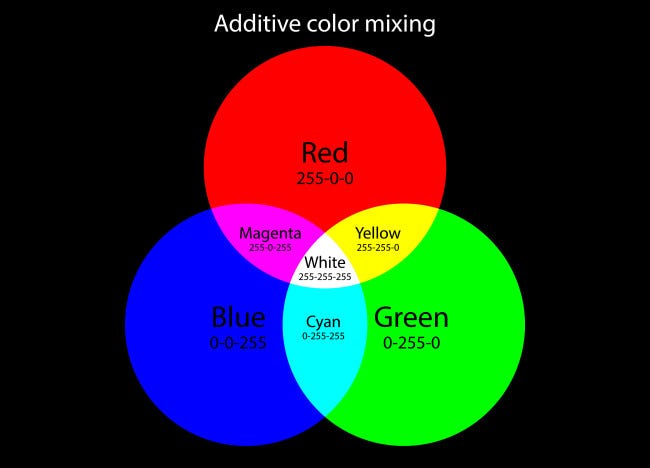
# Objektdetektering

## HSV vs RGB

Et billede, der indeholder tekst, lys

Automatisk genereret beskrivelse<https://www.quora.com/Why-use-an-HSV-image-for-color-detection-rather-than-an-RGB-image>

Fordelen ved HSV er at to røde hue værdier ligger tæt op ad hinanden, hvor de ved RGB kan have vidt forskellige farvekode kombinationer. Ændringer i lys påvirker mindre deviationen hos HSV end hos RGB, hvilket gør det nemmere at isolere den farve, som man ønsker.



# Neuralt netværk

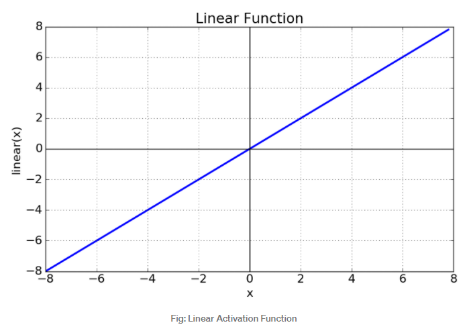
## Transfer learning

## Aktiveringsfunktion

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6?gi=56bd74d85bcf>

Ideen med en aktiveringsfunktion er, at ud fra den værdi, som et neuron giver, så beslutter aktiveringsfunktionen, om den værdi skal med videre i netværket. Der findes flere typer af aktiveringsfunktioner, heraf både lineære og u lineære. Fremover skrives aktiveringsfunktion som AF.

**Lineær aktiveringsfunktion**

Her ses en lineær AF.

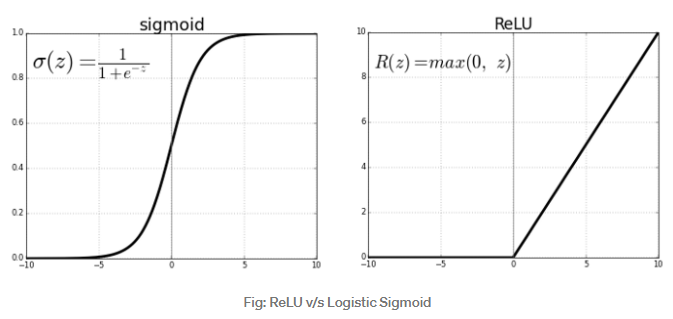
Formel:

Denne AF hjælper ikke på at mindske kompleksiteten

**Nonlineær aktiveringsfunktion**

Den mest benyttet AF. Man ser på om funktionen er differentiabel (finder en hældning) og monoton.

Mindsker kompleksiteten. Ulempen ved en nonlineær AF er, at den smider data væk, hvis dataen ligger udenfor dens rækkevidde.



## Gradient descent

<https://www.youtube.com/watch?v=sDv4f4s2SB8>

Det er en proces, der sker i backpropagation-delen.

