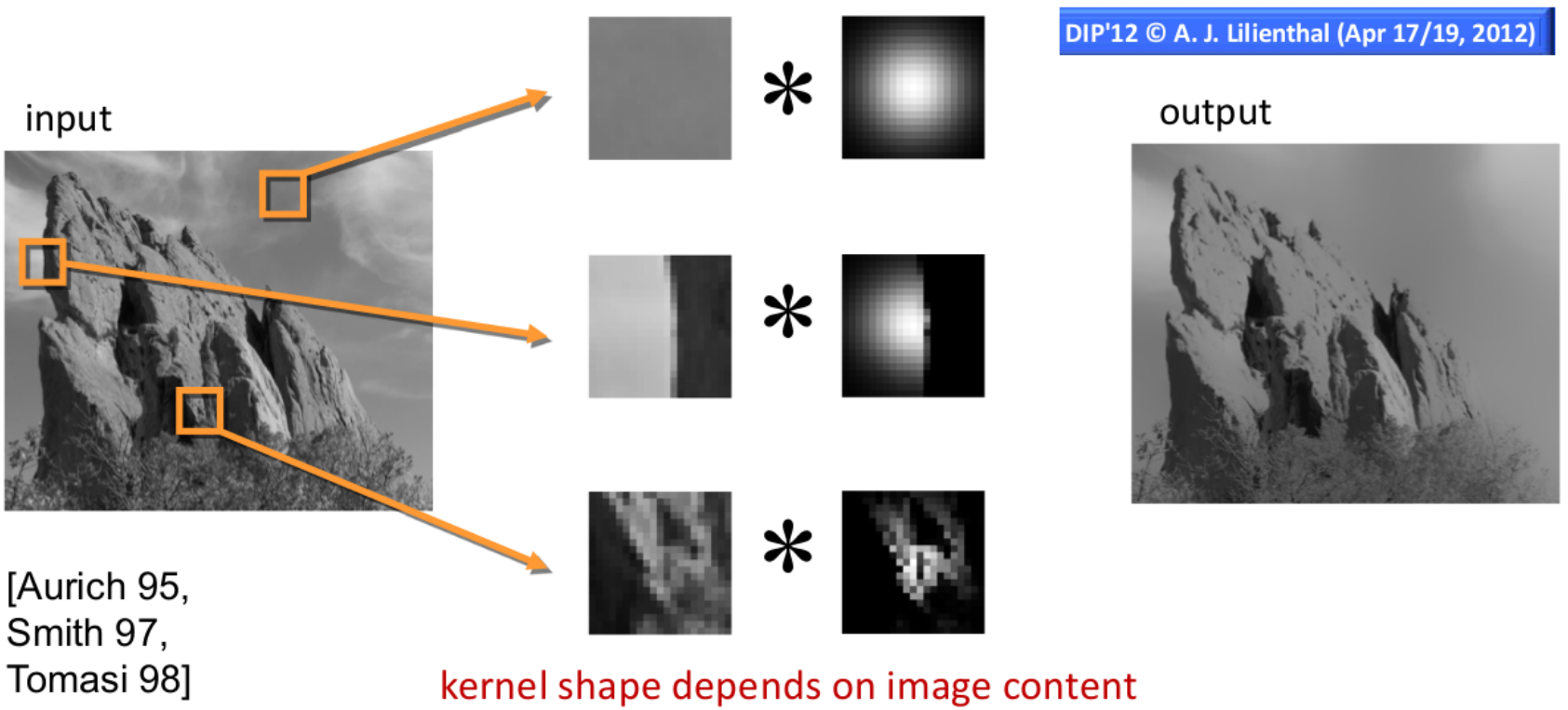
# Bilateral filter

- The bilateral filter is an edge-preserving smoothing filter.
- Main idea: pixels are smoothed based on both spatial proximity (x, y coordinates) and the pixel values p.
- Two Gaussian kernels, one for spatial- and one for pixel-domain.
- The kernel is formulated as:

$$W(i,j) = \frac{1}{w_n} e^{-\frac{i^2+j^2}{2\sigma_d^2} - \frac{\|I_W - I(i,j)\|^2}{2\sigma_r^2}} \tag{24}$$

