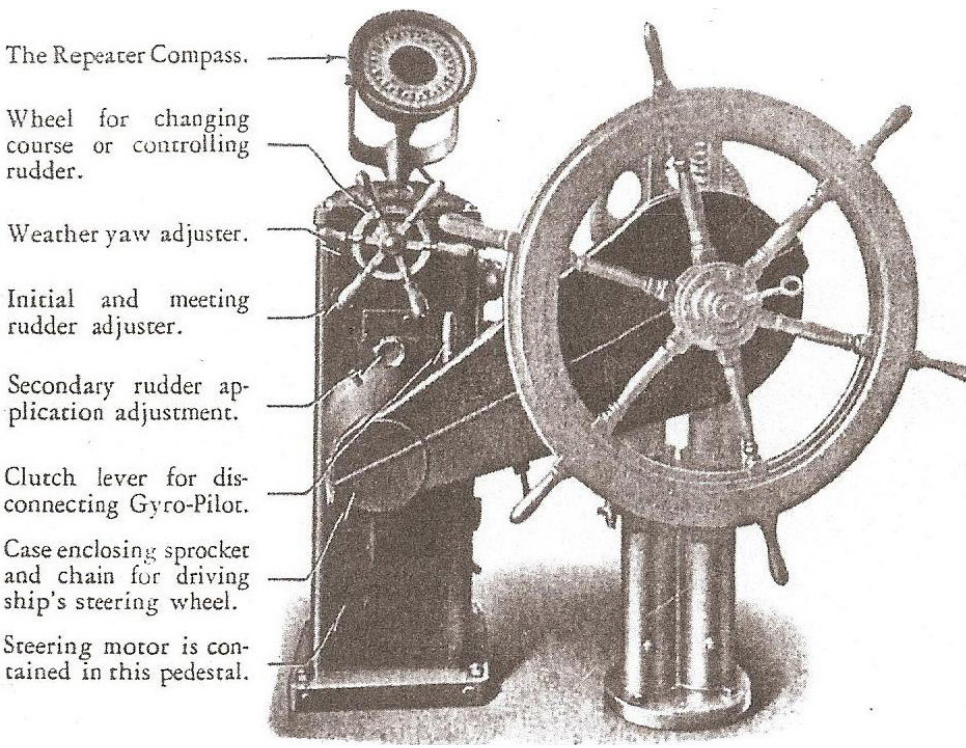
AUTONOMOUS DRIVING
a brief EE120+ level review



The Sperry gyropilot (ca. 1922)

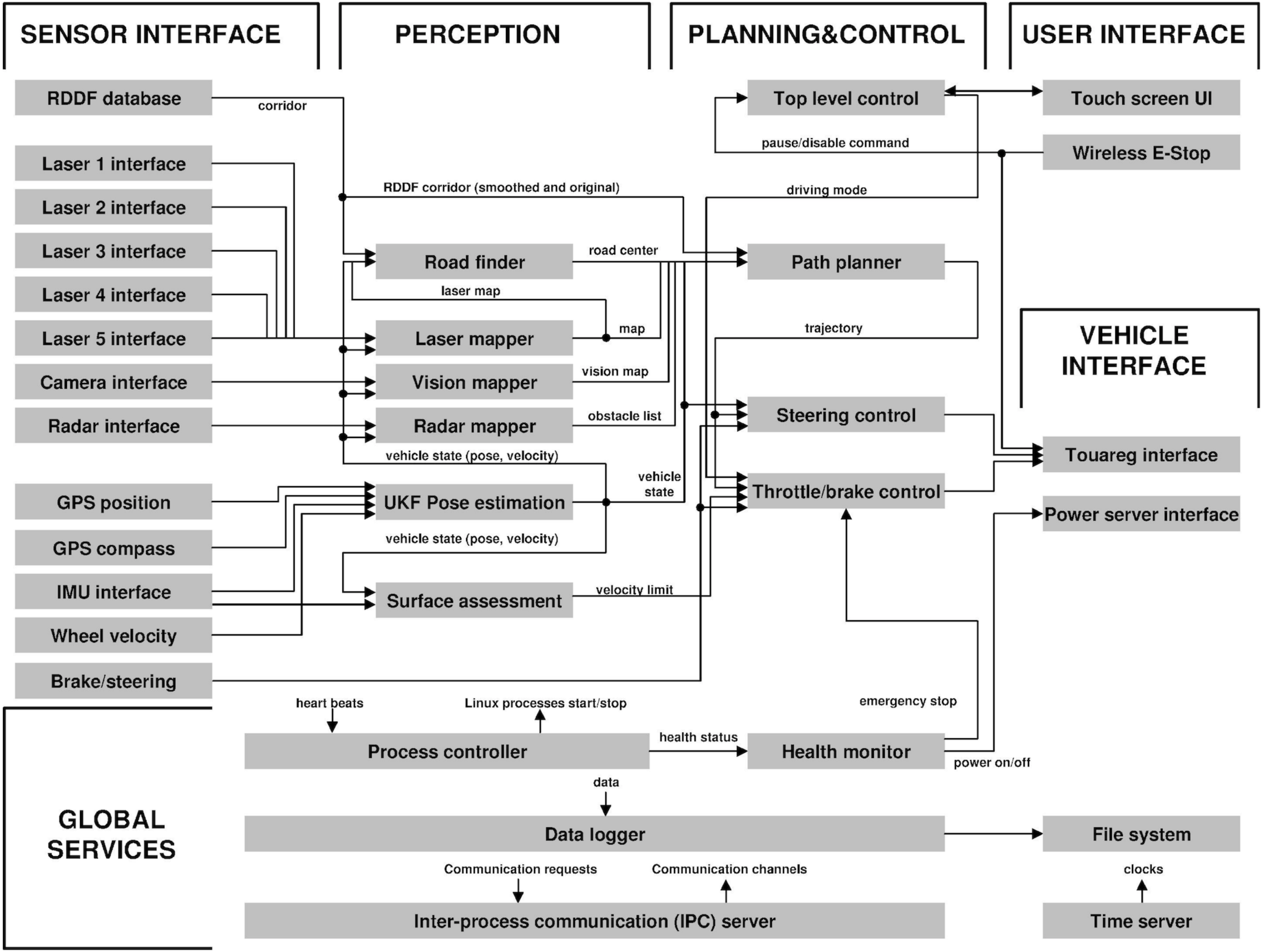
Google car (2012)

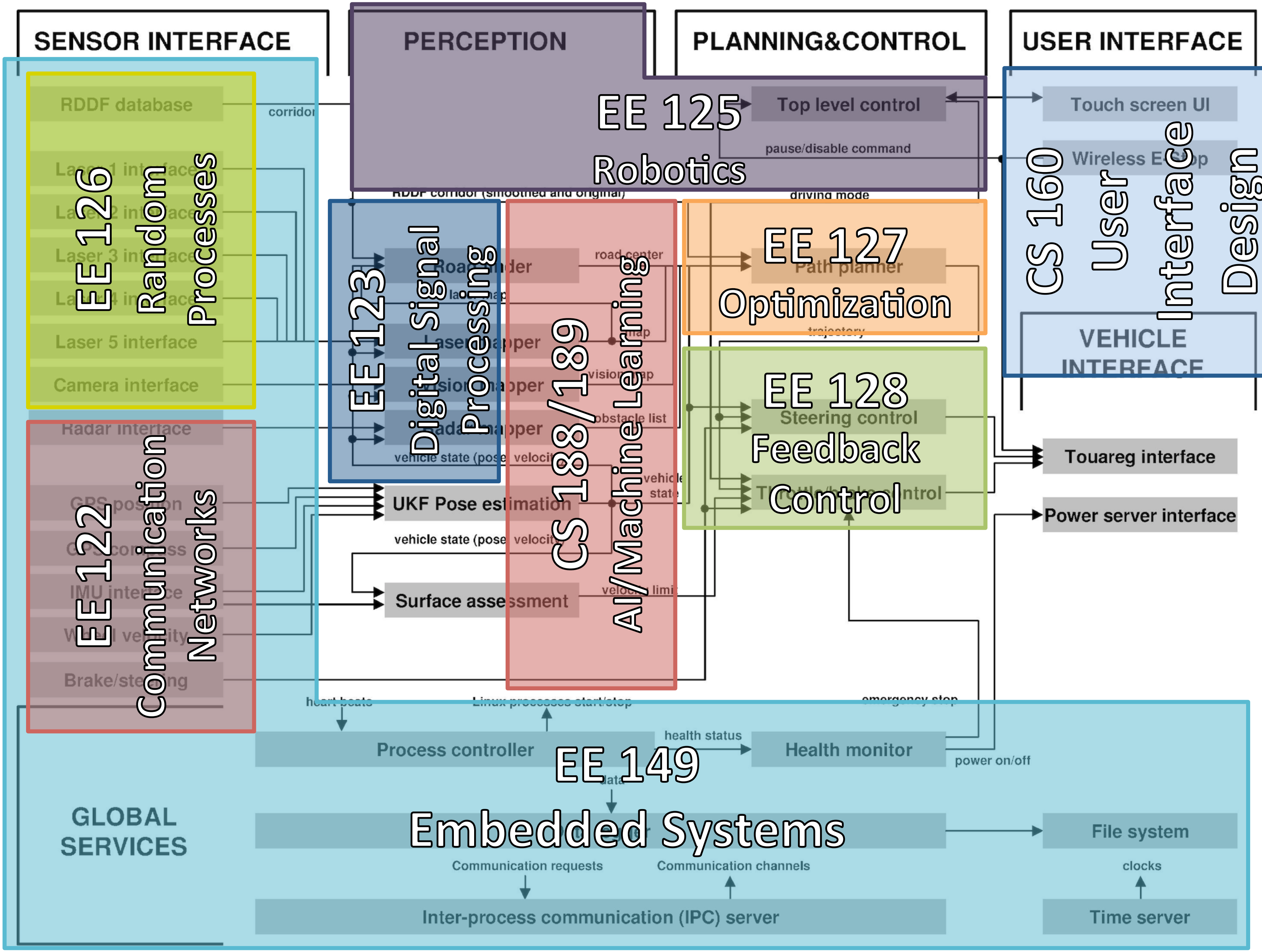
ca. 1957

The idea of self-driving cars has existed for a long time. Recent progress was enabled by computation, sensors, radar, mapping, machine learning, and catalyzed by the DARPA Grand Challenges.

Stanley: The winner of the 2005 DARPA Grand Challenge

Credit: Thrun, Journal of Field Robotics, 23(9), pp. 661-692, 2006. DOI: 10.1002/rob.20147


SENSOR INTERFACE
PERCEPTION
PLANNING&CONTROL
USER INTERFACE
RDDF database
corridor
Top level control
Touch screen UI
Laser 1 interface
pause/disable command
Wireless E-Stop
Laser 2 interface
RDDF corridor (smoothed and original)
driving mode
Laser 3 interface
Road finder
road center
Path planner
Laser 4 interface
laser map
Laser 5 interface
Laser mapper
map
trajectory
VEHICLE INTERFACE
Camera interface
Vision mapper
vision map
Radar interface
Radar mapper
obstacle list
Steering control
vehicle state (pose, velocity)
Touareg interface
vehicle state
GPS position
UKF Pose estimation
Throttle/brake control
Power server interface
GPS compass
vehicle state (pose, velocity)
IMU interface
Surface assessment
velocity limit
Wheel velocity
Brake/steering
heart beats
Linux processes start/stop
emergency stop
Process controller
health status
Health monitor
power on/off
data
GLOBAL SERVICES
Data logger
File system
Communication requests
Communication channels
clocks
Inter-process communication (IPC) server
Time server


SENSOR INTERFACE
PERCEPTION
PLANNING&CONTROL
USER INTERFACE
EE 125
Robotics
Top level control
pause/disable command
Touch screen UI
Wireless E-stop
CS 160
User Interface Design
VEHICLE INTERFACE
RDDF database
corridor
EE 126
Random Processes
Laser 5 interface
Camera interface
Radar interface
RDDF corridor (smoothed and original)
driving mode
EE 123
Digital Signal Processing
CS 188/189
AI/Machine Learning
road center
vision map
obstacle list
EE 127
Optimization
Path planner
trajectory
EE 128
Feedback Control
Steering control
Touareg interface
Power server interface
EE 122
Communication Networks
Brake/steering
UKF Pose estimation
vehicle state (pose, velocity)
Surface assessment
heart beats
Linux processes start/stop
emergency stop
Process controller
health status
Health monitor
power on/off
EE 149
Embedded Systems
GLOBAL SERVICES
File system
Communication requests
Communication channels
clocks
Inter-process communication (IPC) server
Time server

Development steps – automated driving

Degree of automation

- Single sensor
- Sensor-data fusion
- Sensor-data fusion + map

ACC/lane keeping support

Only longitudinal or lateral control

Integrated cruise assist

Partially automated longitudinal and lateral guidance in driving lane
Speed range 0-130 kph

Highway assist

Partially automatic longitudinal and lateral guidance

Lane change after driver confirmation

Supervision of surrounding traffic (next lane, ahead, behind)

Highway pilot

Highly automated longitudinal and lateral guidance with lane changing

Reliable environment recognition, including in complex driving situations

No permanent supervision by driver

Auto pilot

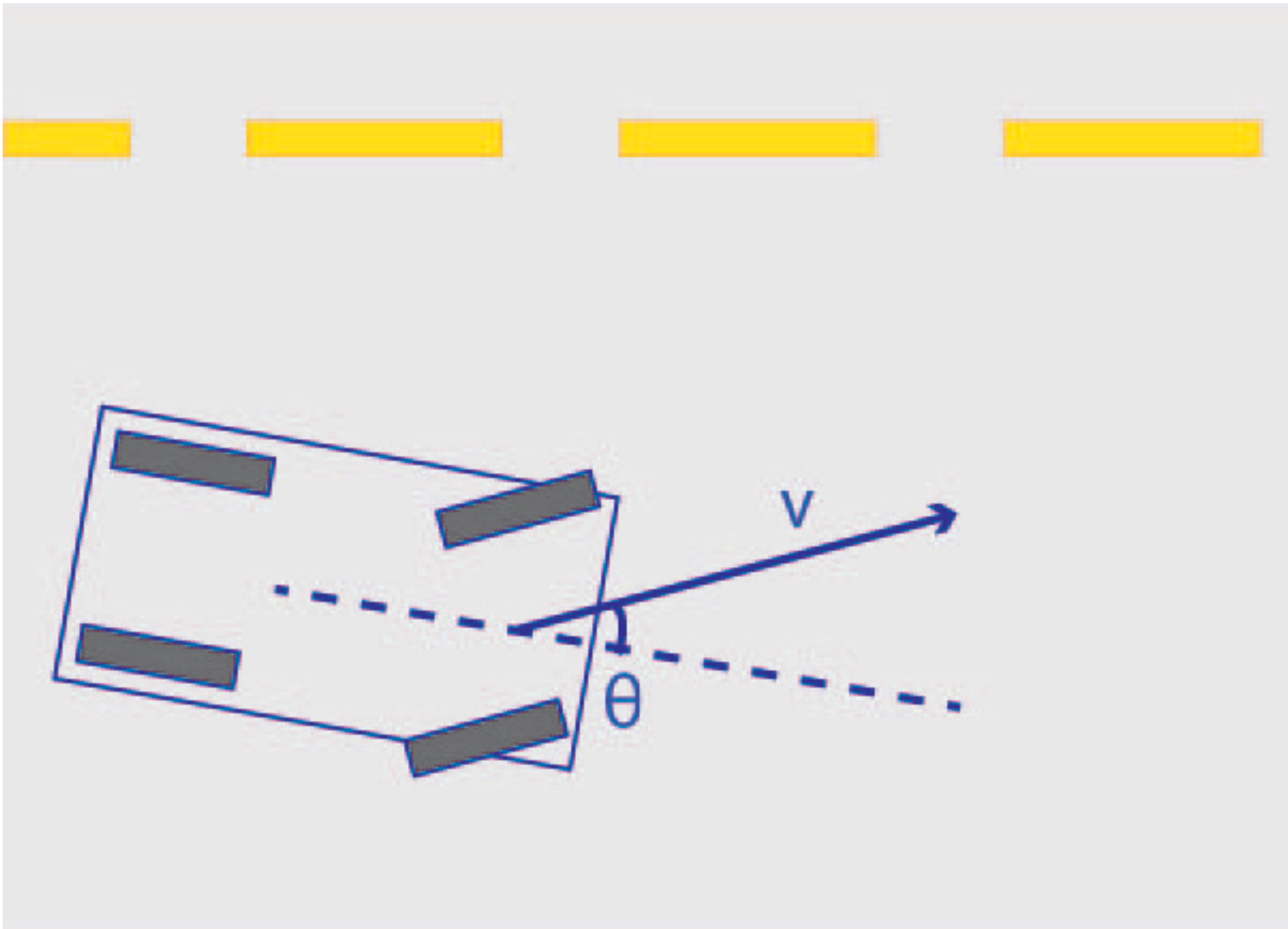
Door-to-door commuting (e.g. to work) in urban traffic