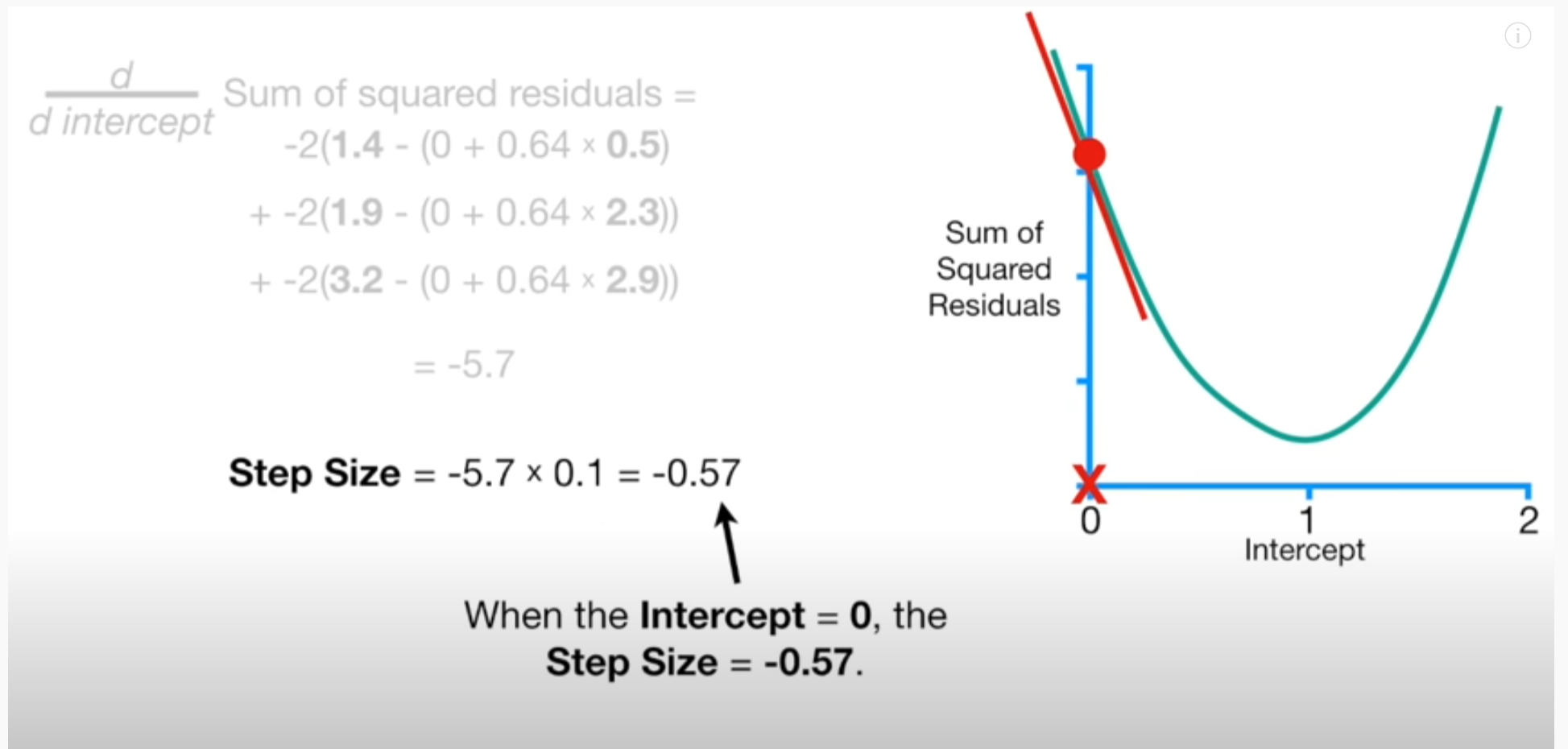
Exploding and vanishing gradients – Den eksploderer hvis normalen af vægtmatricen er over 1 og forsvinder, hvis den er under 1.