- $w_n$ is a normalizing factor
- $I_W$ is the intensity of the image pixel at the currently evaluated x, y
- $I(i,j)$ is the intensity of the image at $x+i, y+j$

# Bilateral filter



# Bilateral filter

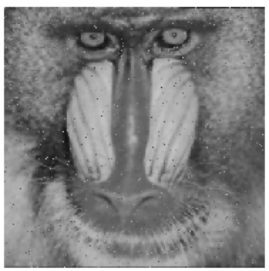
w=0.1 s=0.5

w=0.3 s=0.5

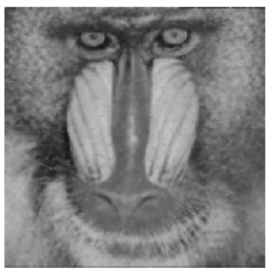
w=0.5 s=0.5

w=0.1 s=1.5

w=0.3 s=1.5

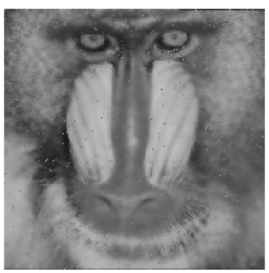


w=0.5 s=1.5



w=0.1 s=2.5

w=0.3 s=2.5



w=0.5 s=2.5

# Outline

# Filtering Noise

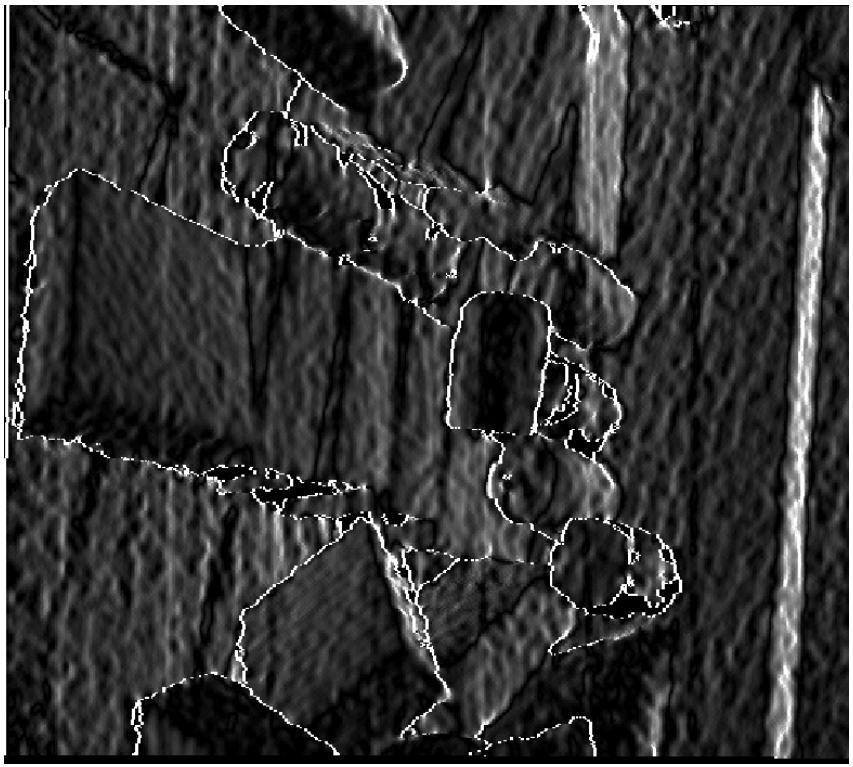
- 3D range sensors usually have internal noise filtering routines.
- Algorithms to estimate reliability of measurements, e.g:
  - For stereo cameras: detect lack of texture, detect spurious measurements, detect connected components.
  - For TOF: integrate multiple measurements, detect low intensity values.
- These algorithms are usually parametric, part of tuning your setup is setting parameters.
- Post-processing steps for noise filtering may be applied.