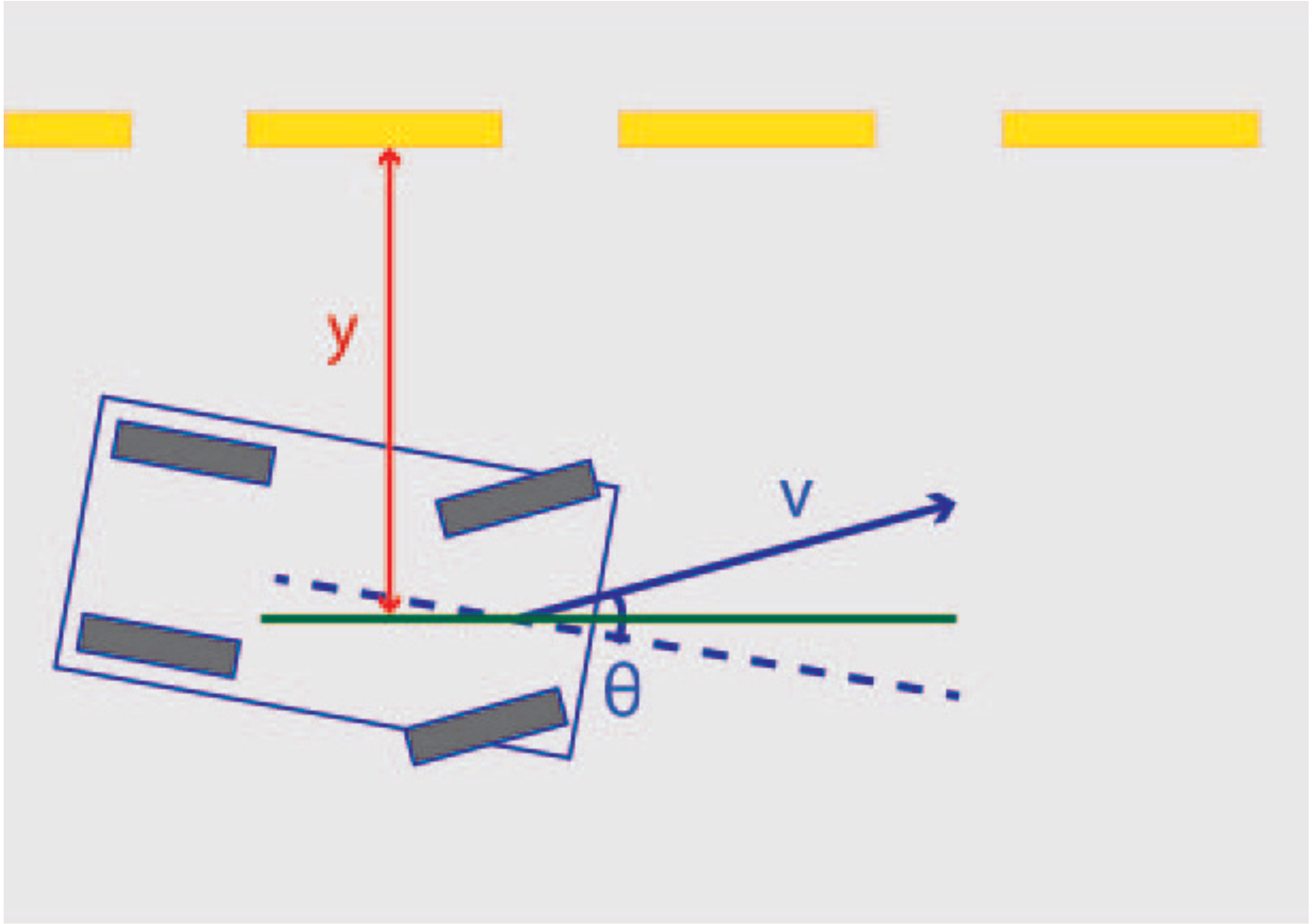
Strictest safety requirements

No supervision by driver

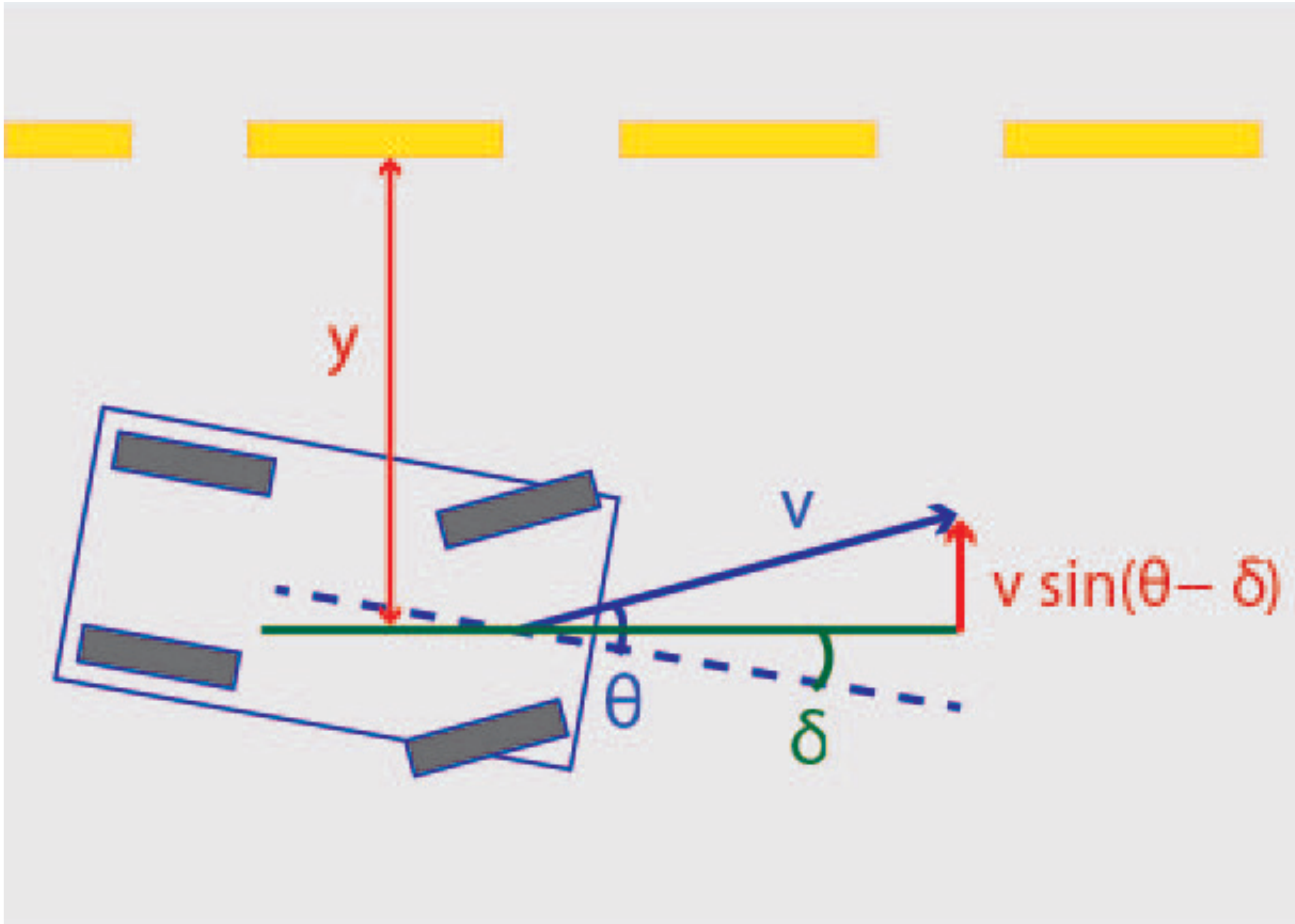
BOSCH

Topic 1: Steering Control





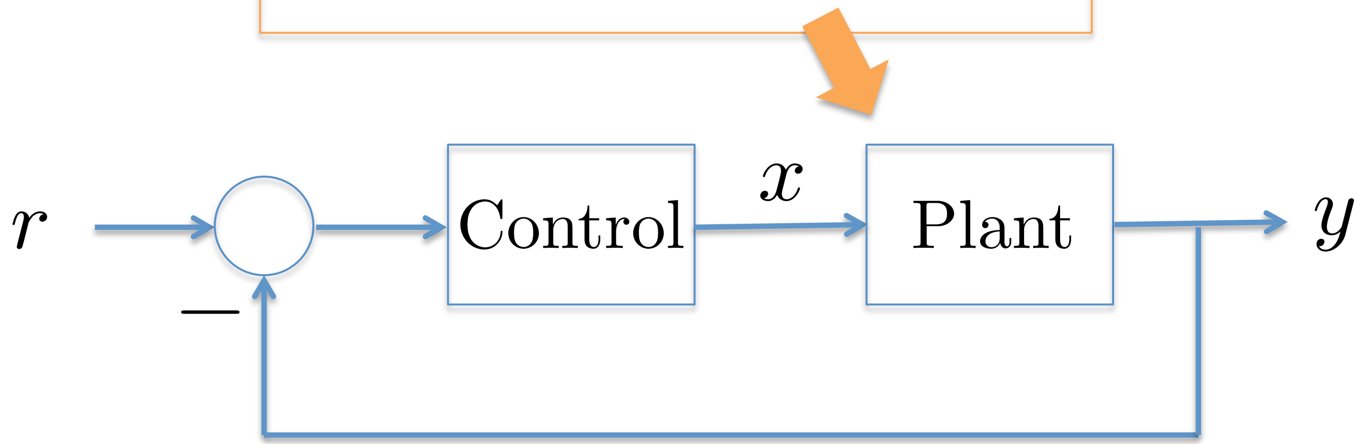
Goal: Maintain $y(t)$ at desired value



$$\frac{dy(t)}{dt} = -v\sin(\theta - \delta)$$

Define $x \triangleq \delta - \theta$. Then, the 'plant' is:

$$\frac{dy(t)}{dt} = v\sin(x(t)) \qquad \text{(nonlinear!)}$$



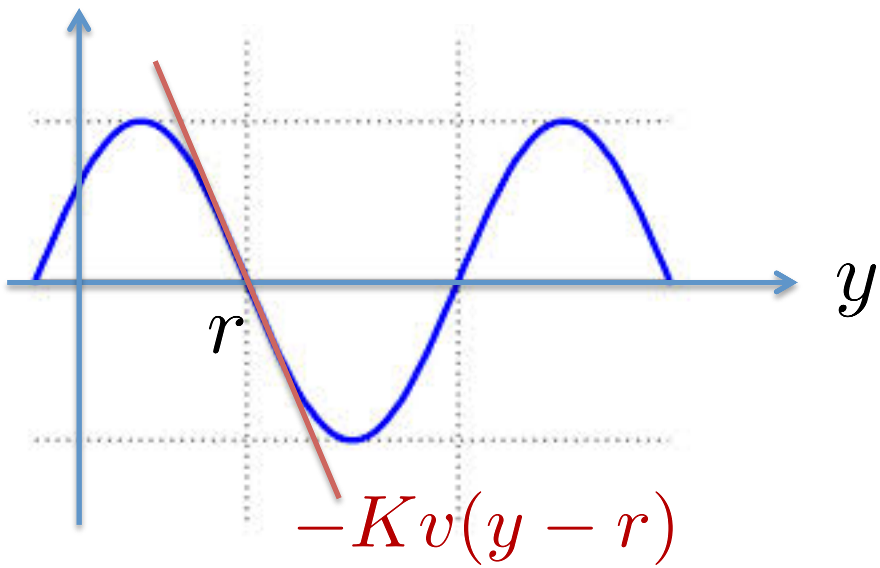
First, try the constant gain control

$$x = K(r - y)$$

Closed-loop: $$\frac{dy}{dt} = v\sin(K(r-y))$$

Near $y = r$,

$$\frac{dy}{dt} = v\sin(K(r-y)) \approx -Kv(y-r)$$



Therefore, for constant r:

$$y(t) - r \approx (y(0) - r)e^{-Kvt}$$

if $y(0) - r$ small

Root Locus Interpretation

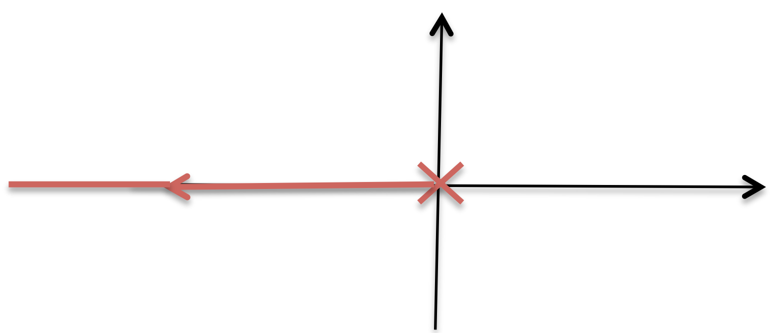
Linearized plant model:

$$\frac{dy}{dt} = v\sin(x) \approx vx$$

$$H_p(s) = \frac{v}{s}$$

Closed-loop poles are the roots of:

$$1 + KH_p(s) = 0 \quad \Rightarrow \quad s = -Kv$$



Steady-state error $= 0$ ($H_p(s)$ has pole at 0)

The controller above does not obey the physical steering limit: $|x| < 90°$

Instead, try the nonlinear controller:

$$x = \tan^{-1}(K(r-y))$$

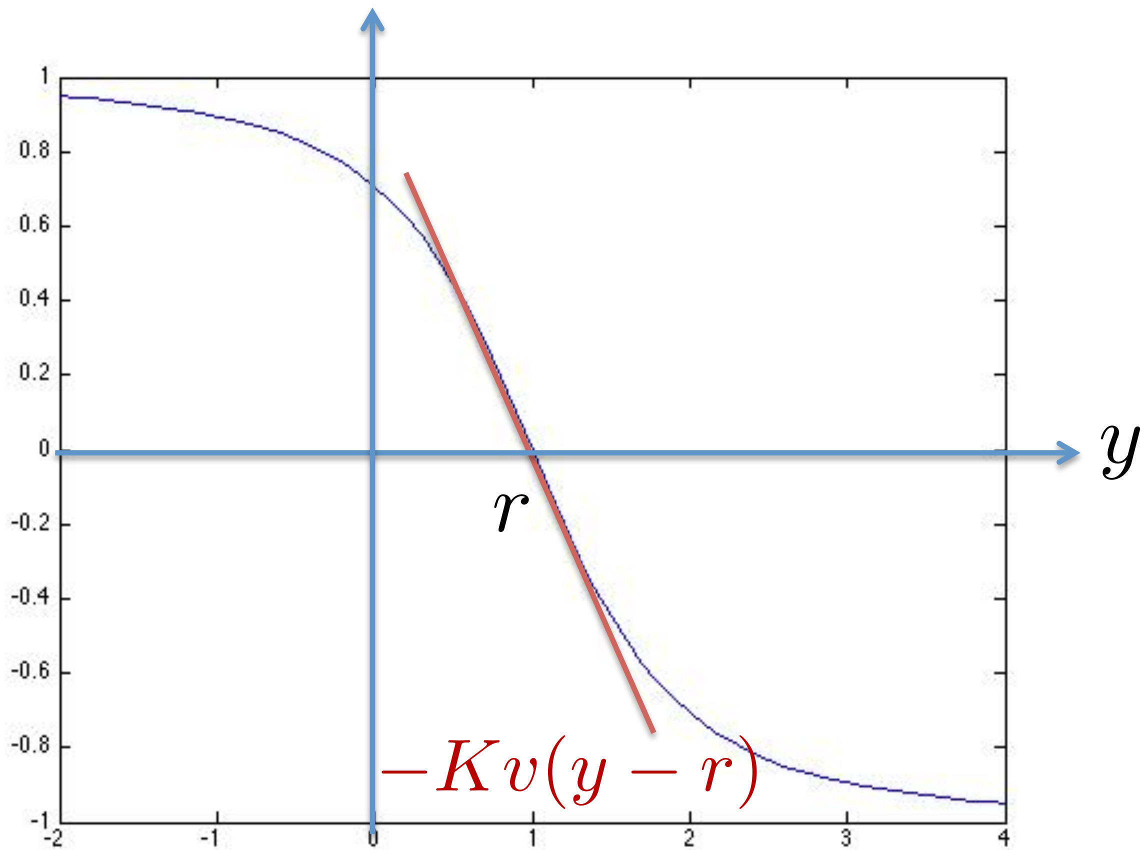
Closed-loop:

$$\frac{dy}{dt} = v \sin(\tan^{-1}(K(r-y)))$$

Substitute: $\sin(\tan^{-1}(u)) = \frac{u}{\sqrt{1+u^2}}$

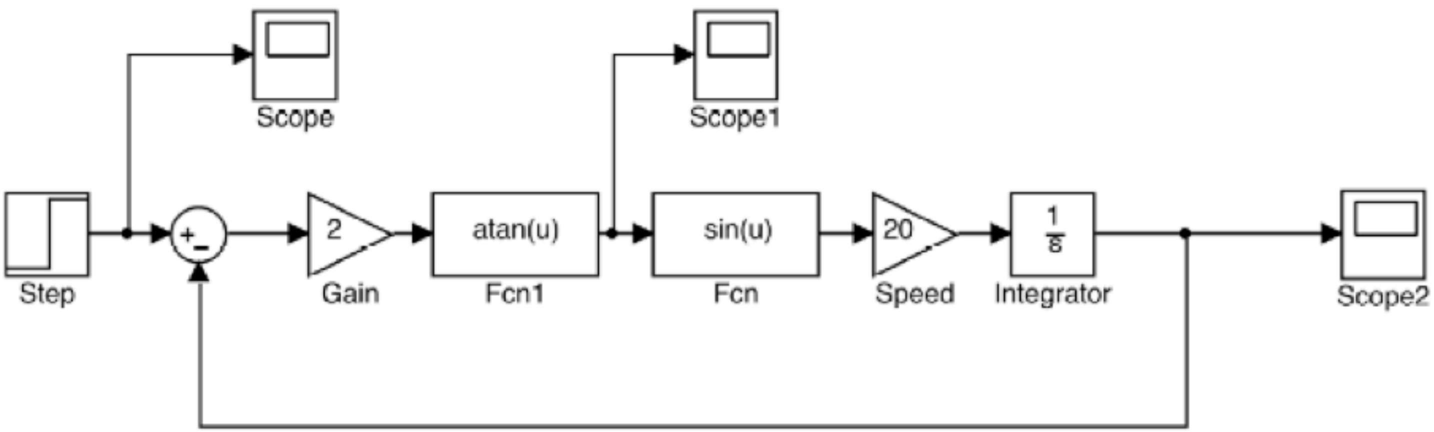
$$\frac{dy}{dt} = v \frac{K(r-y)}{\sqrt{1+K^2(r-y)^2}}$$

$$\frac{dy}{dt} = v\frac{K(r-y)}{\sqrt{1+K^2(r-y)^2}}$$

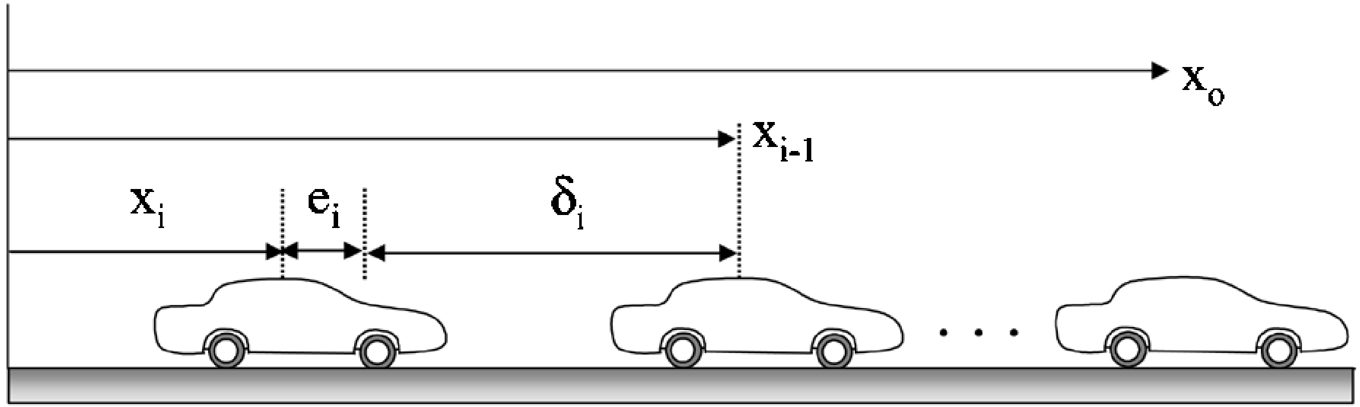


$$y(t) \to r \ \text{ for all } y(0)$$

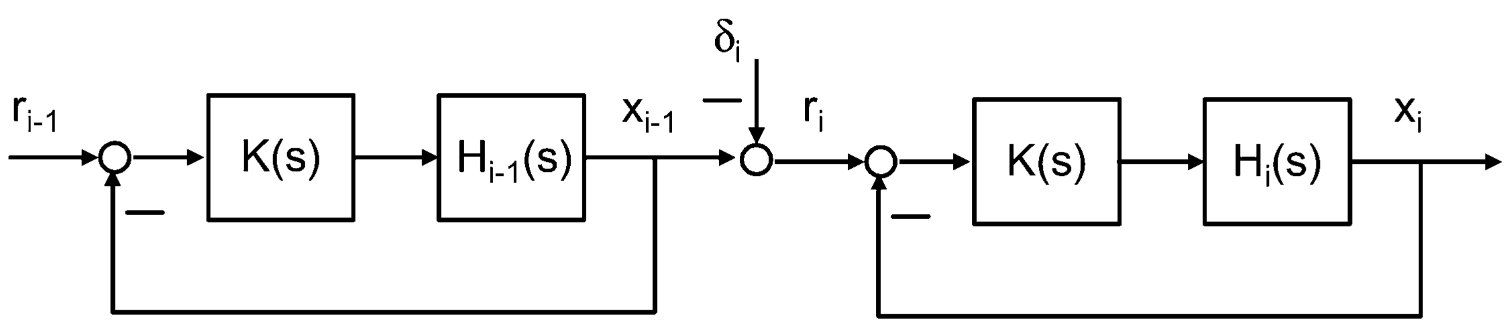
Simulink Diagram:



Topic 2: Vehicle Following



Goal: Maintain the gap $x_i - x_{i-1}$ at δ_i; thus the reference for vehicle i is $r_i = x_{i-1} - \delta_i$



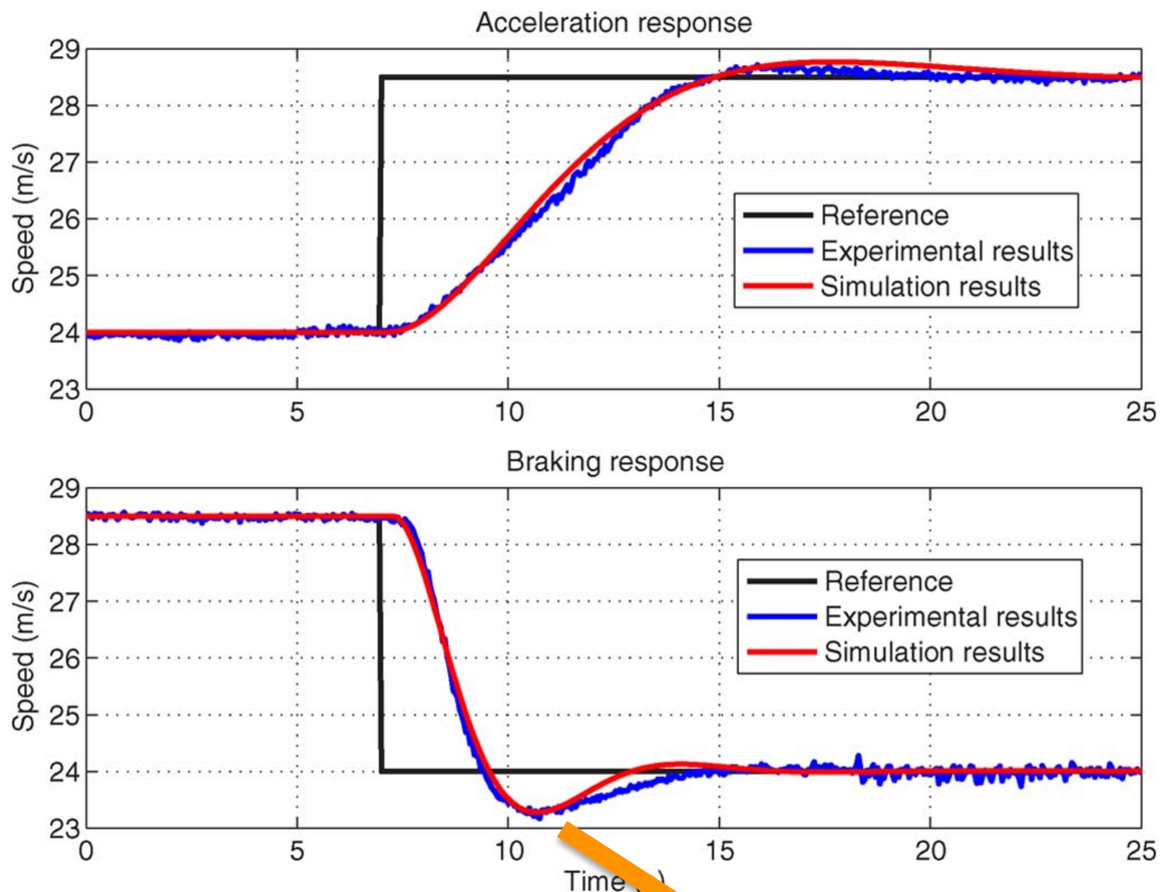
Credit: (Seiler et al., 2004)

Vehicle Model

$$H(s) = e^{-T_d s} \underbrace{\underbrace{\frac{k}{s^2 + 2\zeta\omega_n s + \omega_n^2}}_{\triangleq\, H_v(s)\ \text{(velocity)}} \frac{1}{s}}_{\text{position}}$$

Model validated and parameters identified for experimental vehicles by PATH (Partners for Advanced Transit and Highways) researchers at UC Berkeley

	k	ζ	ω_n	T_d
Accelerating	0.156	0.661	0.396	0.146
Braking	1.136	0.5	1.067	0.287



overshoot due to engine breaking

Credit: (Milanes et al., 2014)

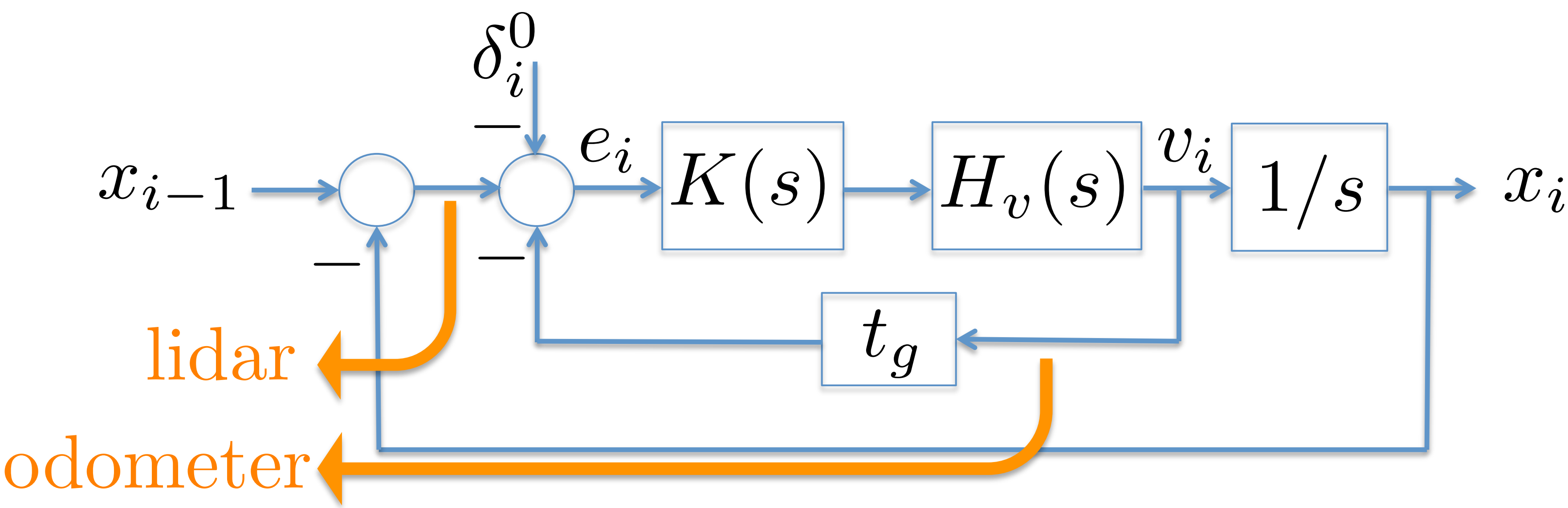
Time Gap Control

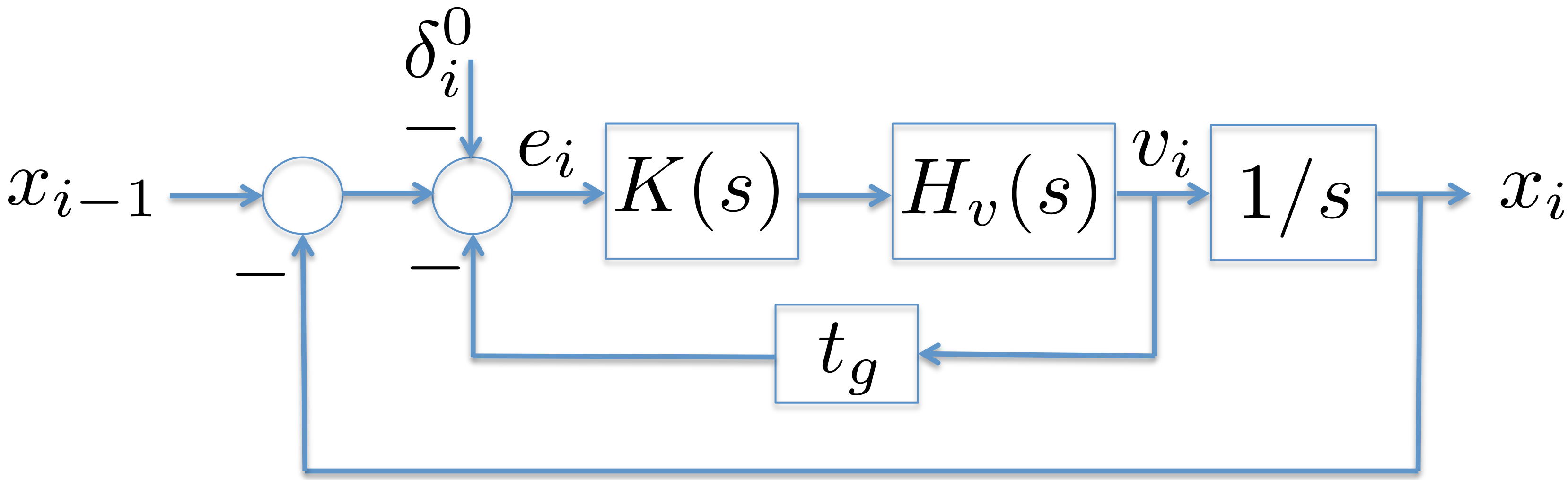
Larger δ_i with increasing speed:

$$\delta_i = t_g v_i(t) + \delta_i^0$$

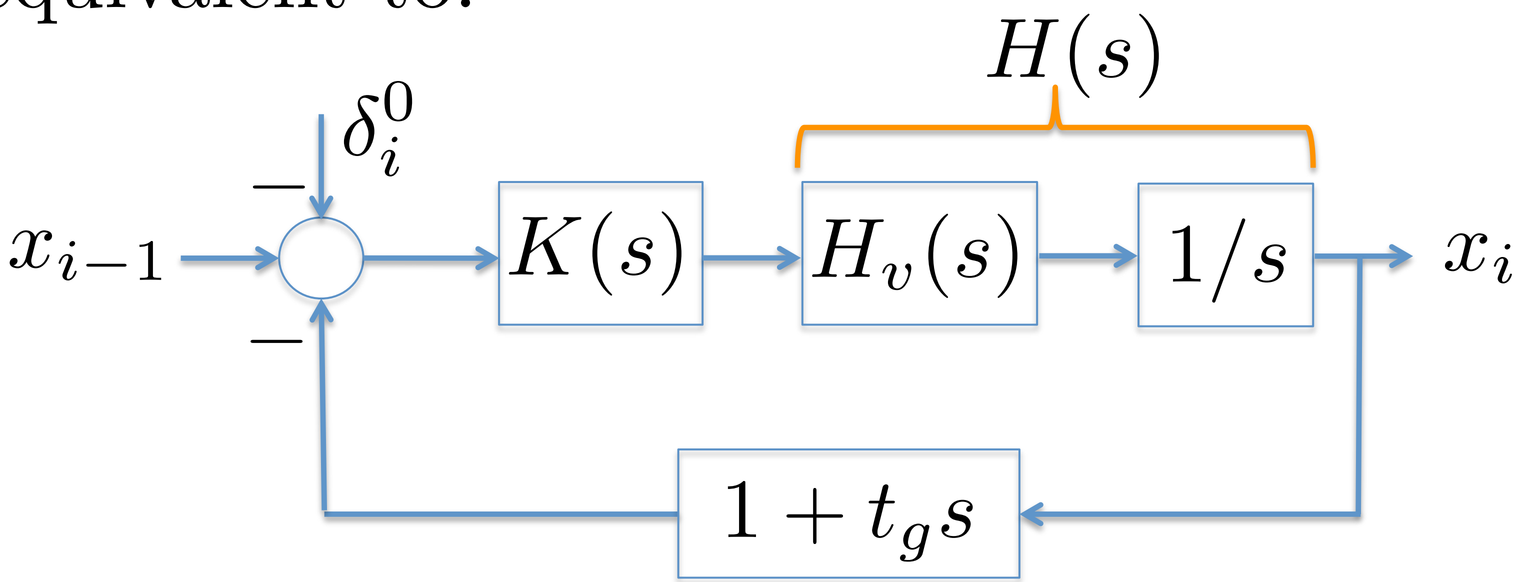
where t_g is a fixed time gap (e.g., $t_g = 1$ sec means about one vehicle length per 10 mph).

Error: $e_i = x_{i-1} - x_i - v_i t_g - \delta_i^0$





is equivalent to:



Note $t_g = 0$ recovers the fixed distance control.

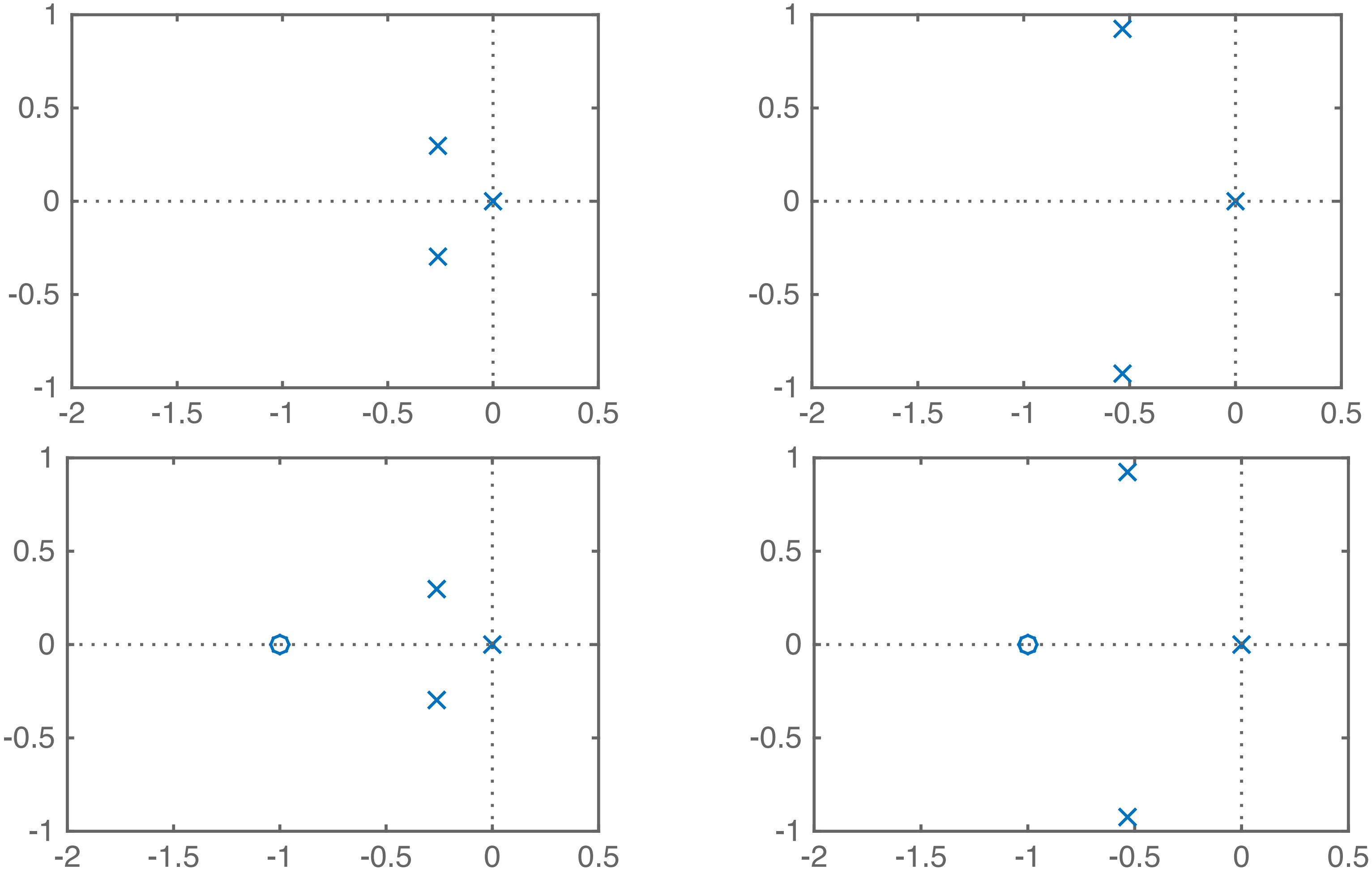
Sketch root locus for cst.gain control and $T_d = 0$

acceleration

braking

$t_g = 0$

$t_g = 1$



All-pass approximation for delay:

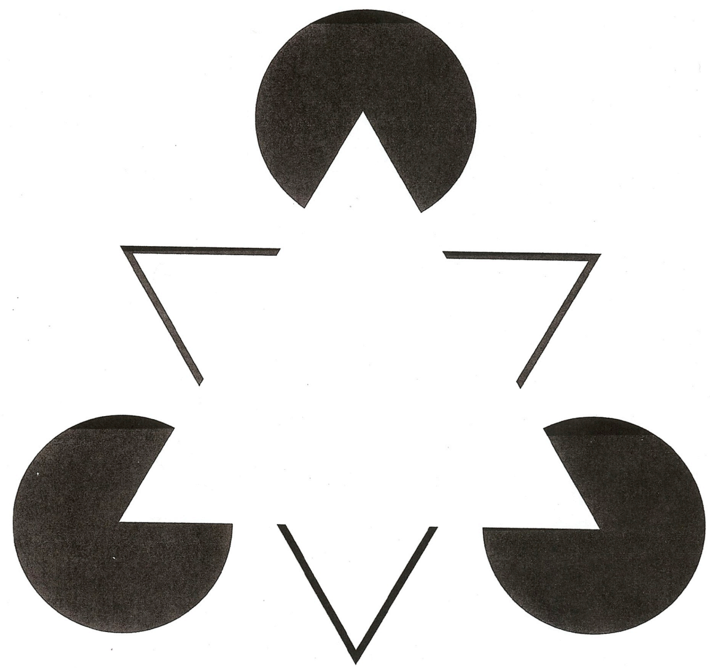
$$e^{-T_d s} \approx \frac{1 - \frac{T_d}{2} s}{1 + \frac{T_d}{2} s}$$

Augment transfer function with this all pass and design controller using Matlab rltool

Topic 3: Edge Detection

- Detection of object boundaries is critical in computer vision (e.g., detecting lane markings).
- Visual cortex detects light-dark discontinuities. The brain interpolates between contours.