Når den optimale linje skal findes for sine punkter, benyttes gradient descent. Gradient descent sørger for at, hvis der er stor afstand med en loss funktion, så tager den store skridt, og størrelsen reduceres i takt med at den kommer tættere på den optimale løsning.

Opsummeret, så sørger gradient for at finde den bedste tilpassende linje / flade til alle datapunkterne på baggrund af loss funktionens hældning og skæring (altså hvor den møder et toppunkt på kurven).

## Læringskurve / learning rate

Gradient descent bestemmer step size sammen med læringskurven. Når gradient descent differentieres, så finder man den hældning, som kurven har i det pågældende punkt. Den hældning ganges med learning rate, og derved bestemmes step size på den måde for at undgå at man tager et for stort skridt, der resulterer i at man overskrider den laveste afvigelse af den fittede linje. Gradient descent stopper, når step size er mindre end learning rate værdi.



## Batch Normalisering

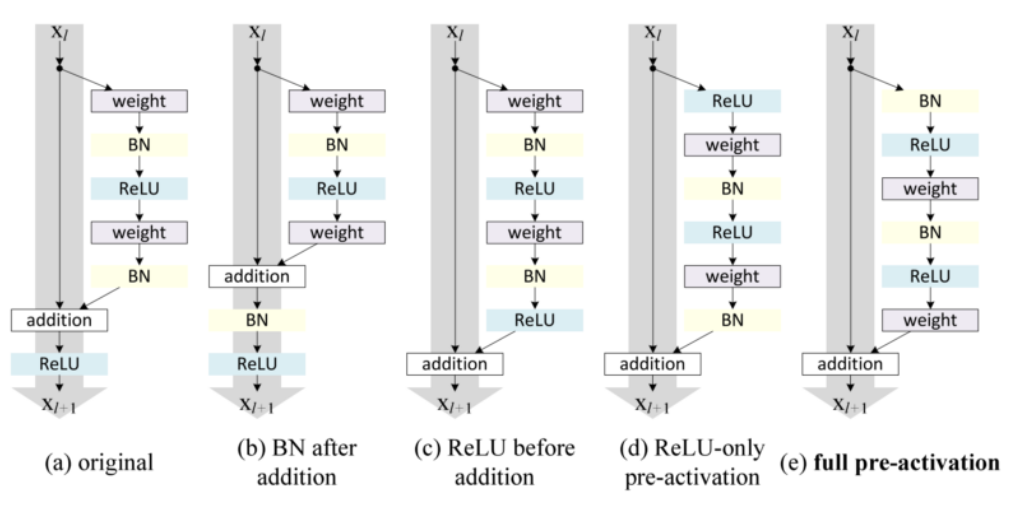
Batch normalization kan tilføjes til konvolutional og fully connected lag, og kan hjælpe til at forbedre gradiant flow, som også kan beskrives som det generelle forløb af netværkets optimering i træningen. Det kan derfor betyde at der kan bruges højere learning rate, da det tillader en mere aggressiv optimering af netværket. Batch normalization virker ved at normalisere outputtet af hvert activation map hvis det bliver brugt på et convolutional lag, eller generelt på hele aktiveringen for et fully connected lag. I normaliseringen bliver alle værdier tvunget til at være normalfordelt, og bliver derfor rettet ind. Det sker ved at hver værdi trækkes fra middelværdien af alle værdier og derefter divideres med kvadratroden af variansen. Der kan også trænes to værdier, shift og scale, til at blive brugt som faktorer i udregningen til at rette ind på værdierne hvis mean er langt fra 0 eller variansen er langt fra 1.

## Residual Blocks

Residual blocks anvendes, fordi meget dybe netværk er svære at træne pga. forsvindende eller eksploderende gradienter. De tager inputtet og føder det fremad, hvor det adderes efter en standard eller depthwise separable convolution lige før den tilhørende aktiveringsfunktion.

