


# TSBB18

## Några geometriska verktyg

Ingemar Ragnemalm

# Föreläsningens innehåll

## 3D-transformationer

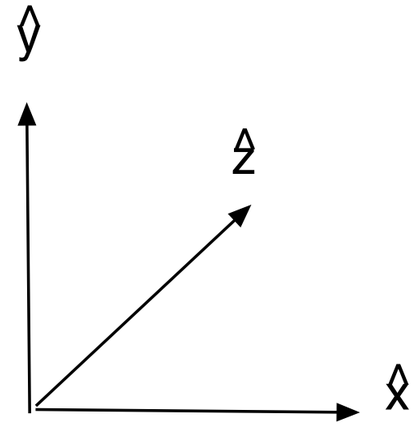
## Hierarkisk modellering

## Projektion

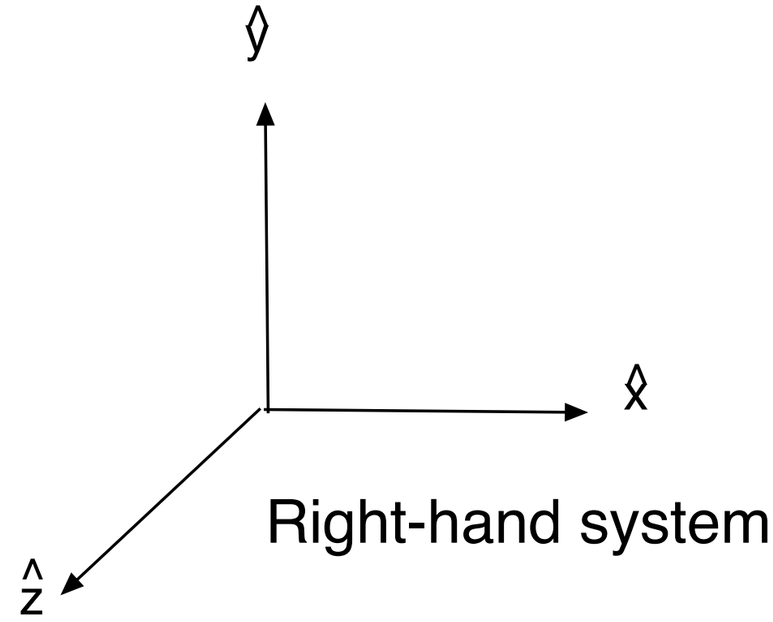
## Homografier

## Inverskinematik

# 3D-koordinatsystem




Left-hand system




Right-hand system

# Transformationer

Beskriv positioneringar och samband i 3D-rymden matematiskt

Translation, rotation, skalning samt projektion.

**August Ferdinand Möbius**

Uppfann homogena koordinater och barycentiska koordinater.