Credit: (H. Cheng, 2011)

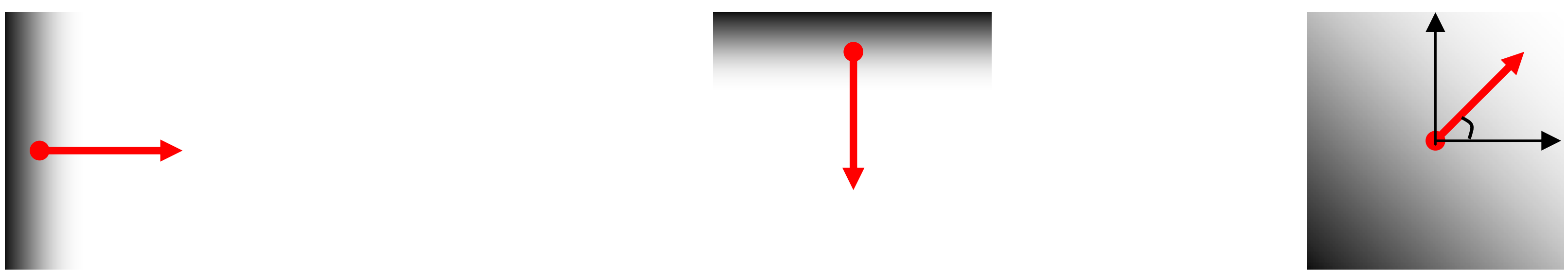


Credit: (Kanizsa, 1979)

Edge detection algorithms seek sharp gradients in the image.

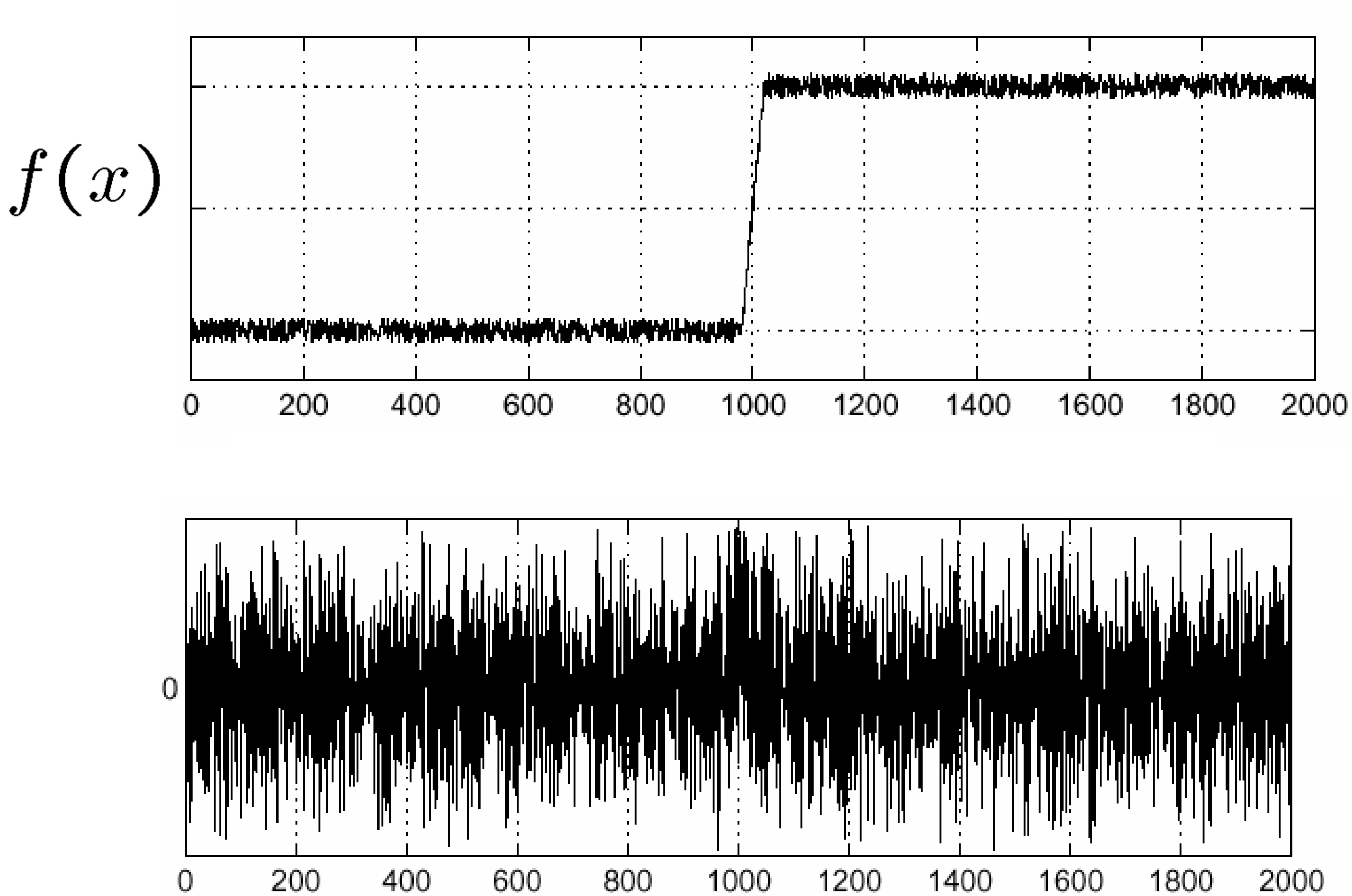
$$\nabla f(x, y) = \left[\frac{\partial f}{\partial x} \; \frac{\partial f}{\partial y} \right]$$

The gradient points in the direction of most rapid change in intensity:



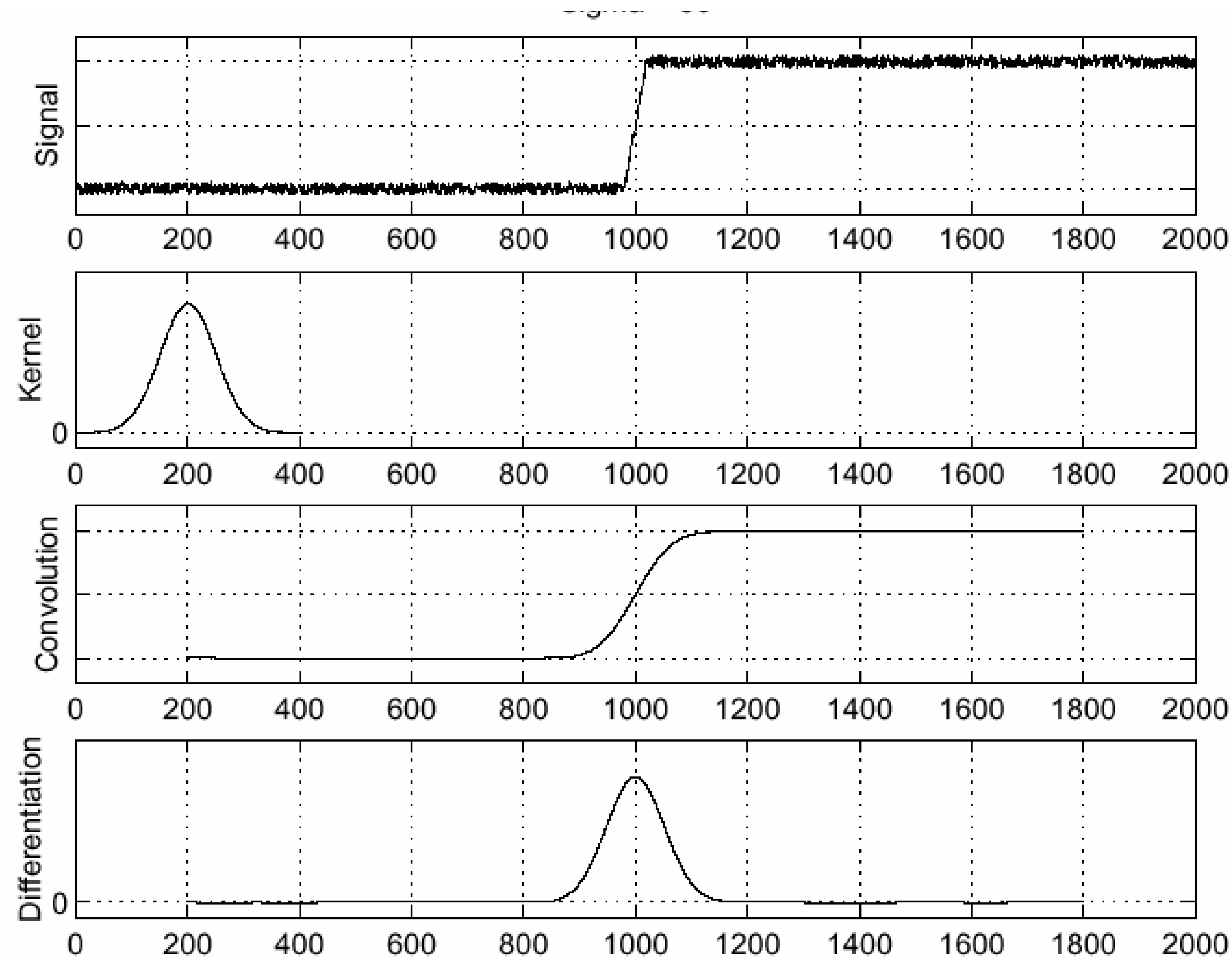
Credit: Efros, Computational Photography

Problem: Algorithms based on derivatives are sensitive to noise



Credit: Efros, Computational Photography

Solution: Low-pass filter before differentiation



Credit: Efros, Computational Photography

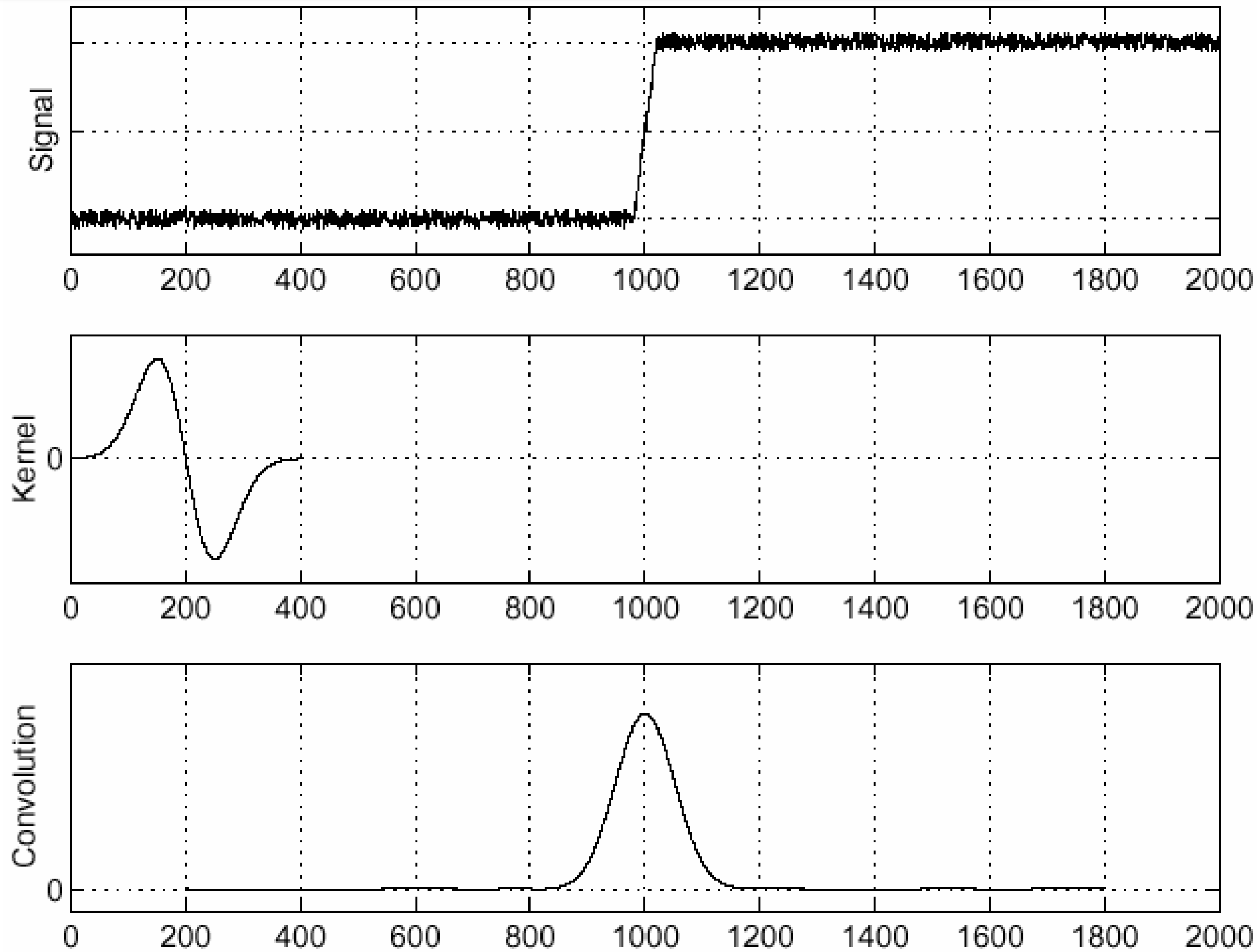
Save one operation using the property:

$$\frac{d}{dx}\{h(x) * f(x)\} = \frac{dh(x)}{dx} * f(x)$$

$f(x)$

$\frac{dh(x)}{dx}$

$\frac{dh(x)}{dx} * f(x)$



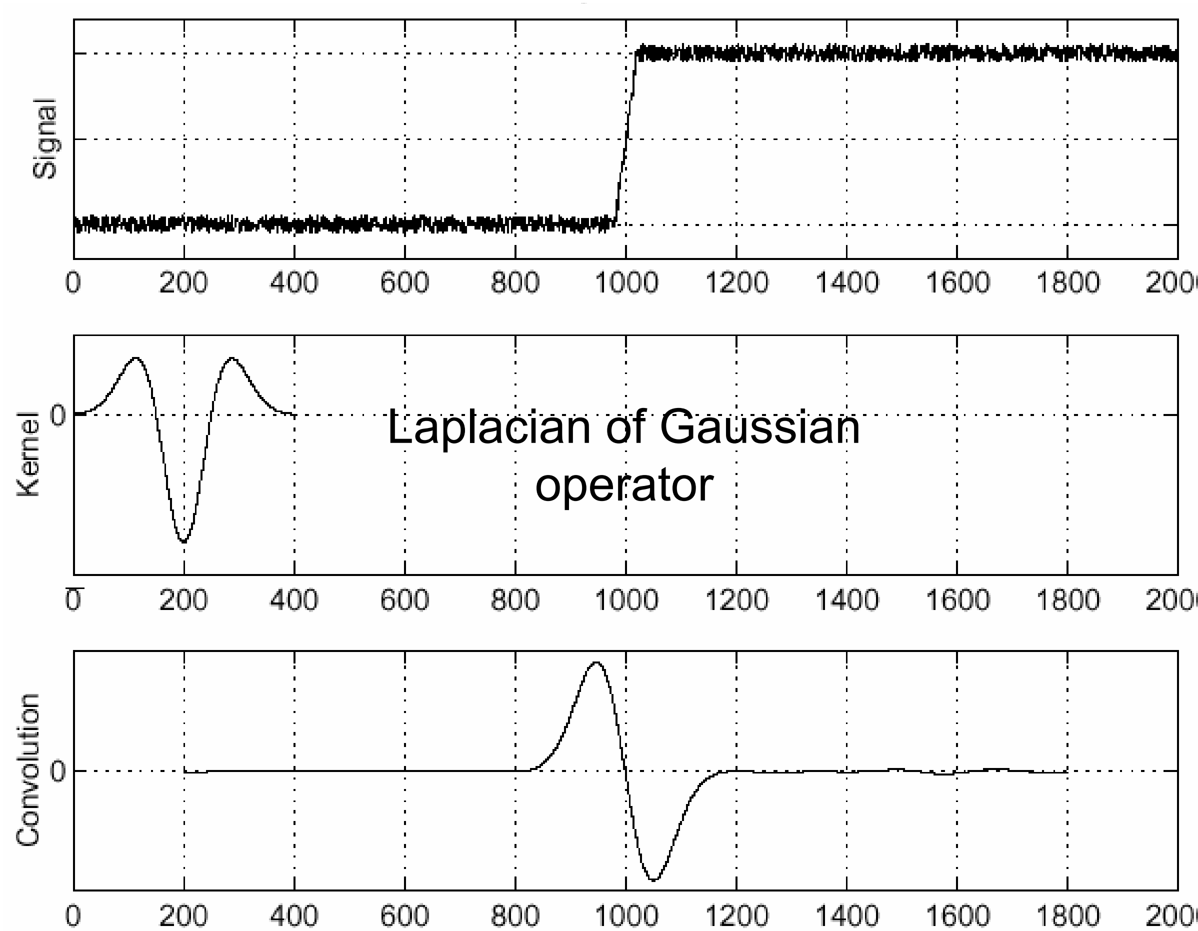
Credit: Efros, Computational Photography

The peaks of $\frac{d}{dx}\{h * f\} = \frac{dh}{dx} * f$ are the zero crossings of $\frac{d^2}{dx^2}\{h * f\} = \frac{d^2 h}{dx^2} * f$