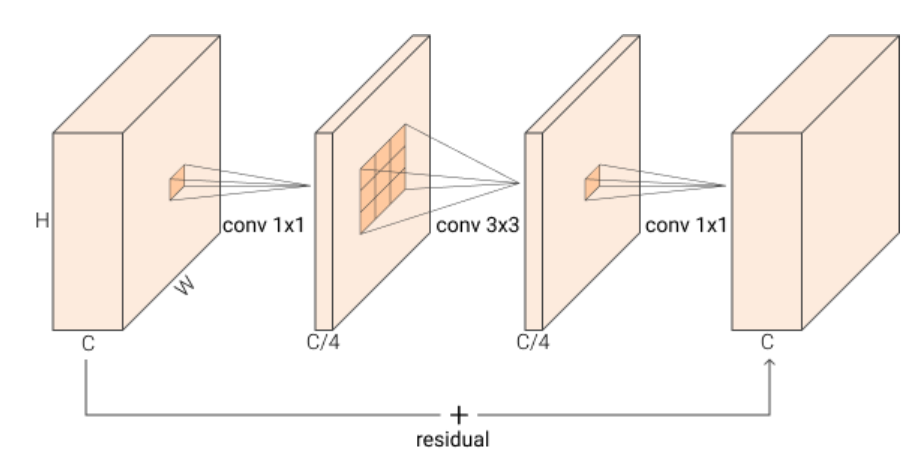
Der findes forskellige typer af residuals. Herunder ses forskellige metoder, hvorpå man kan behandle de data, som trækkes ud og sendes videre i netværket.

Jeg har set billedet omvendt flere gange. En residual er i bund og grund bare en skip connection, hvor det data, som kommer som input bare adderes til outputtet af de blokke imellem.

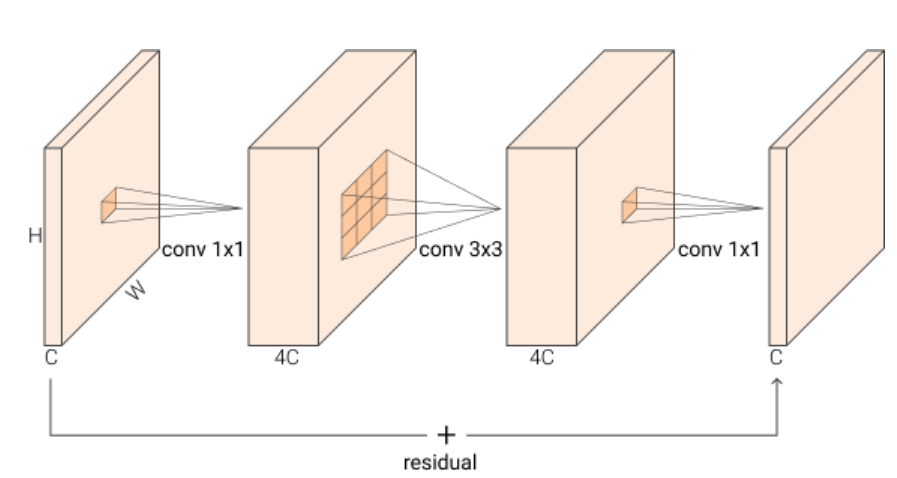


## Inverted Residual blocks

En almindelig residual blok ser sådan ud.



En inverted residual blok ser sådan ud.



Og kaldt en omvendt bottleneck blok.