# Homogena koordinater

## Lägg till en "fusk-koordinat"

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix}$$

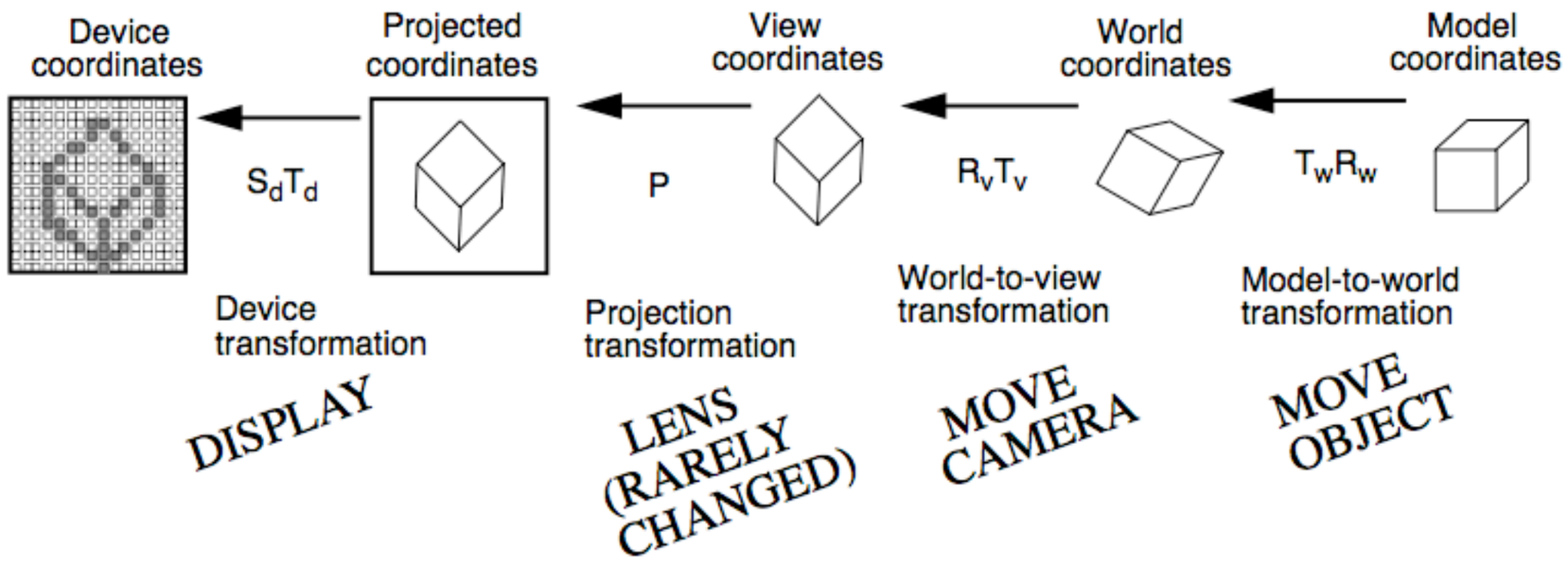
$$x = x_h/h$$

$$y = y_h/h$$

$$z = z_h/h$$

# Transformationskedjan

**Modellkoordinater**
**Världskoordinater**
**Vykoordinater**
**Projicerade koordinater**
**Pixelkoordinater**



# Translation och skalning i 3D:

**Translation: T($t_x$, $t_y$) =**

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

**Translation: T($t_x$, $t_y$, $t_z$) =**

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Scaling: S($s_x$, $s_y$) =**

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Scaling: S($s_x$, $s_y$, $s_z$) =**

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Rotation kring axlarna

**Kring x-axeln:** $R_x(\theta) =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

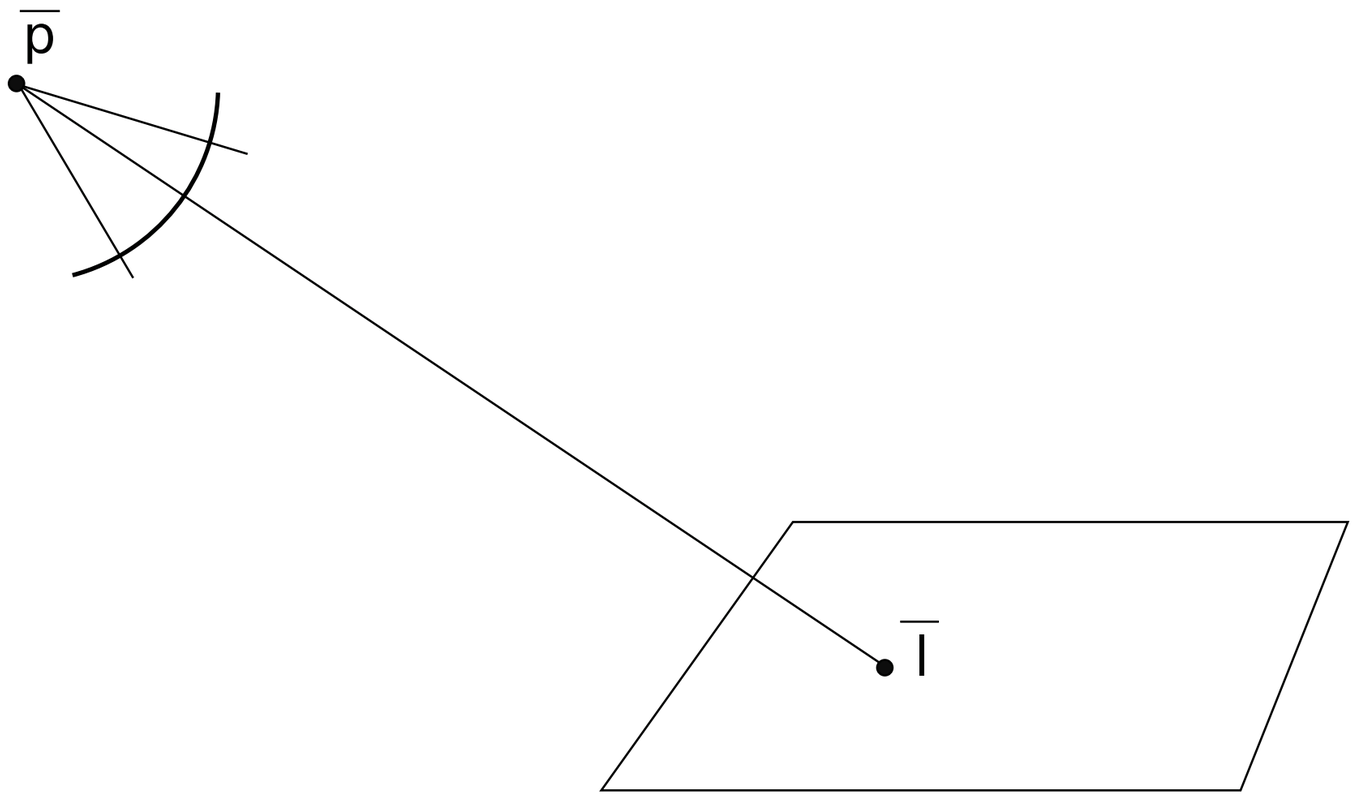
**Kring y-axeln:** $R_y(\theta) =$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Kring z-axeln:** $R_z(\theta) =$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Kameraplacering