# Depth Map Filters

- Basic idea: apply a filter on the depth map to reduce noise.
- Examples: Gaussian / mean filter, median filter, bilateral filter, TV L1 robust filter.

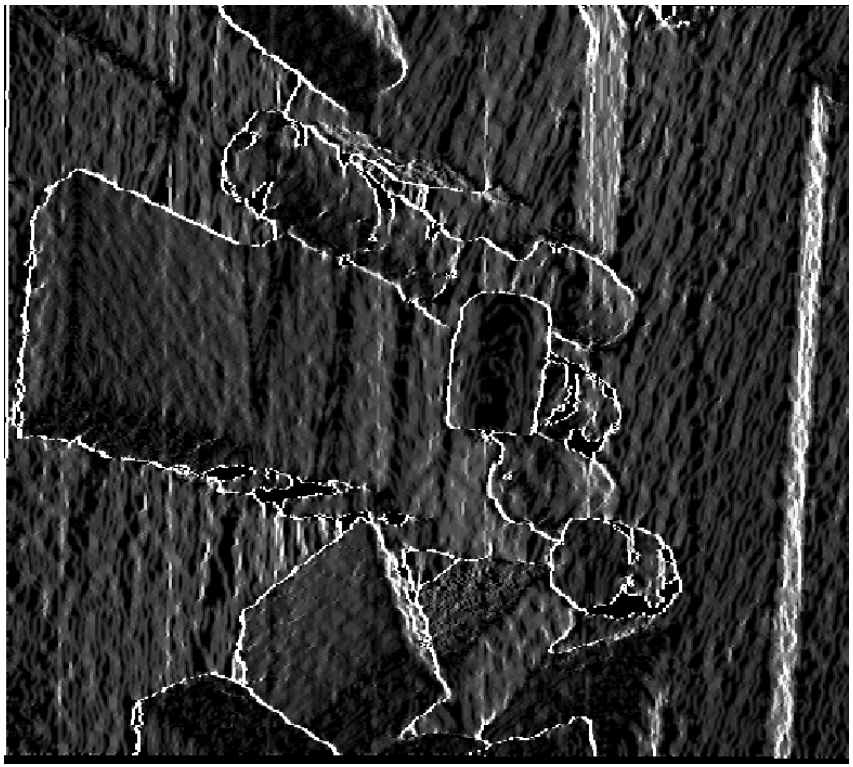
# Depth Map Filters

- Basic idea: apply a filter on the depth map to reduce noise.
- Examples: Gaussian / mean filter, median filter, bilateral filter, TV L1 robust filter.