$f(x)$

$\frac{d^2 h}{dx^2}$

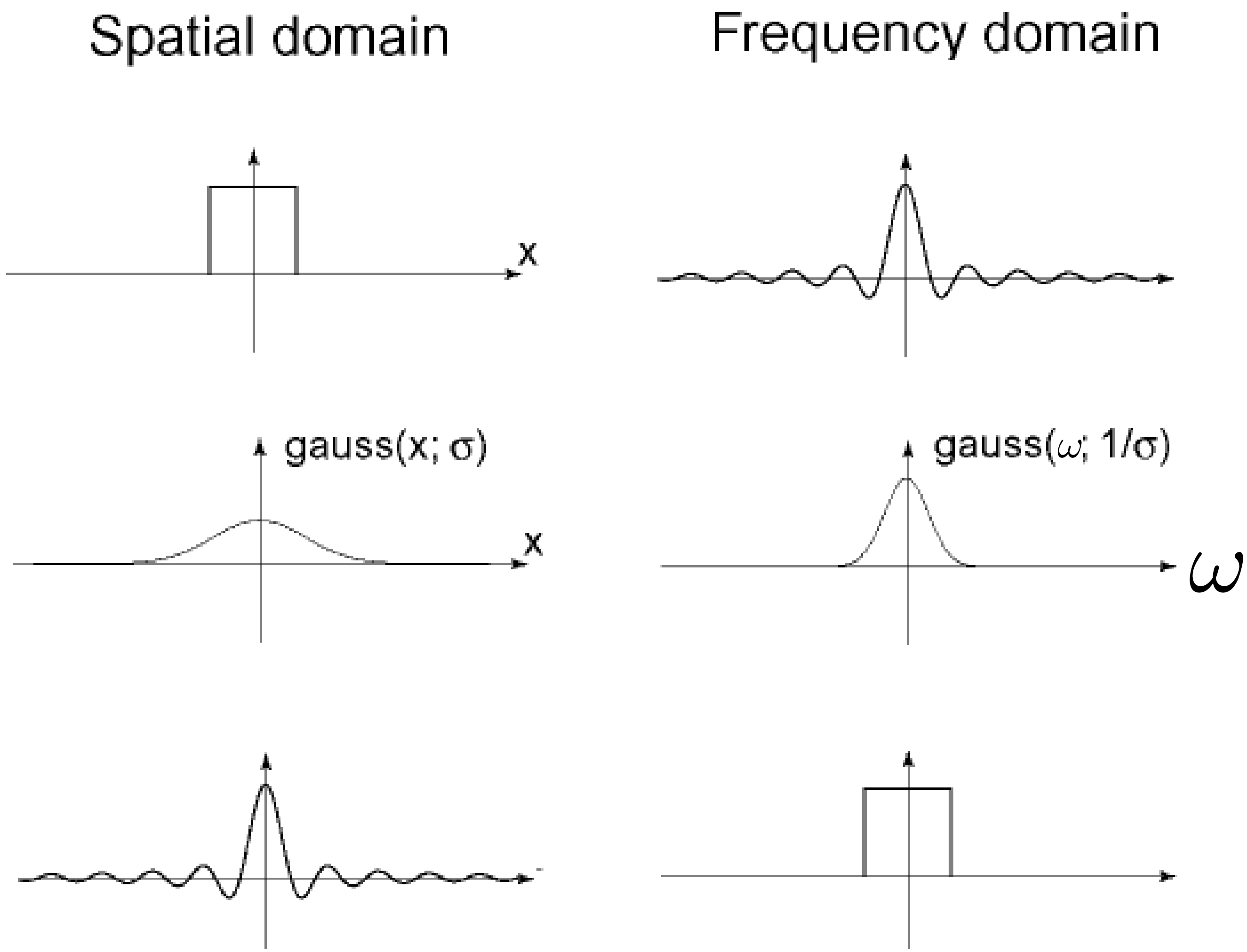
$\frac{d^2 h}{dx^2} * f$



Credit: Efros, Computational Photography

Gaussian Filter

Removes noise while minimizing spatial smoothing

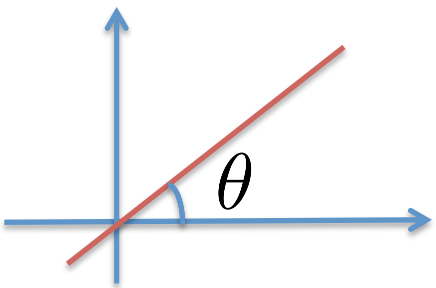


Credit: Efros, Computational Photography

2D Gaussian Filter

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Directional derivatives along $[\cos\theta \;\; \sin\theta]$:



First derivative:

$$D_\theta f(x,y) = \nabla f \cdot [\cos\theta \;\; \sin\theta]$$
$$= \frac{\partial f(x,y)}{\partial x}\cos\theta + \frac{\partial f(x,y)}{\partial y}\sin\theta$$

Second derivative:

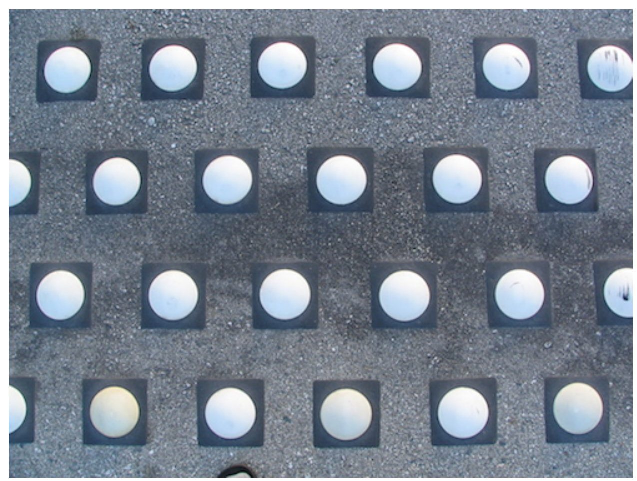
$$D_\theta^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2}\cos^2\theta + \frac{\partial^2 f(x,y)}{\partial x \partial y}\cos\theta\sin\theta + \frac{\partial^2 f(x,y)}{\partial y^2}\sin^2\theta$$

Therefore, we can detect edges in the θ direction from the zero crossings of:

$$D_\theta^2\{h(x,y) * f(x,y)\} = D_\theta^2 h(x,y) * f(x,y)$$

Directional derivatives are suitable to detect lanes or stop markings:

Choose direction θ to be orthogonal to them.



What about isotropic edges?

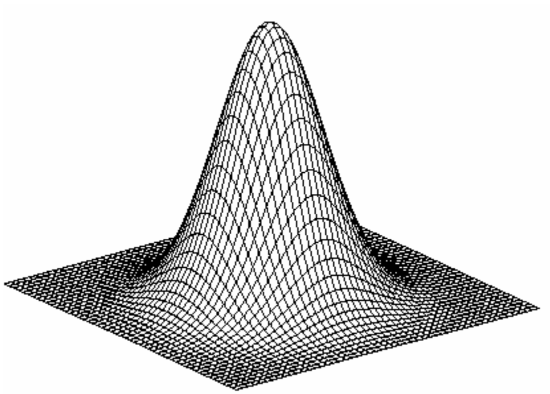
When there is no directionality (e.g., circular objects, such as Botts dots), use the Laplacian:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

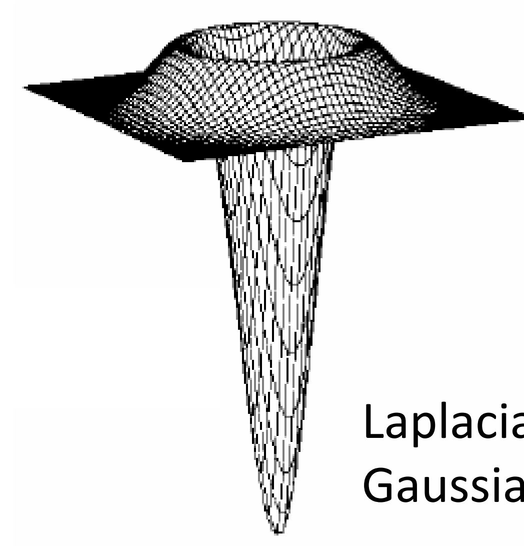
which is invariant under rotations of the image.

Look for zero crossings of:

$$\nabla^2 \{h(x,y) * f(x,y)\} = \nabla^2 h(x,y) * f(x,y)$$



Gaussian



Laplacian of Gaussian

Finite Difference Approximations of Derivatives

Estimate derivatives from samples of $f(x)$:

$$\left.\frac{df(x)}{dx}\right|_{x=n\Delta} \approx \frac{1}{\Delta}\{f(n\Delta+\Delta) - f(n\Delta)\}$$

(forward difference)

$$\frac{1}{\Delta}\{f(n\Delta) - f(n\Delta-\Delta)\}$$

(backward difference)

$$\frac{1}{2\Delta}\{f(n\Delta+\Delta) - f(n\Delta-\Delta)\}$$

(central difference)

Taylor series to assess the approximation accuracy:

$$f(n\Delta + \Delta) = f(n\Delta) + \Delta f'(n\Delta) + \frac{\Delta^2}{2} f''(n\Delta) + \mathcal{O}(\Delta^3)$$

$$f(n\Delta - \Delta) = f(n\Delta) - \Delta f'(n\Delta) + \frac{\Delta^2}{2} f''(n\Delta) + \mathcal{O}(\Delta^3)$$

Forward difference:

$$\frac{1}{\Delta}\{f(n\Delta + \Delta) - f(n\Delta)\} = f'(n\Delta) + \mathcal{O}(\Delta)$$

Backward difference:

$$\frac{1}{\Delta}\{f(n\Delta) - f(n\Delta - \Delta)\} = f'(n\Delta) + \mathcal{O}(\Delta)$$

Central difference:

smaller error

$$\frac{1}{2\Delta}\{f(n\Delta + \Delta) - f(n\Delta - \Delta)\} = f'(n\Delta) + \mathcal{O}(\Delta^2)$$

Second Derivative Approximation

$$\frac{1}{\Delta^2}\{f(n\Delta+\Delta)-2f(n\Delta)+f(n\Delta-\Delta)\}=f''(n\Delta)+\mathcal{O}(\Delta^2)$$

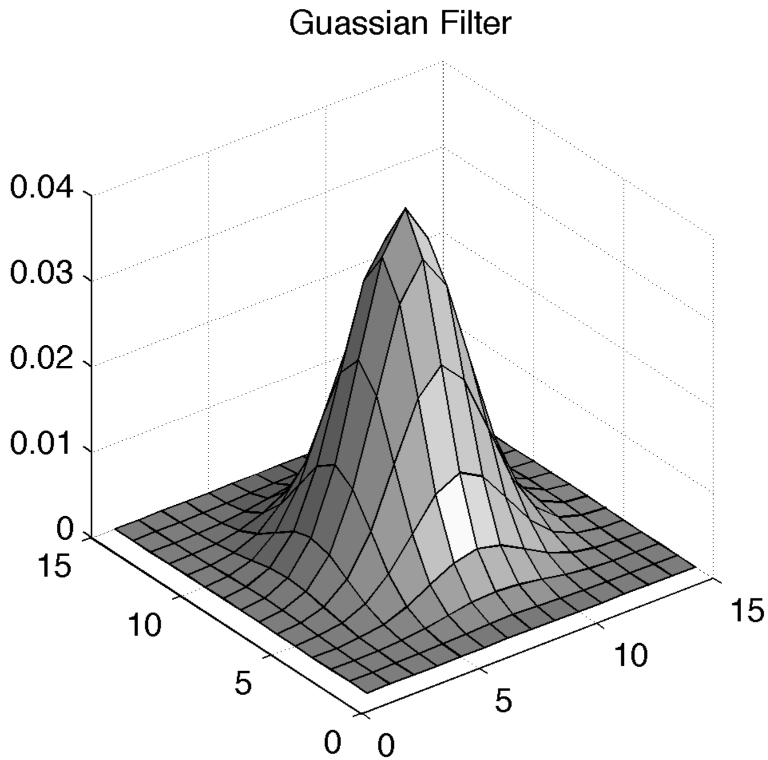
Finite Difference Approximations in 2D

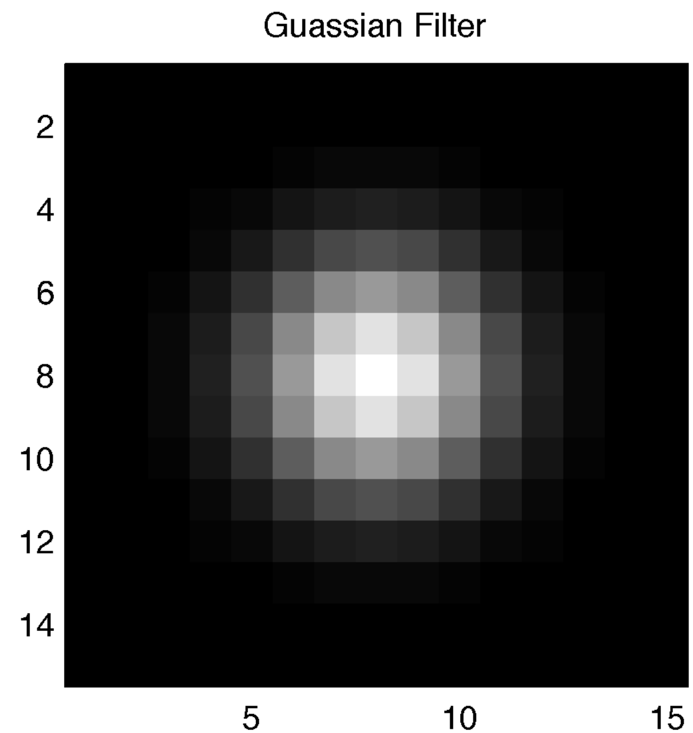
$$\left.\frac{\partial}{\partial x}f(x,y)\right|_{(n_1\Delta,n_2\Delta)}\approx\frac{1}{\Delta}\{f(n_1\Delta+\Delta,n_2\Delta)-f(n_1\Delta,n_2\Delta)\}$$

$$\left.\frac{\partial}{\partial y}f(x,y)\right|_{(n_1\Delta,n_2\Delta)}\approx\frac{1}{\Delta}\{f(n_1\Delta,n_2\Delta+\Delta)-f(n_1\Delta,n_2\Delta)\}$$

$$\begin{aligned}\nabla^2 f(x,y)\approx&\frac{1}{\Delta^2}\{f(n_1\Delta+\Delta,n_2\Delta)+f(n_1\Delta-\Delta,n_2\Delta)\\&f(n_1\Delta,n_2\Delta+\Delta)+f(n_1\Delta,n_2\Delta-\Delta)\\&-4f(n_1\Delta,n_2\Delta)\}\end{aligned}$$

Gaussian Filter and its Derivatives in MATLAB

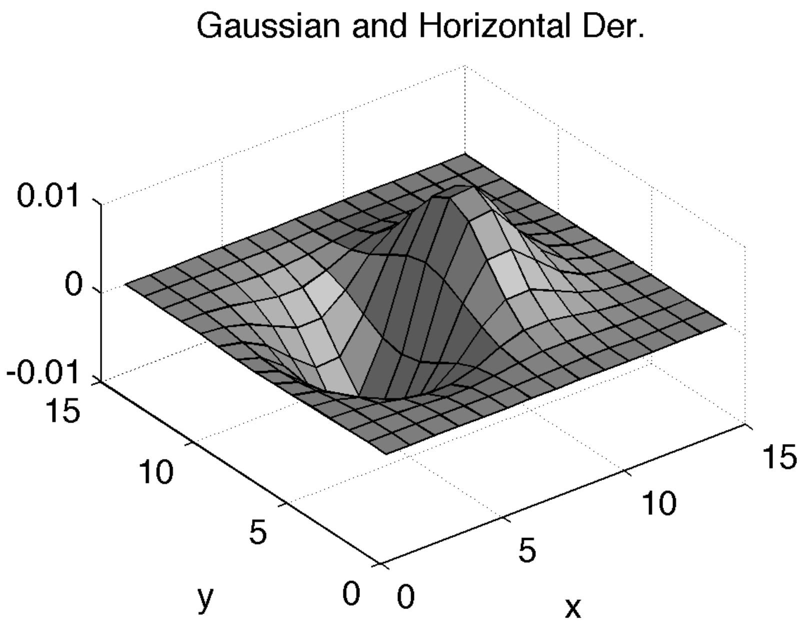


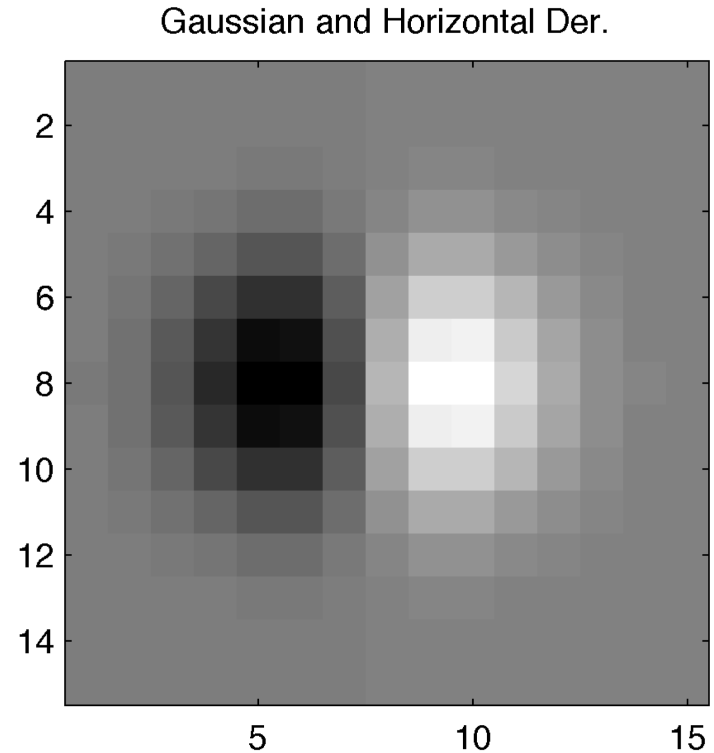


Gaussian Filter, 15x15 pixels, σ=2

```
>> g=fspecial('gaussian',15,2);
```

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

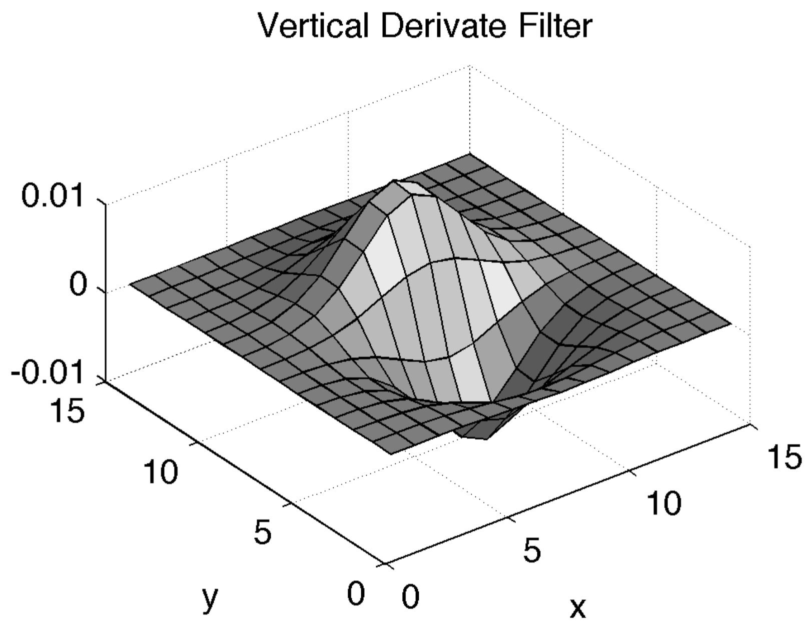


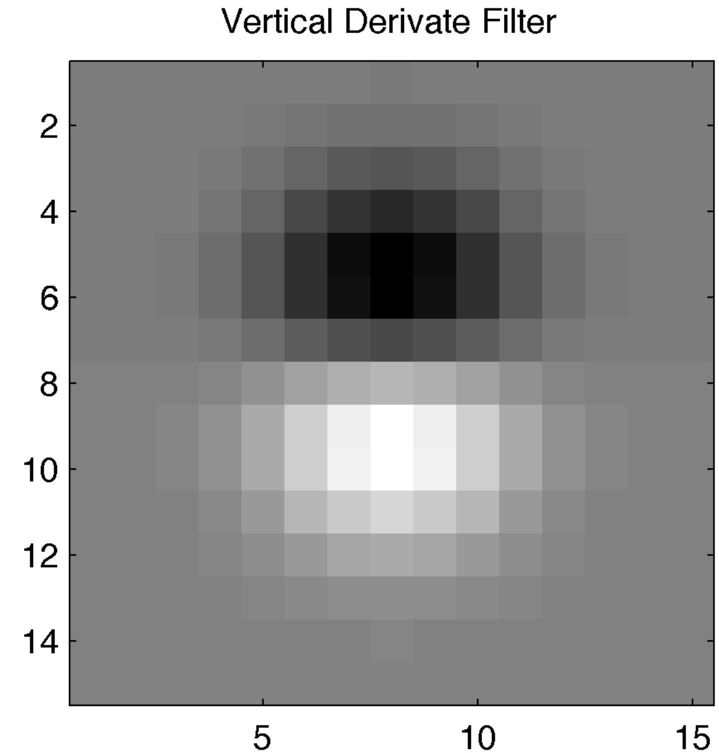


Combine Gaussian Filter with Horizontal Derivative

```
>> dx=conv2(g,[-1,1],'same');
```