<https://paperswithcode.com/method/inverted-residual-block>

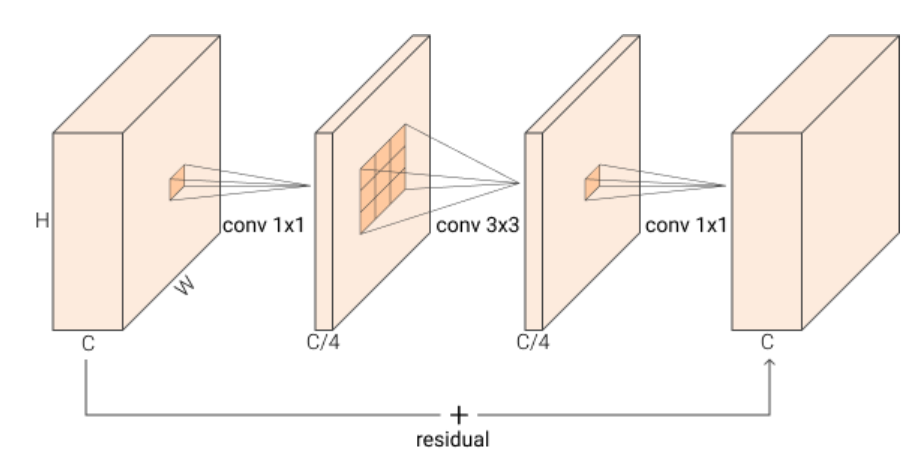
Fordelen er man reducerer antallet af parametre betydeligt.

## Bottlenecks

<https://towardsdatascience.com/residual-bottleneck-inverted-residual-linear-bottleneck-mbconv-explained-89d7b7e7c6bc>

Der er måske en misforståelse af bottlenecks. Beskrivelsen fra rapporten skriver, at det er fordi at aktiveringsfunktionen er lineær. Efter at have læst det igen er jeg kommet frem til at en bottleneck formegentlig er den 1x1xdybde konvolution, der er i netværket. Dvs. når man konverterer dybdelagene til et enkelt lag. Samtidig med når man konveterer tilbage fra 1 lag til samme størrelse dybdelag, som før.

Lineær bottleneck er så en bottleneck block, som ses her, men uden en aktiveringsfunktion.



Beskrivelsen fra opgaven:

*Linear bottleneck betyder at det sidste lag af en residual blok fjernes aktiveringsfunktionen, som giver et lineært output i stedet for ReLU's ulineære output. Det giver et mindre tab af data på bekostning af ydeevne, og derfor kaldes det for bottlenecks. Sammen med inverted residual blokke, så gør det databehandlingen mere hukommelseseffektiv. hvor billede viser MobileNet V1's lagstruktur og viser MobileNet V2's lagstruktur. Forskellen ses på lag, hvor der kun er et enkelt stride, så anvendes inverted residual blocks og lineære bottlenecks, mens hvis der en stride på mere end en, så anvendes kun den lineære bottleneck.*

## Coco matrix score

# Regulering

## State Space

Det er en lineær repræsentation af et dynamisk system i enten kontinuert eller diskret form.

Ud fra state space kan man finde systemets stabilitet, hvilket er egenværdien af matrix A. Mindre end 0 giver et stabilt system.

State space sørger for at man kan konvertere bevægelsesligninger for en mekanisk model om til matricer og vektorer. State beskriver en gruppe variabler, som beskriver historien af et system til at forudsige fremtidigt variabler. Antallet af state variabler skal være lig med antal af storage elementer i systemet.

# Reguleringsteknik

## LQR

LQR er et feedback system, ligesom pole placement. I LQR laver man en cost function i stedet for pole placement til sin gain *K*. Cost funktionen styrer således, hvilken af sine parametre, der har det mest ønsket udfald. Så i stedet for at lave Pole placement, så kan man vurdere, hvor meget energi man vil bruge.  
**Performance og Effort.**  
Dårlig performance bliver straffet af Q sammen med x, og effort bliver straffet af R med u.

Q ganges med og for at lave en andengradsligning, altså uanset hvad så forbliver værdien positiv, det samme sker med u og R. I vores tilfælde har vi gjort det dyrt at dreje, og billigt at køre lige ud.