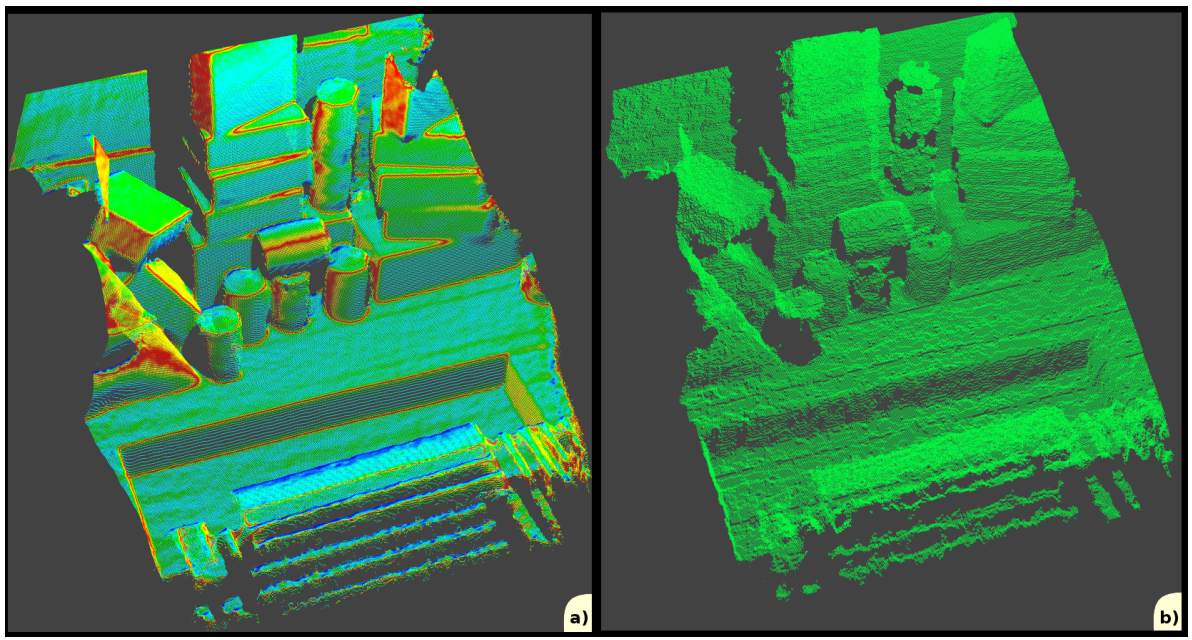
# Depth Map Filters

- Basic idea: apply a filter on the depth map to reduce noise.
- Examples: Gaussian / mean filter, median filter, bilateral filter, TV L1 robust filter.



# Volumetric Filtering

- Basic idea: apply a filter in 3D space.
- Advantage: use multiple viewing angles to reduce noise.
- Example: render measurements in an occupancy grid.



# Outline

# Sample Questions

True/False questions:

a: The bilateral filter is an edge-preserving smoothing filter.

# Sample Questions

Design/Derive questions:

a: How would the kernels of a mean, median and Gaussian filter look like for a 3x3 filter mask? Draw the kernels when possible. Assume a $\sigma = 0.5$ for the Gaussian filter. (3 points)

b: What effect do you expect to see from each filter? (3 point)

c: Apply the three filters to the grayscale image below, using the shrink image border handling strategy. What possible strategies are there for handling the borders and what are their pro's and con's? (3 points).

| | | | |
|---|---|---|---|
| 10 | 15 | 250 | 251 |
| 14 | 18 | 25 | 255 |
| 18 | 25 | 28 | 241 |

# Sample Questions

In this task you have a robot moving on a 2 dimensional xy plane. You can control the speeds $u_x, u_y$ of the robot. In addition, you can observe the x and y distance to a set of unique landmarks in the environment.

a: Formulate a Kalman filter for tracking the state of the robot $(x, y)$ (2 points).

# Sample Questions

In this task you have a robot moving on a 2 dimensional xy plane. You can control the speeds $u_x, u_y$ of the robot. In addition, you can observe the x and y distance to a set of unique landmarks in the environment.

b: Assume you have a noise with variance of 0.4 on the $u_x, u_y$ transition, and a noise with variance 0.1 on the x, y landmark observations. Your robot starts at $(0, 0)$, with an initial variance estimate of $P_0 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}$ and moves with control inputs u at $t_1 = (0.8, 1.2)$ and $t_2 = (0.6, 0.6)$. You are observing one landmark, located at $(1, 1)$. You observe the distance to the landmark as $l_1 = (0.55, 0.1)$ and $l_2 = (0.15, 0.45)$. Perform the Kalman update steps and show the state estimate at times $t_1$ and $t_2$. Does the filter converge? (4 points)

# Sensors and Sensing
# Filtering

Todor Stoyanov

Centre for Applied Autonomous Sensor Systems
Örebro University
Sweden