Markplan

$$\bar{n} = \bar{p} - \bar{l}$$

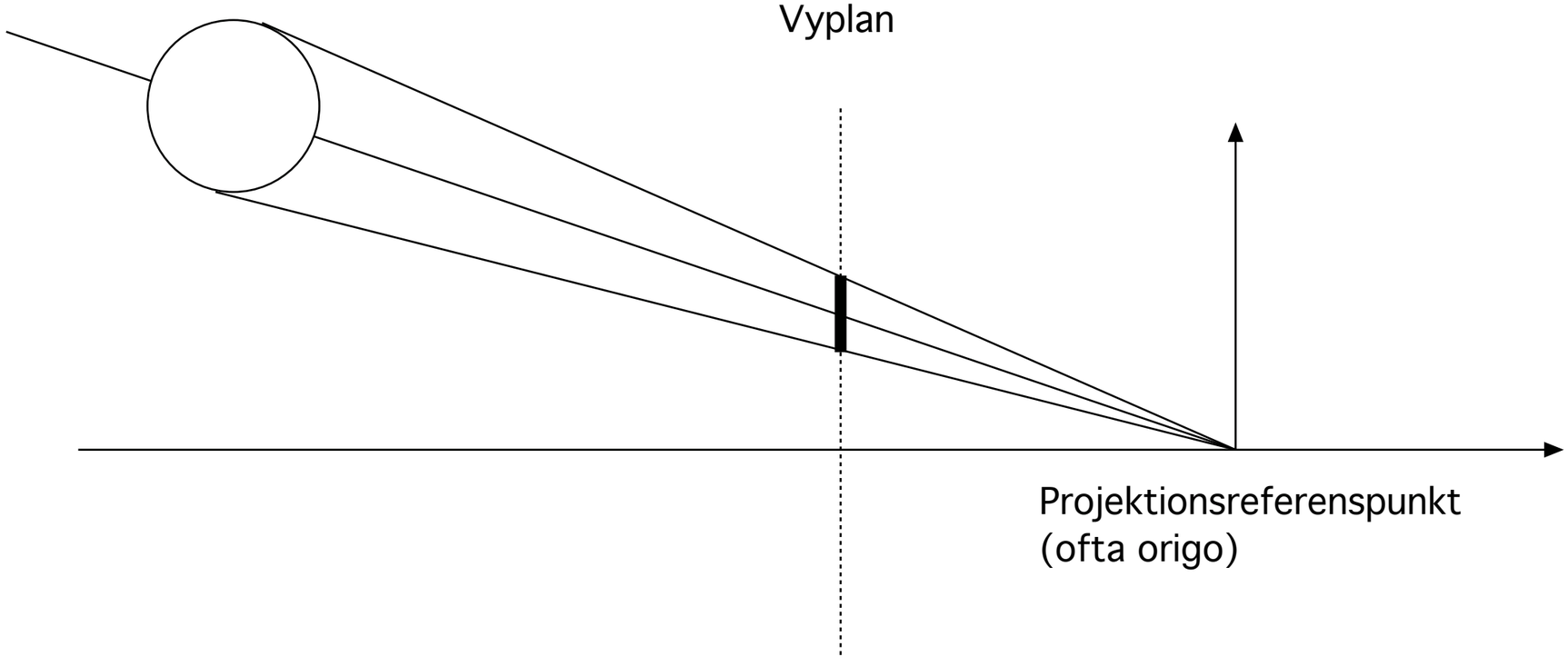
$$\hat{n} = \frac{\bar{p} - \bar{l}}{|\bar{p} - \bar{l}|}$$