Fordele

* Hvor meget energi du vil tillade hvert hjul skal køre
* Hvor meget offset man vil sætte – Q matrix. Hvor præcis skal den være.

Ulemper

* Den kan være svær at styre i forhold til hvor effektiv skal den være at operere?
* Hvor meget skal den korrigere fra den fejl.

Hvor pole placement kæmper for at rette op på fejlen.

## Riccati ligning

Det er en tidsorienteret løsning i diskretiseret tid.

## Fejligning

<https://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?article=2841&context=etd_all>

Ligning for rotation af robottens z-akse

For at simplificere udledningen antages det at rotationen sker i center af robotten, og at vinkelhastigheden for hjulene derfor bliver de samme, dog i modgående retning af hinanden.

Udledning af :

Indsætter på og :

# SLAM Simultaneous Localization And Mapping

Kan oftes opdeles I to grupper –

Filtering

* Extended kalman filter / particle filter
  + Models the problem as an online state estimation.
  + Robotten state opdateres løbende imens den kører.

Smoothing

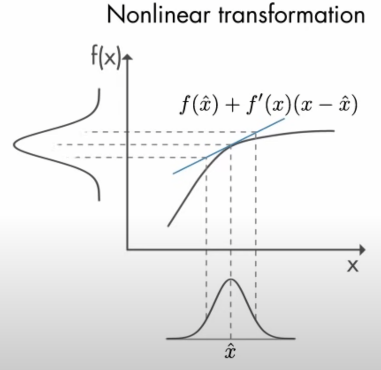
* Måler den kørte rute ud fra alle målinger.

Helt basalt skal man bruge deadreckon, som er en LIDAR til at måle afstand til objekter foran sig, og odometry, altså at kende ændringerne fra sidste position til nuværende, enten med beregninger eller med enkodere.

At lave et map ville være nemt, hvis enkodere og lidar var perfekte. Men det er fejl i både lidar og odometri. Og for hver gang vi tager et nyt snap shot af omgivelserne med LIDAR, så øges fejlen på billedet.

Mest brugte SLAM er graph pose estimates.

Pose graph er alle nodes en pose, og edges imellem dem repræsenterer space representation imellem dem. Optimiseringssystemer finder en bedste node konfiguration, som har mindst fejl, og det er en least squares solution.

Derudover er der Extended kalman filter og particle filter.

Extended kalman filter er en ovenbygning på kalman filtret. Kalman filtret kan kun benyttes på linære systemer. Ved extended kalman filter lineariserer man den nonlinære linje for hvert punkt. Omkring the mean af det nuværende state estimate.

Ulemper:

* Kan være svært at beregne Jacobianten
* Høj computational cost
* Virker kun på systemer, som kan differentieres
* EFK er ikke smart, hvis systemet er meget nonlinært.

Et billede, der indeholder bord

Automatisk genereret beskrivelse

## ICP – Iterative Closest point

<https://www.ipb.uni-bonn.de/html/teaching/msr2-2020/sse2-03-icp.pdf>

Man mapper forskellen imellem to point clouds. Næste skridt er så at prøve at minimere afstanden imellem dem. Det gør man hhv. med en translation og en rotation. Baseret på det gæt, som vi har nu, så genberegner vi data sammenhængen og prøver at lave en ny alligment.

## Partikelfilter

Vi har støjende lidar data og støjende odemetry (lokaliseringsberegninger af robotten fra estimering og enkoer). Sammen med et kalman filter kan man estimere den state. Det kræver at begge er gaussisk distribueret.

Kaldes også for monte carlo

**Template matching**

Man tager et udsnit af et billede, og kører den henover et billede, ligesom man ville med en kernel. For hvert step tjekker man så, hvor meget templaten overlapper med billedet. Det sted(er), hvor den returner den bedste sammenligning vil den så sætte, og man kan få en metric for hvad den kommer med. Når den er meget lille, så er har man et match.