$$\frac{\partial}{\partial x} h(x,y)$$

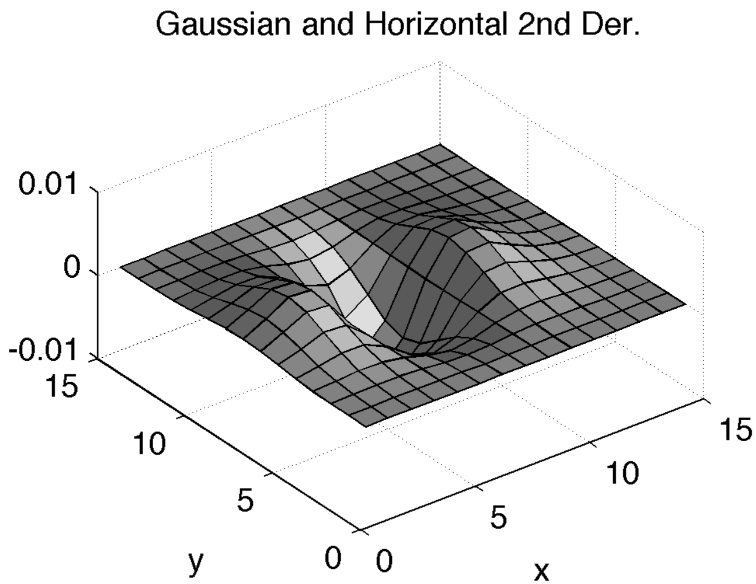


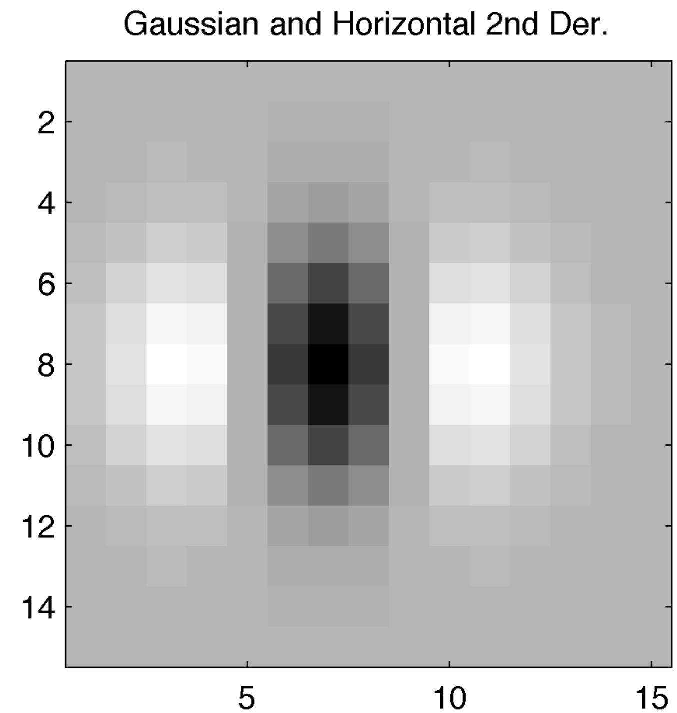


Vertical Derivative Filter (with Gaussian)

```
>> dy=conv2(g,[-1;1],'same');
```

$$\frac{\partial}{\partial y} h(x,y)$$

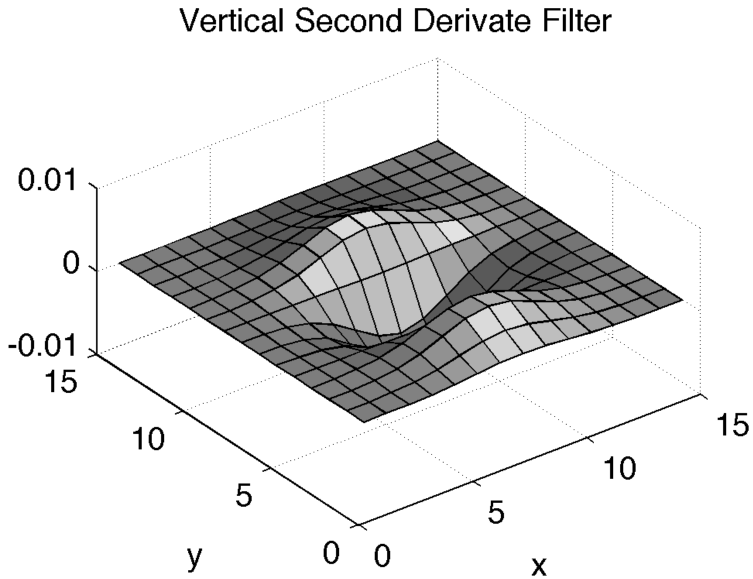


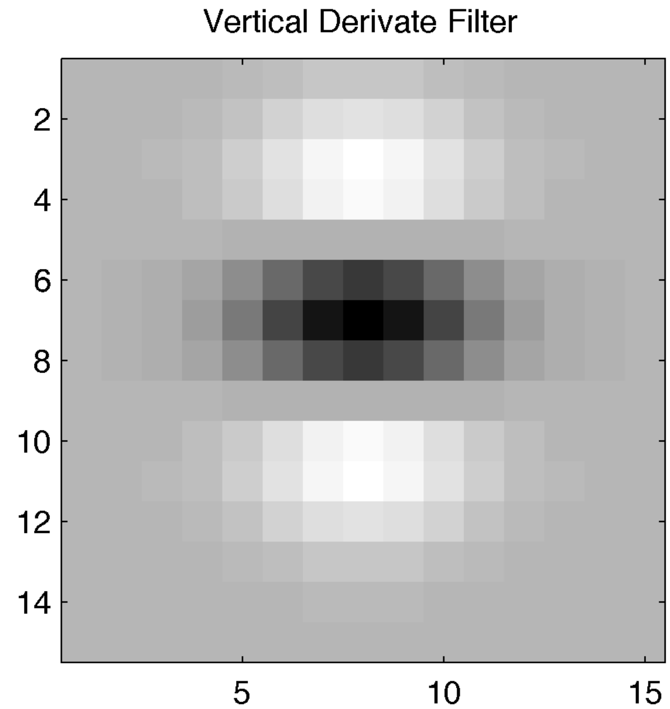


Second Derivative of Gaussian in Horizontal Dir.

```
>> ddx=conv2(dx,[-1,1],'same')
```

$$\frac{\partial^2}{\partial x^2} h(x, y)$$

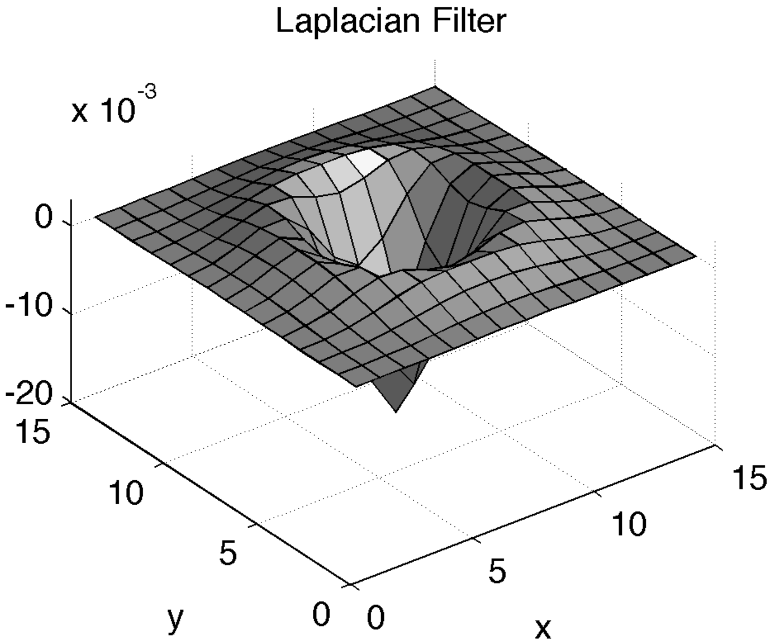


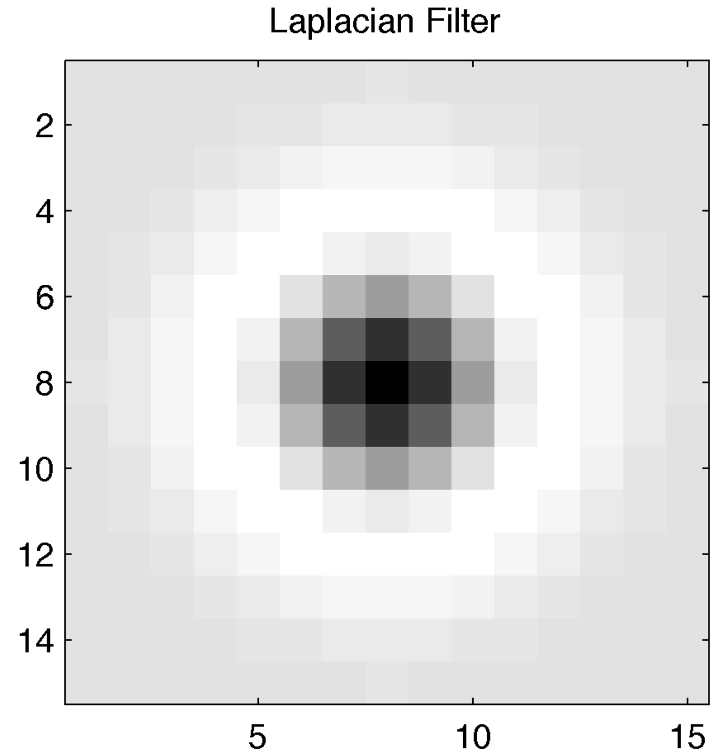


Vertical Second Derivative Filter (with Gaussian)

```
>> dy=conv2(g,[-1;1],'same');
```

$$\frac{\partial^2}{\partial y^2} h(x,y)$$



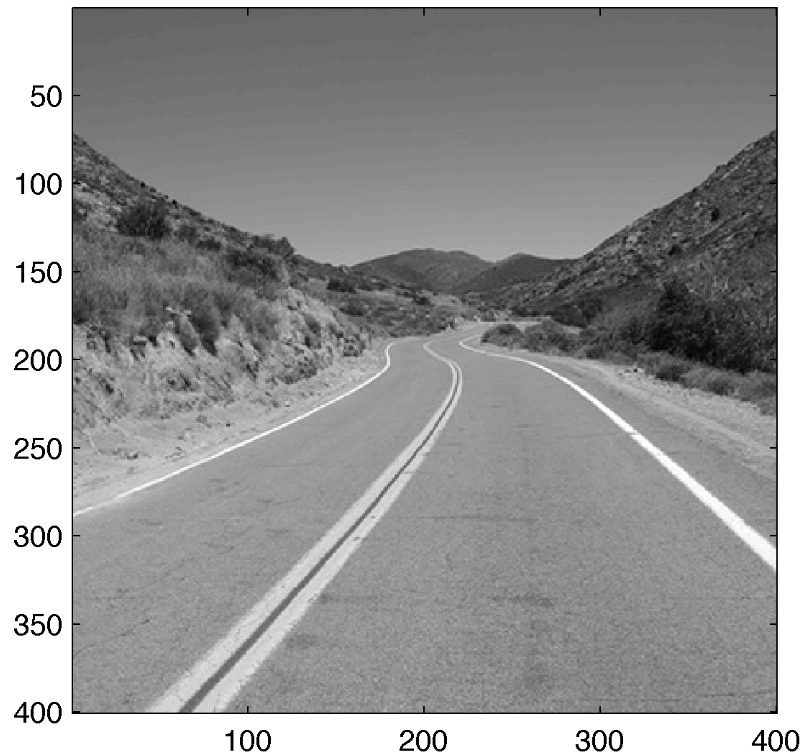


Laplacian Filtering

```
>> lg=fspecial('log',15,2);
```

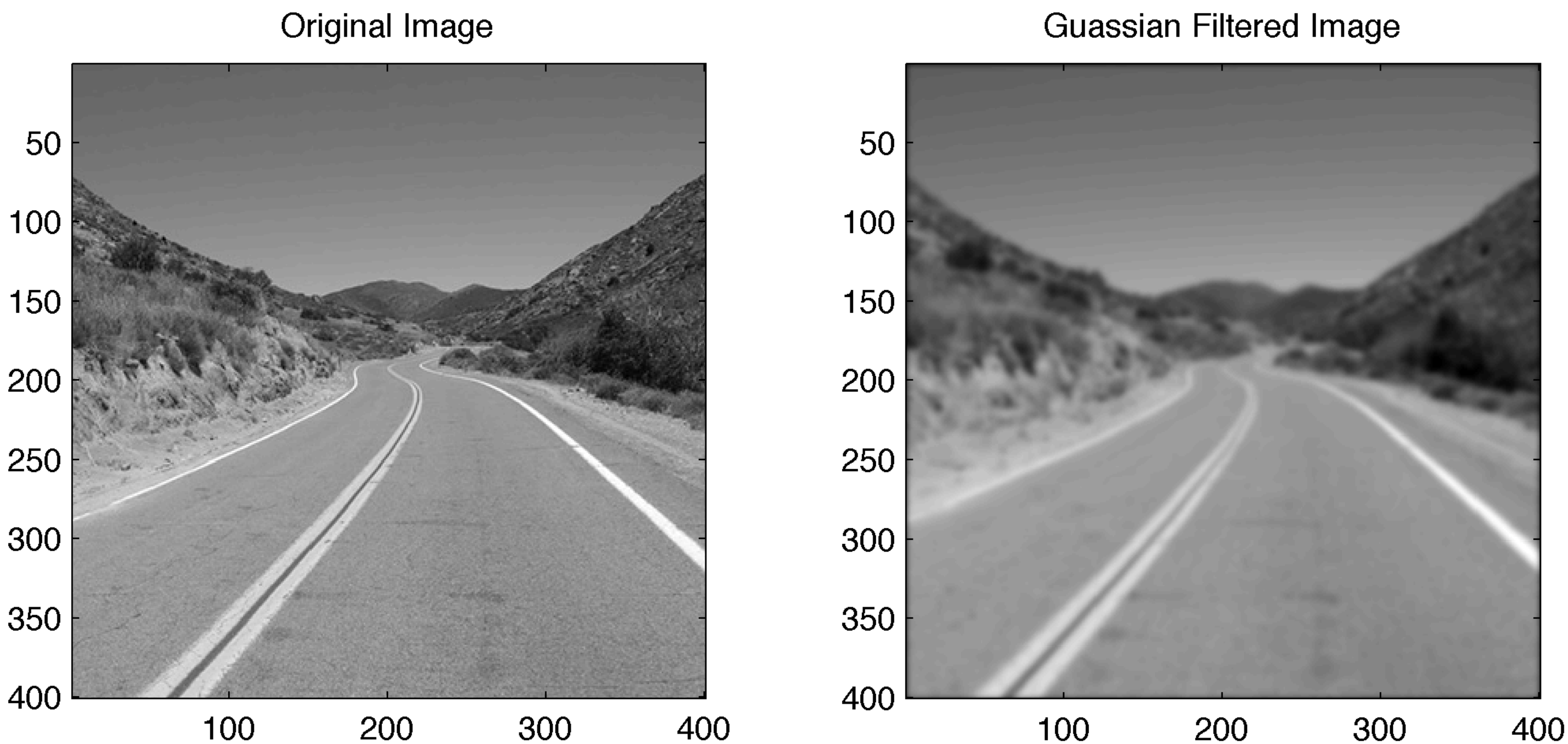
$$\nabla^2 h(x,y)$$

Edge Detection Examples with MATLAB



Load An Image

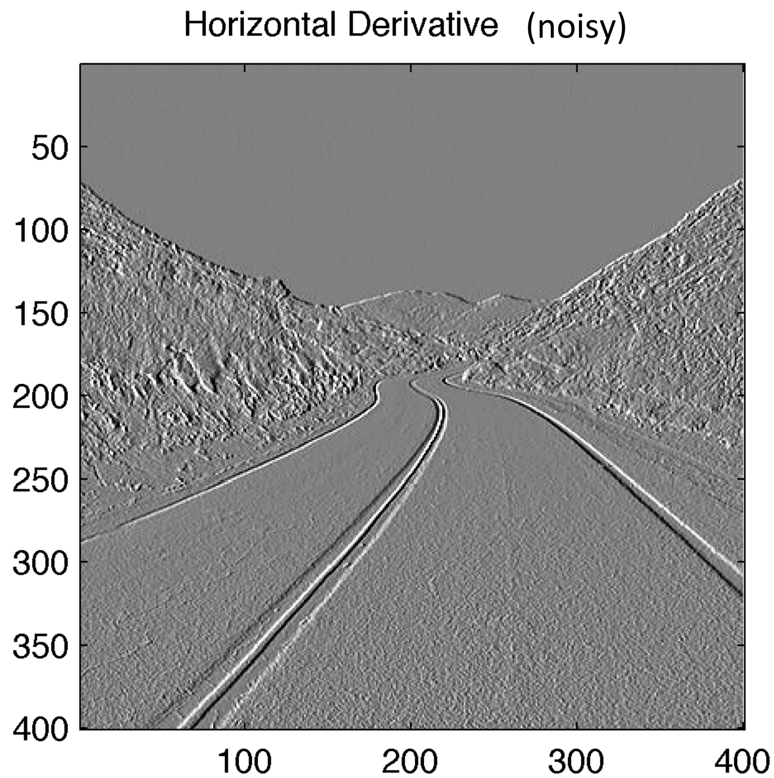
```
>> road=imread('road.jpg');
>> imagesc(road); %Scale values and display image
```