$$\hat{u} = \frac{\bar{v} \times \hat{n}}{|\bar{v} \times \hat{n}|}$$

$$\hat{v} = \hat{n} \times \hat{u}$$

$$\begin{bmatrix} u_x & u_y & u_z & -u \bullet \bar{p} \\ v_x & v_y & v_z & -v \bullet p \\ n_x & n_y & n_z & -n \bullet p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Perspektivprojektion



# Projektionsmatris mha nedersta raden med homogena koordinater

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -z_{vp} & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$x_h = f*x$$

$$y_h = f*y$$

$$z_h = -z * zvp$$

$$h = -z$$

**Ngt annorlunda om vyplan läggs i positiv z.**

# Homografier

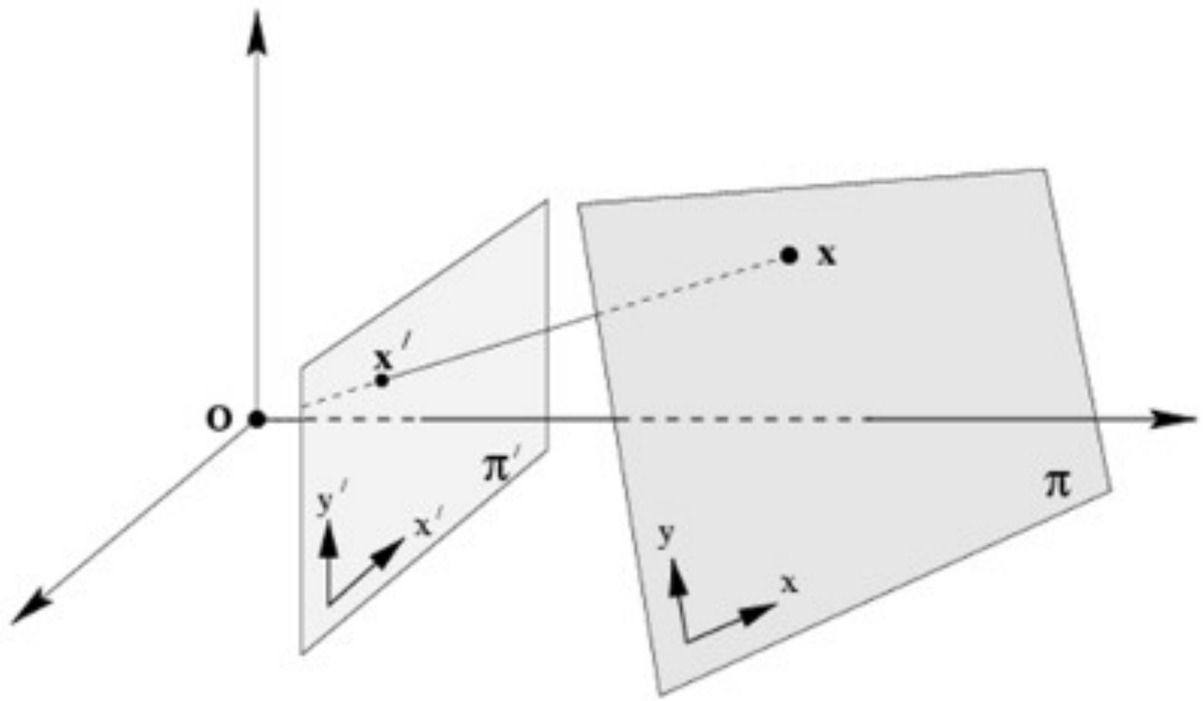
**Projektion är en *homografi* = linjebevarande transform, linjer mappas på linjer.**