# Ruteplanlægning

## Workspace

## Brushfire

Brushfire algoritmen viser en model-baseret refleksion. Den giver forhindringer en værdi af 1, og som vi kommer længere væk, så øges værdien. Gradienten er pile, som viser retningen mod højere værdier (lysintensitet) hos de omkringliggende celler.

## GVD

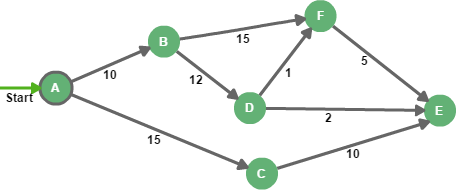
<https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html>

* Skeletonize
  + Den går over billedet flere gange, og hver gang fjerner den en pixel fra kanten. Det fortsætter den med indtil at der ikke kan fjernes flere pixels. Billedet korreleres med en maske med tal imellem 0 og 255, som svarer til hvert muligt mønster fra den 8 omkringliggende pixels. En look-up tabel tildeler pixels en værdier 0, 1, 2 eller 3 og fjernes dem selektivt for hver iteration.
* Skeleton lee
  + Skeleton lee er faktisk en 3D metode. Men kan anvendes på 2D. Den benytter en octree data struktur (Altså 8 børn under den ene node) og tjekker en 3x3x3 kernel omkring pixlen. Den kører over flere gang og fjerner pixels indtil det ikke ændres mere. Den har to skridt: først laver den en liste af kandidater, som kan fjernes, og bagefter tjekker den listen igen for et bibeholde den bedst mulige forbindelse.
* Medial axis
  + Den fungerer ved at alle punkter i den har mere end 1 punkt, som er det tætteste på de omkringliggende objekter. Kaldes ofte en topologisk skelet, da det er et 1 pixel bredt skelet af objektet.

## A\* / Dijkstra

Hvad er forskellen?

## Dijkstra

Man har noder og edges. Hver edge forbinder 2 noder, og har en cost. Dijsktra forsøger at finde frem til den korteste rute uden at udtømme kortet. Man kigger fra startpunktet, og tjekker alle edges, der går ud fra den, og gemmer deres cost. Så tager man den billigest edge først og tjekker noden fra den. Her tjekker man igen alle omkringliggende edges. Og algoritmen førstsætter så med at tjekke den ’billigeste’ rute hver gang, så hvis en ny rute fra node nr. 2 er højere end en rute fra node nr. 1, så tjekker man altså også den anden rute fra node nr. 1. Se eks:

A-B = 10  
A-C = 15  
B-D = 22  
B-F= 25  
C-E = 25

Målet er nu fundet, men da D stadigvæk er mindre, så tjekker vi om den har en edge videre, som stadigvæk er billigere.

D-F = 23  
D-E = 24

Og det samme bliver gjort for F-E, da den også er mindre end D-E.

F-E = 28

Den korteste rute fra A-E er derfor igennem B og D.

Dijkstra er en grådig algoritme.

## A star

Fungerer som Dijkstra, men har også en parameter, som tjekker om vi er tættere på målet i det nye step, altså en heuristic. En oplagt metode er en euclidian distance. Så ligesom Dijkstra, så tildeles hver node en værdi, og i A\* adderes den så med euclidian distance. A\* prioriterer således nodes, som er tættere på målet.