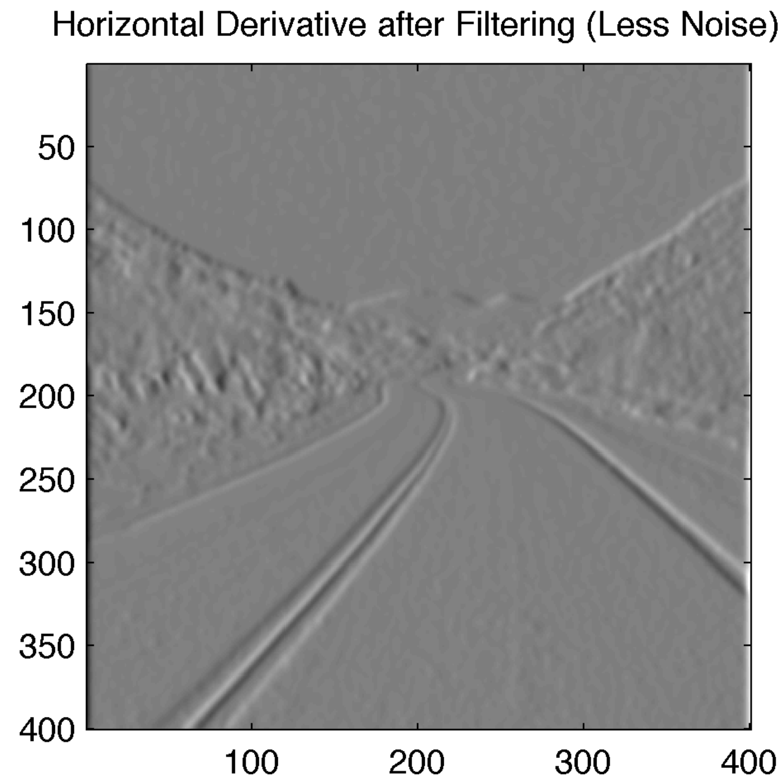


Gaussian Filtering

```
>> groad=conv2(road,g,'same');
```

$$f(x,y) \text{ vs. } h(x,y) * f(x,y)$$

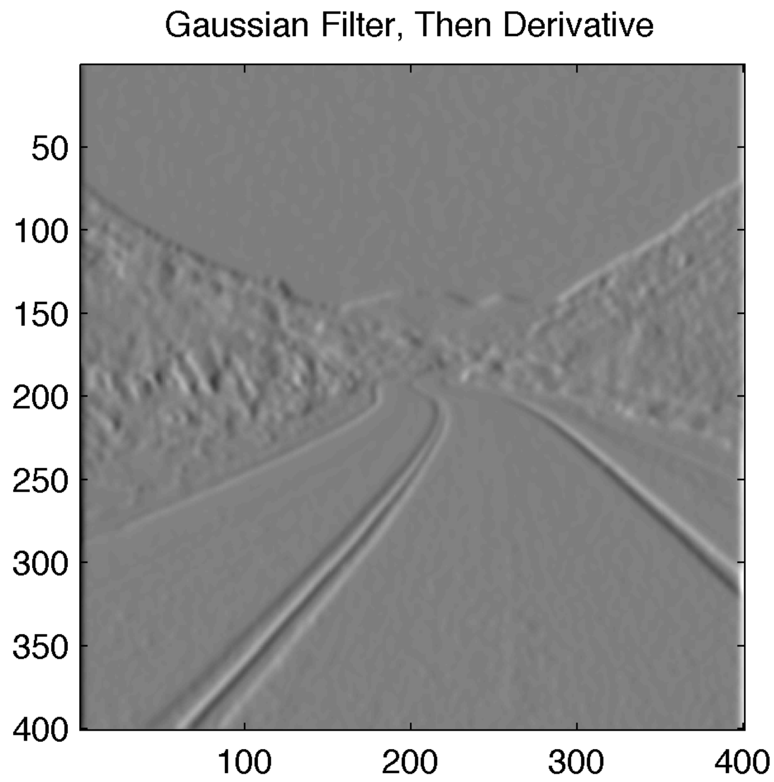


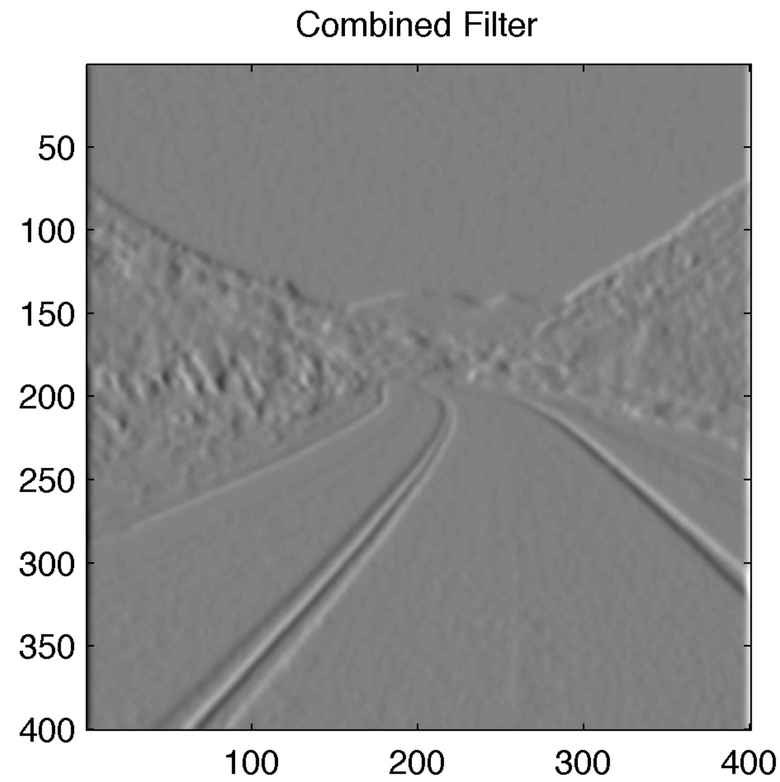


Edge Detection Along Horizontal Direction

```
>> conv2(road,[-1,1],'same')
>> conv2(groad,[-1,1],'same')
```

$$\frac{\partial}{\partial x} f(x,y) \text{ vs. } \frac{\partial}{\partial x}\{h(x,y) * f(x,y)\}$$

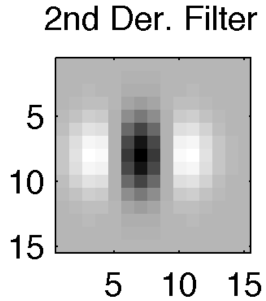


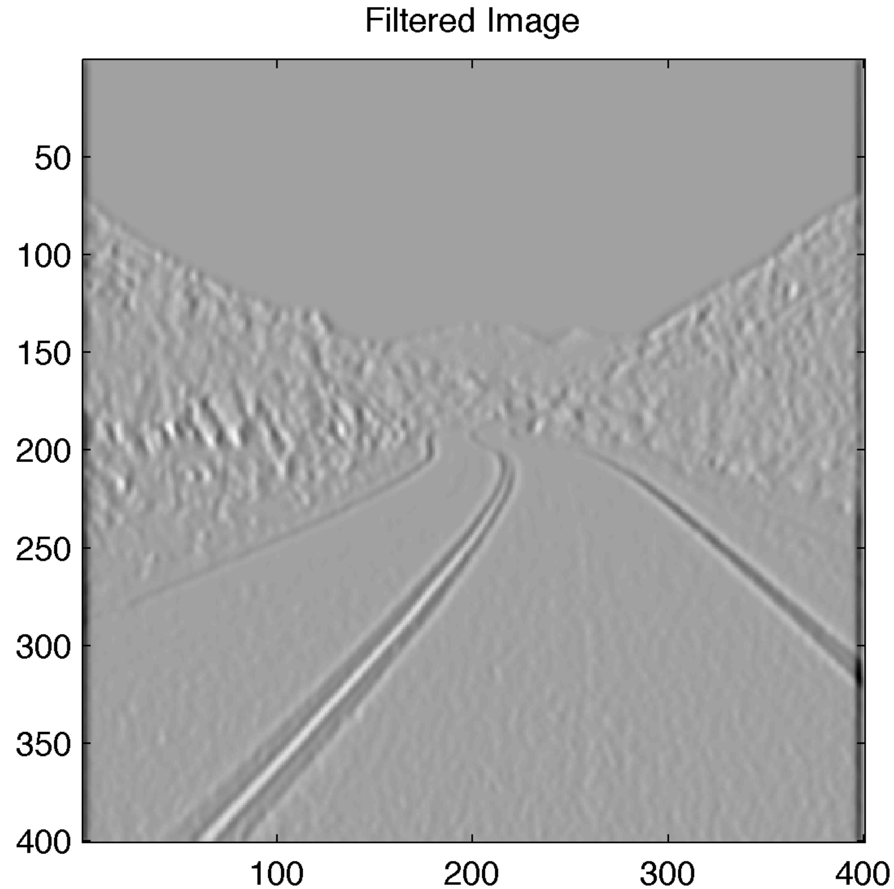


Same Result

```
>> conv2(groad,[-1,1],'same')
>> conv2(road,dx,'same')
```

$$\frac{\partial}{\partial x}\{h * f\} = \frac{\partial h}{\partial x} * f$$





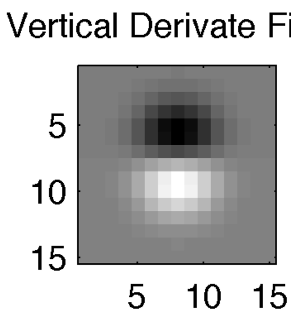
Second Derivative in Horizontal Direction

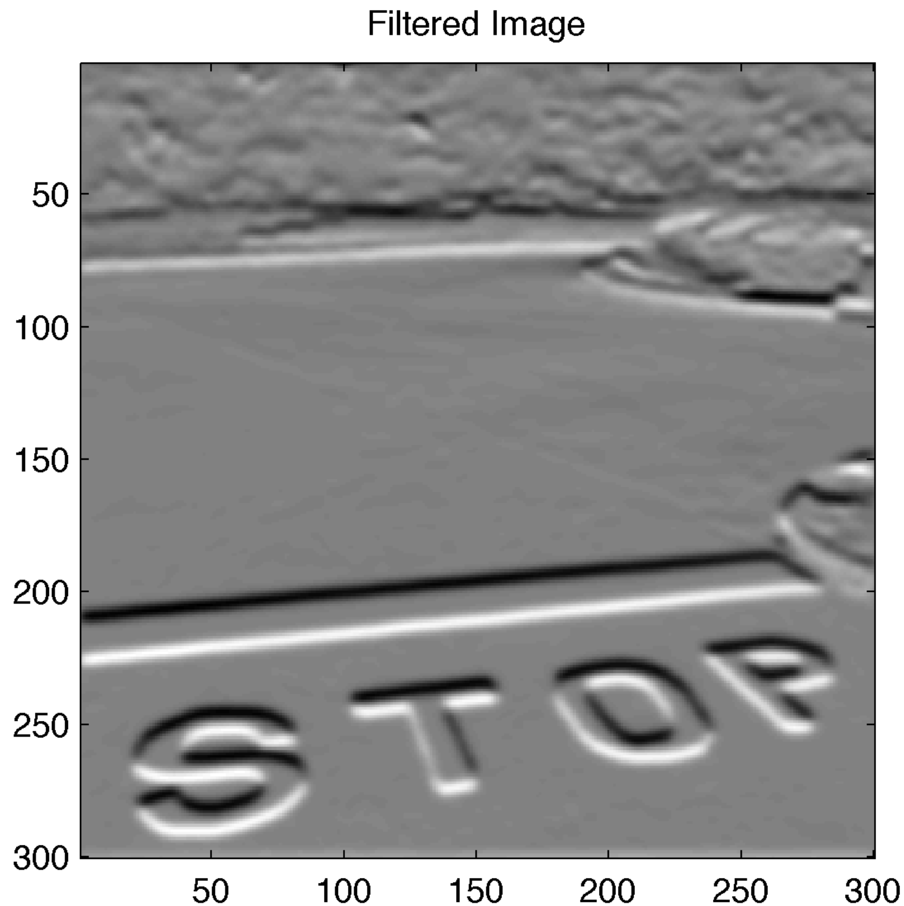
```
>> conv2(road,ddx,'same')
```

$$\frac{\partial^2}{\partial x^2} h(x,y) * f(x,y)$$

Stop Line Detection

>> stop_im = imread('stop.jpg')

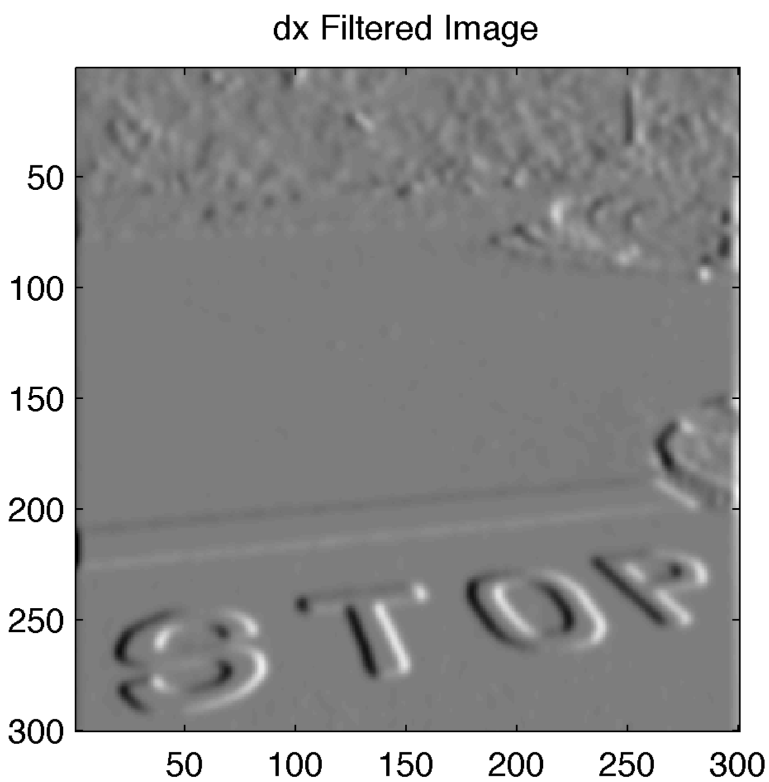


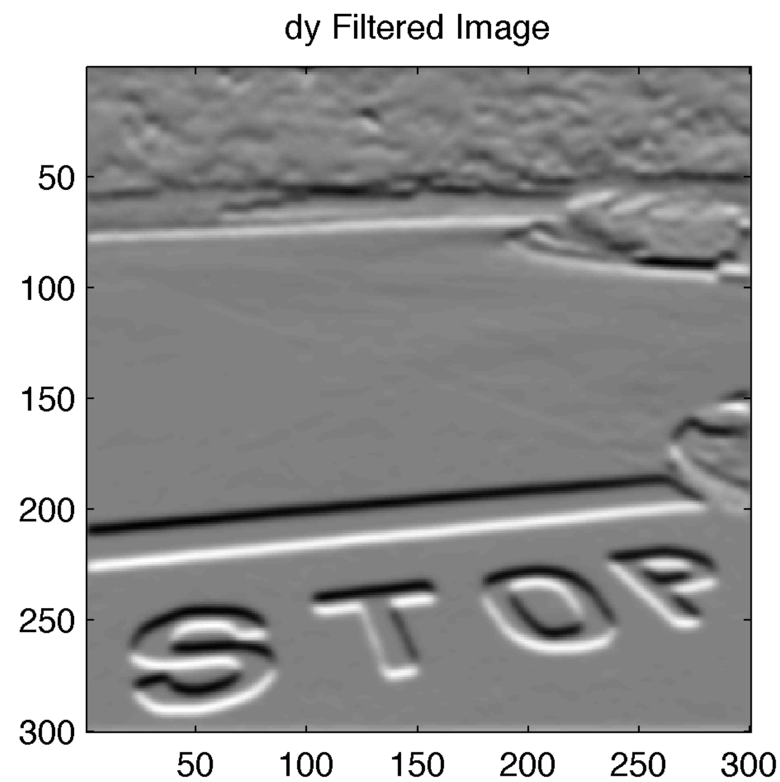


Detecting Stop Bar using Vertical Derivative

```
>> conv2(stop_im,dy,'same')
```

$$\frac{\partial h}{\partial y} * f$$

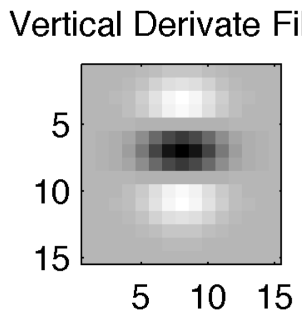


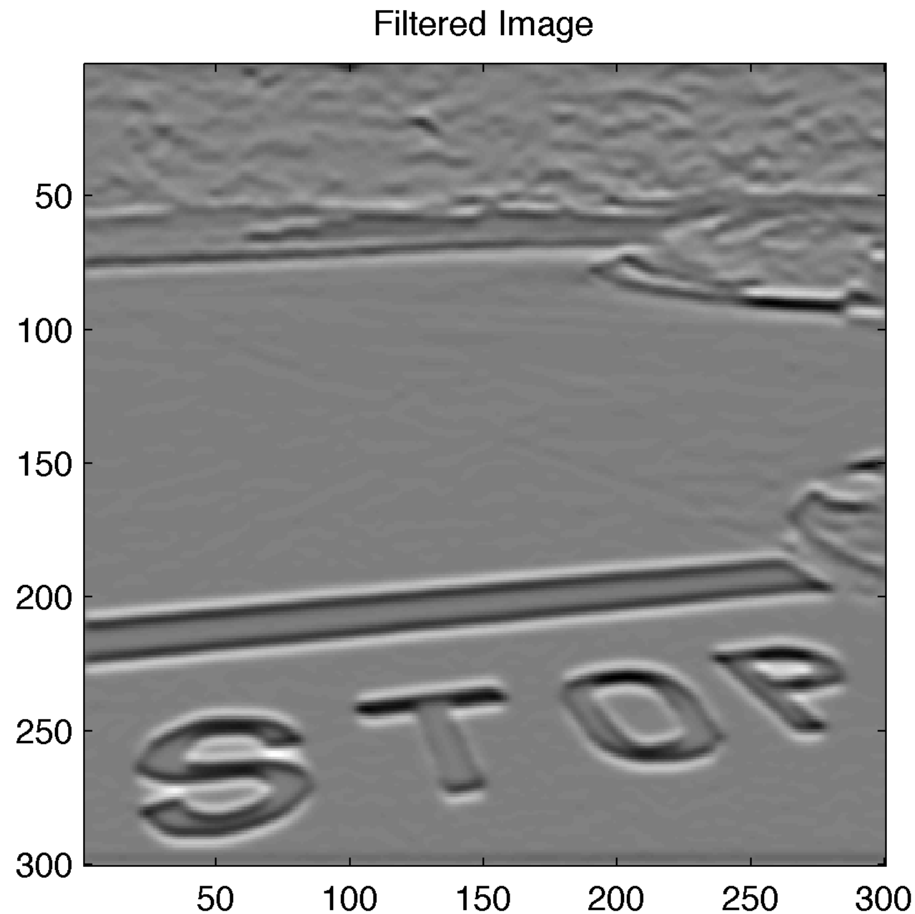


Comparing Horizontal and Vertical Derivative

```
>> conv2(stop_im,dx,'same')
>> conv2(stop_im,dy,'same')
```

$$\frac{\partial h}{\partial x} * f \text{ vs. } \frac{\partial h}{\partial y} * f$$