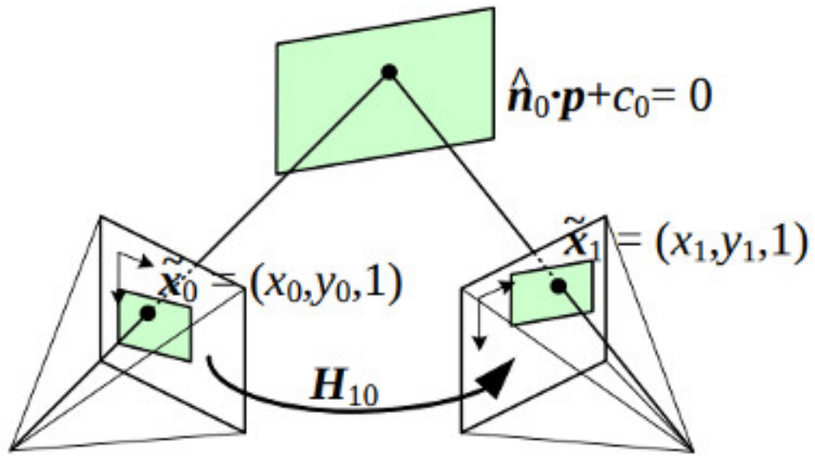
**Beskriver en mappning från ett plan till ett annat.**