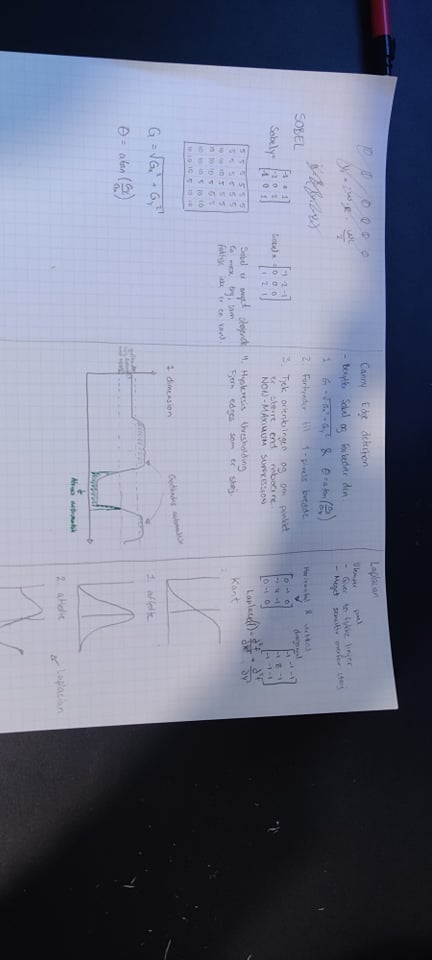
## Cubic spline

## Path Coverage

# Forhindringsdetekering

## Lidar

## Kamera

**Kantdetektering**

* Canny
* Sobel
* Laplacian

**Types of blur**

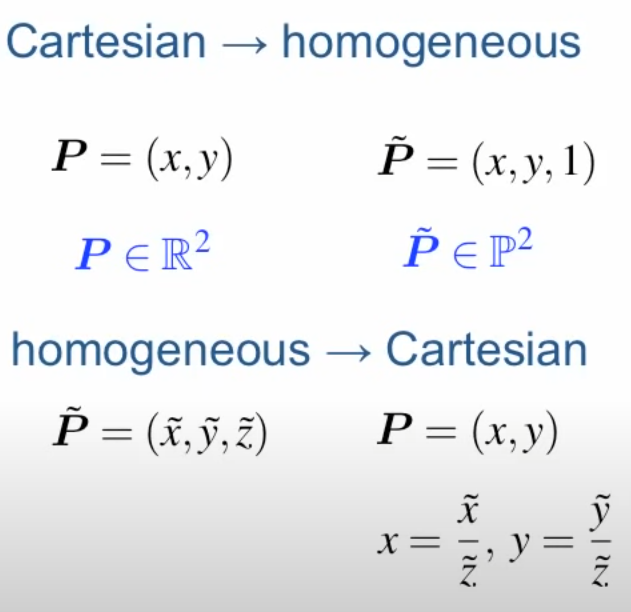
* Gaussian
* Median
  + Den eneste som altid bruger en pixel fra billedet og ikke laver en helt ny værdi
* Bilateral

**Imagefilter.Modefilter**

<https://www.geeksforgeeks.org/python-pil-medianfilter-and-modefilter-method/>

Vælger den hyppigste pixel i kernel.

## Homografi



Man kan finde en homografi med bare fire punkter.

Hvor udtrykker at for ikke-nul skalafaktor .

Det giver følgende ligninger:

Den ovenstående ligning udtrykket 3 ligninger med 8 ukendte homografi enheder. Den tredje ligning () kan substitueres ind i de andre for at få:

Tilsvarende billede bord punkter giver to ligning med 8 ukendte, og vi får 8 ligninger ved at inkludere fire punkter

Enhederne findes ved at løse ligning , hvor , der er den pseudoinverse.

Med opencv bruger man methoden: cv::getPerspectiveTransform(...), der tager fire billedepunkter, og fire bord punkter som input og beregner en homografi imellem dem.

# Hvordan kunne man ellers gøre?

## Segmentering

Edge detection og hough circles.

## Objektdetektering

**Optimer nuværende netværk**

* Flere billeder
* En anden segmentering

**Andre metoder**

* Pytorch / Tensorflow
  + Det skal man lave det selv
* Andre netværk – det kræver så at man bruger en anden metode
* Semantisk segmentering

## Reguleringsteknik – Pole-placement

Alternativt til LQR, for at regulere i forhold til state, x, i stedet for output, y.