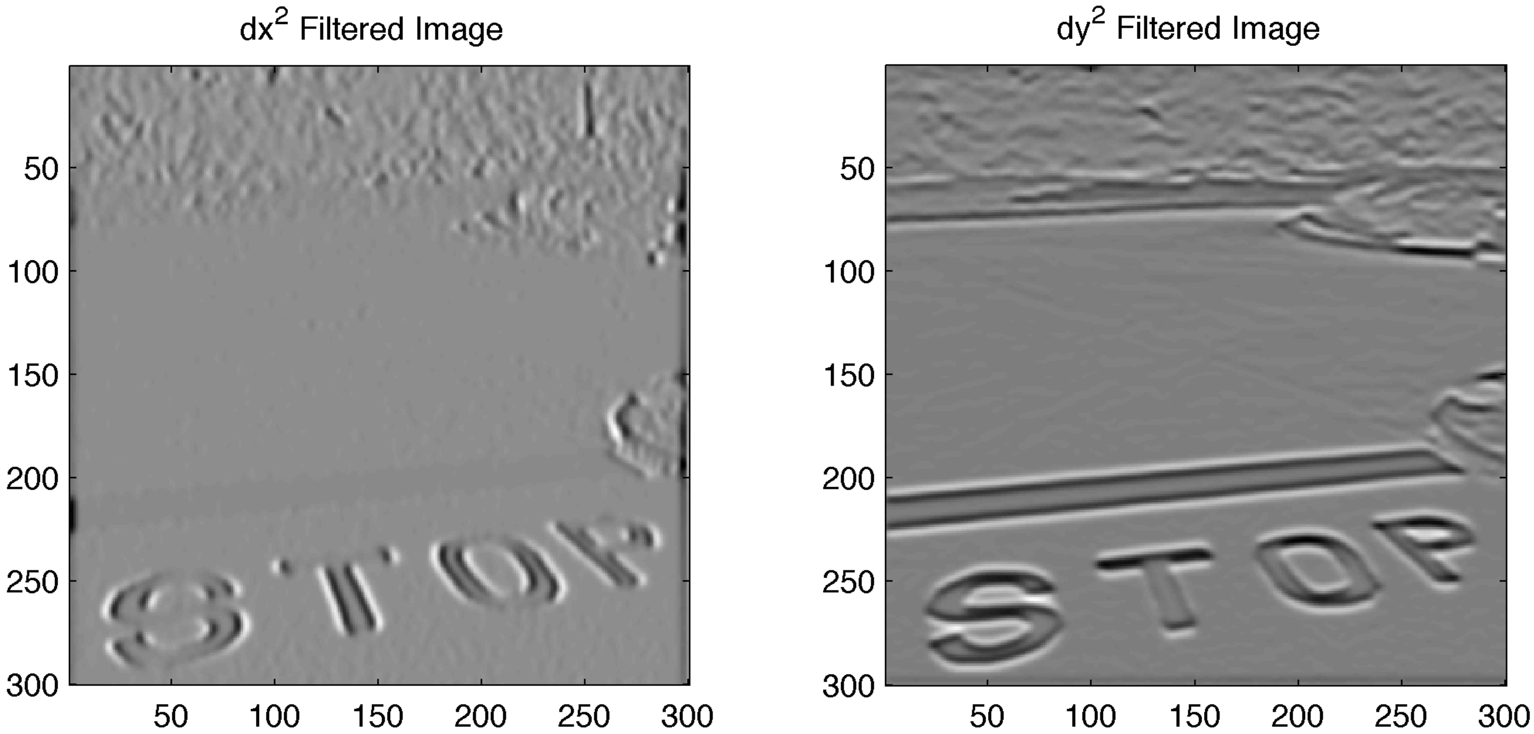




Detecting Stop Bar using Vertical Second Derivative

```
>> conv2(stop_im,dy,'same')
```

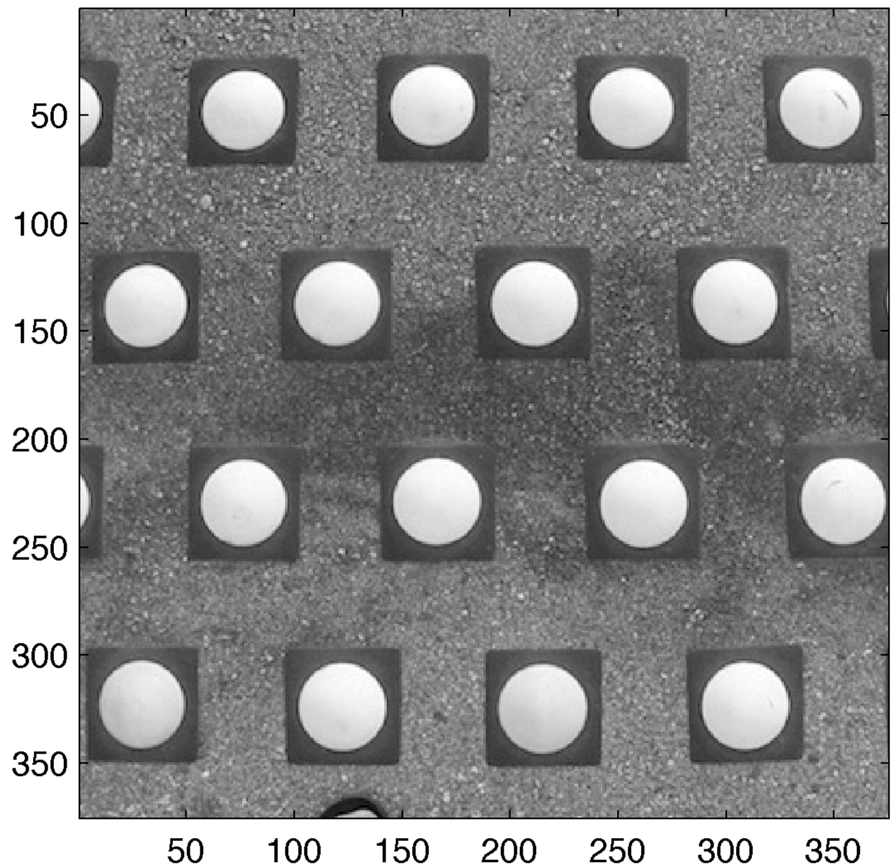
$$\frac{\partial^2}{\partial y^2} h(x,y) * f(x,y)$$



Comparing Horizontal and Vertical Second Derivatives

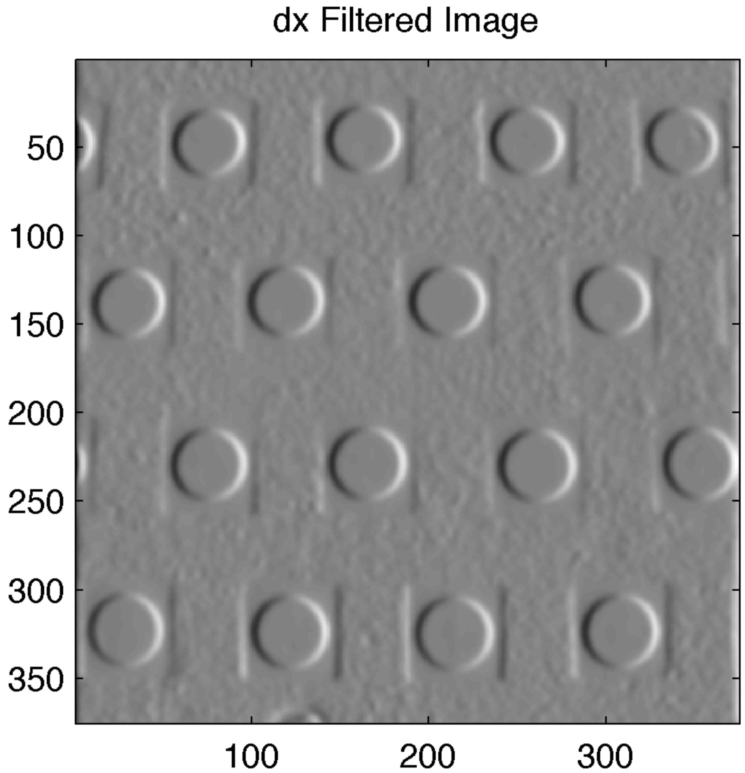
```
>> conv2(stop_im,ddx,'same')
>> conv2(stop_im,ddy,'same')
```

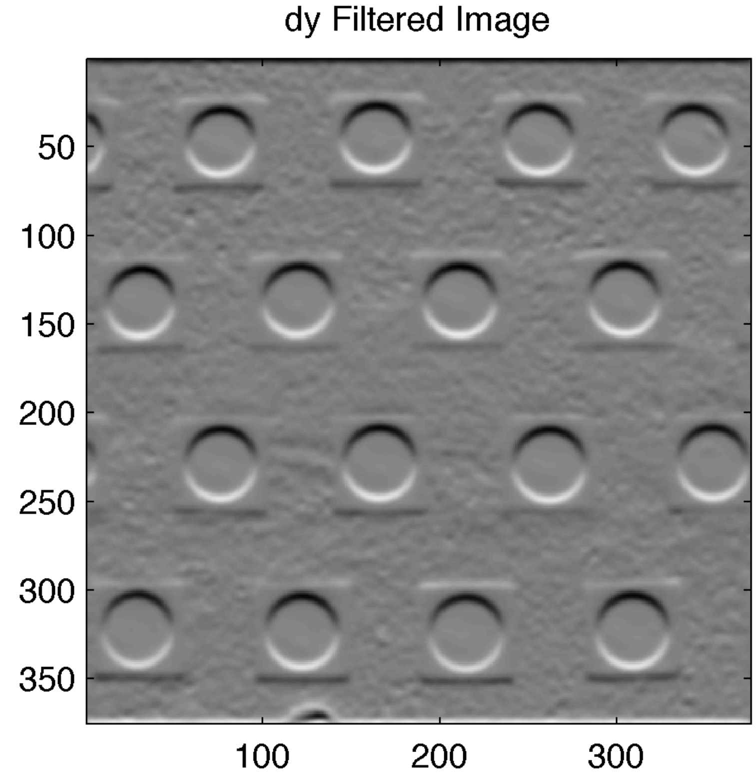
$$\frac{\partial^2}{\partial x^2} h(x,y) * f(x,y) \text{ vs. } \frac{\partial^2}{\partial y^2} h(x,y) * f(x,y)$$



Detecting Botts Dots

```
>> botts=imread('botts.jpg');
```

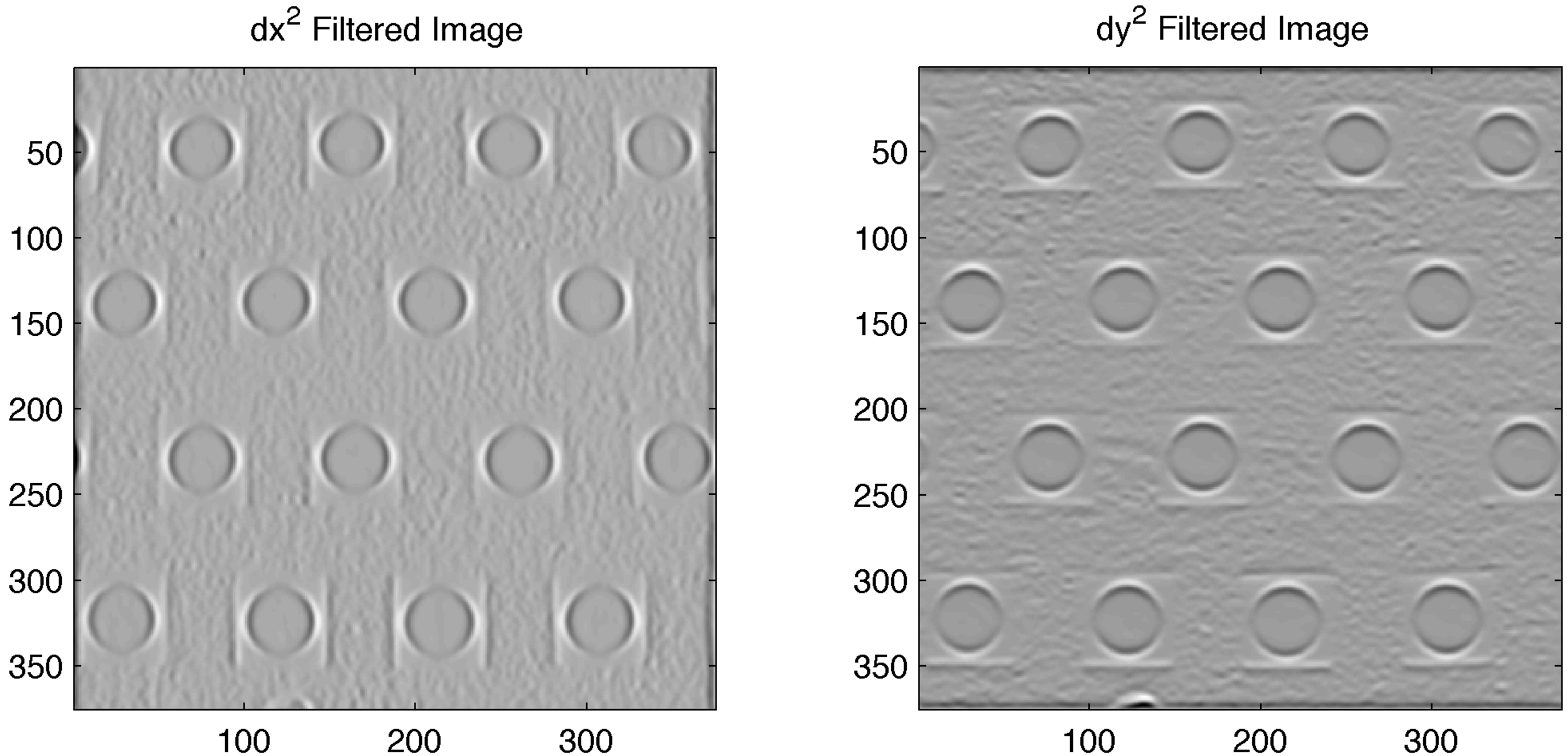




Horizontal and Vertical Derivative Filtering

```
>> conv2(botts,dx,'same')
>> conv2(botts,dy,'same')
```

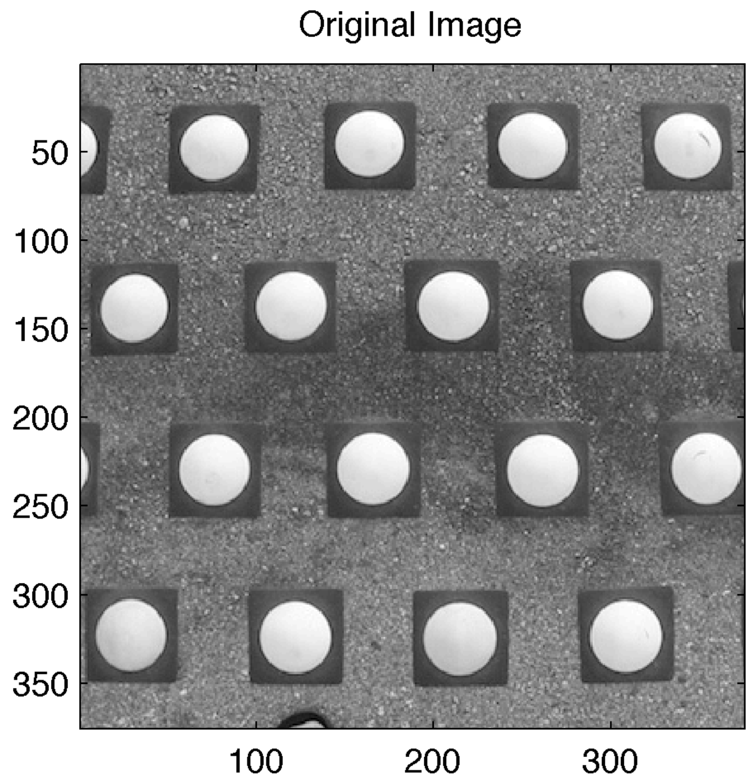
$$\frac{\partial}{\partial x} h(x,y) * f(x,y) \text{ vs. } \frac{\partial}{\partial y} h(x,y) * f(x,y)$$

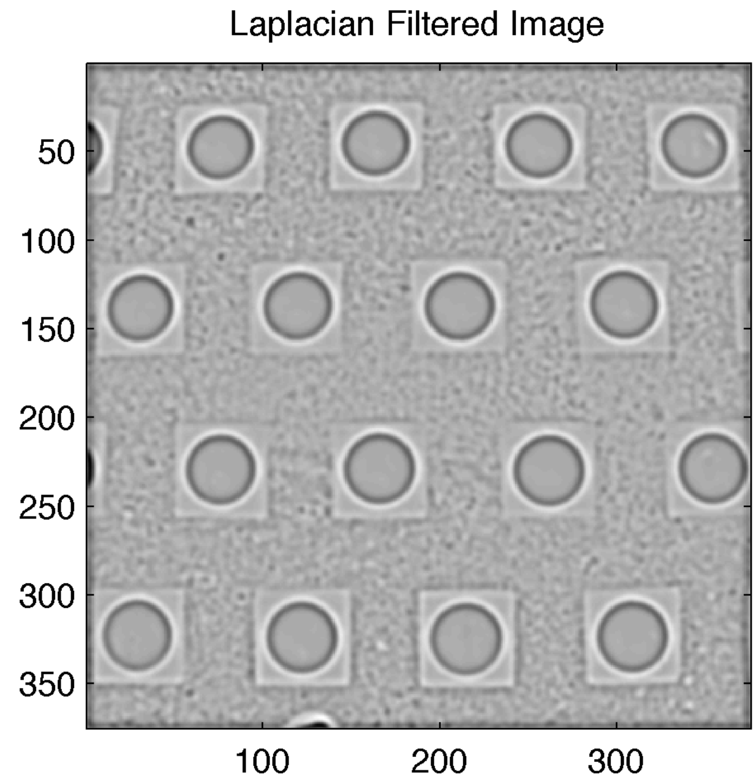


Horizontal and Vertical Second Derivative Filtering

```
>> conv2(botts,ddx,'same')
>> conv2(botts,ddy,'same')
```

$$\frac{\partial^2}{\partial x^2} h(x,y) * f(x,y) \text{ vs. } \frac{\partial^2}{\partial y^2} h(x,y) * f(x,y)$$





Original versus Laplacian

```
>> conv2(botts,lg,'same')
```

$$\nabla^2 h * f$$