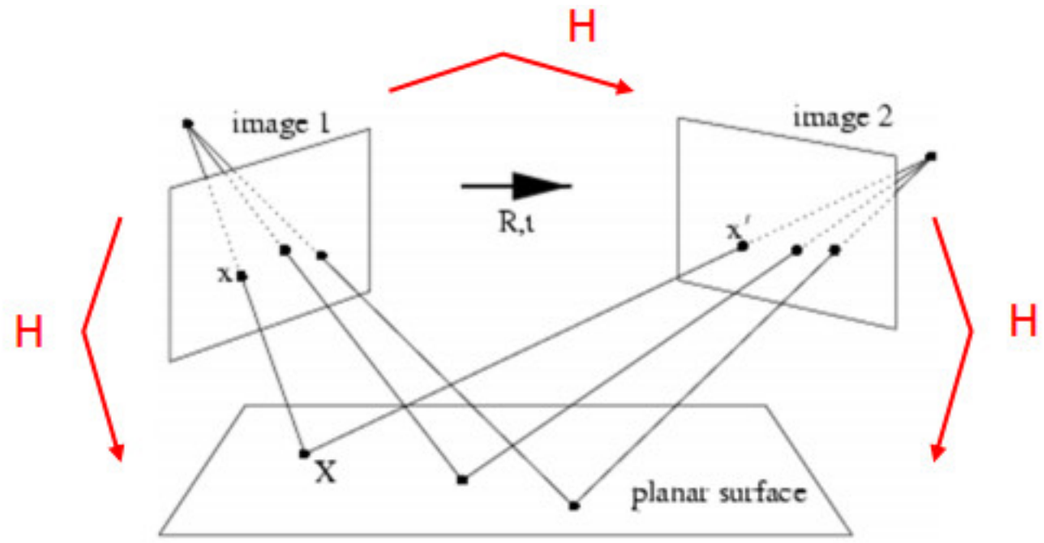
**Baserat på homogena koordinater.**

**Utnyttjas för att beräkna objekts position i en bild.**

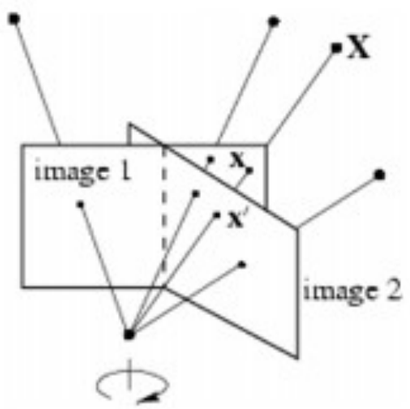
# Mappning från ett plan till ett annat

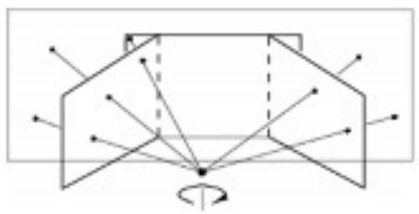


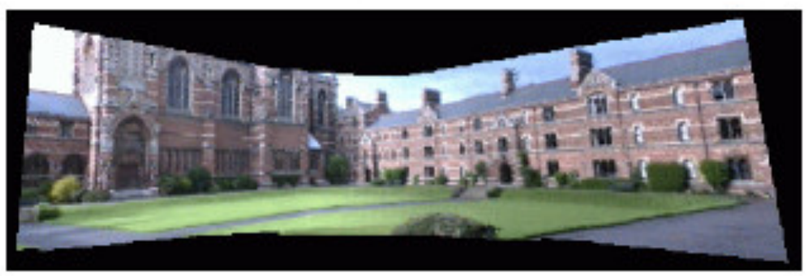



Rotating camera, arbitrary world

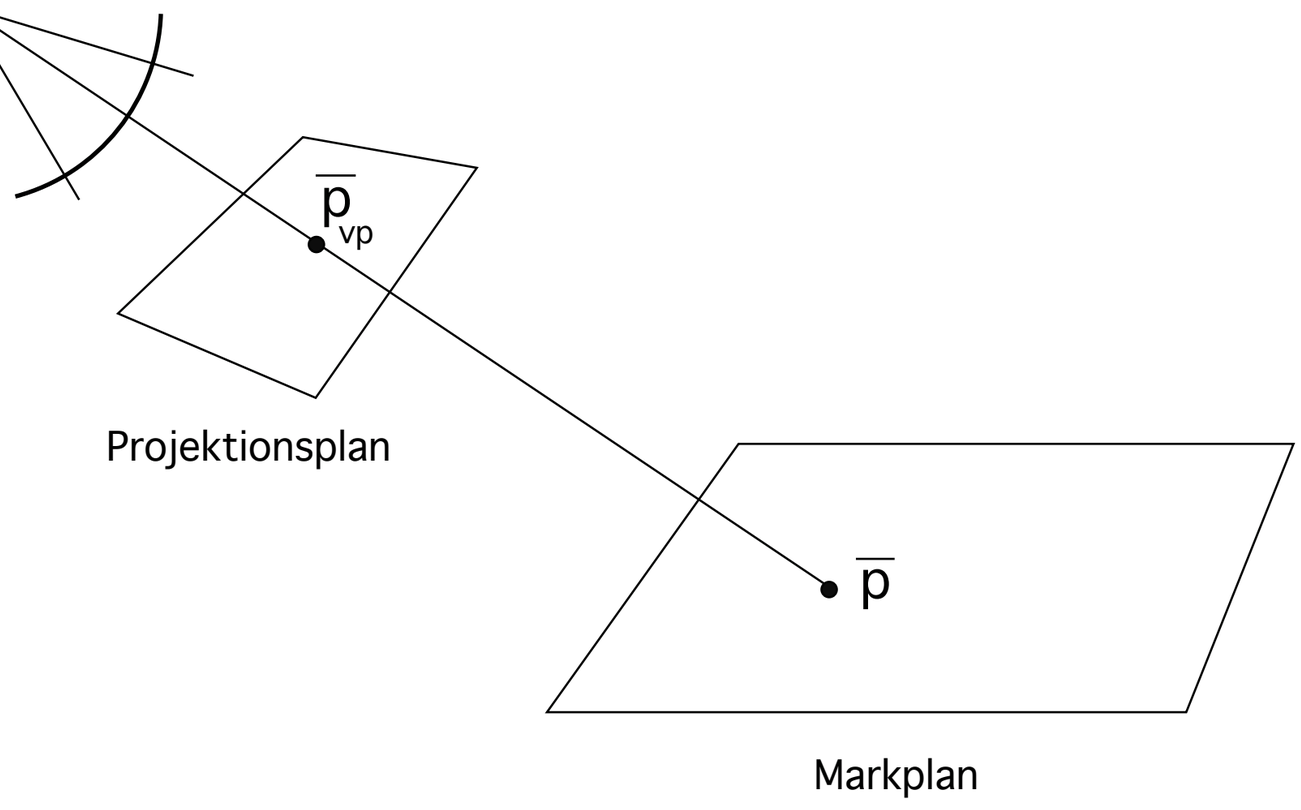




# Homografi för bildanalys

**Projektionsmatrisen är en inverterbar matris H!**




$$\overline{p}_{vp} = H * \overline{p}$$

$$\overline{p} = H^{-1} * \overline{p}_{vp}$$

## OpenCV:

**findHomography**
**calibrateCamera**

# Homografimatrisen

## Matris i homogena koordinater i 2D

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Homografier

**Kraftfullt verktyg om man arbetar med objekt i/på plana ytor.**

**Kräver kamerakaliberering och korrektion för olinjäriteter i lins.**

**Stöds av OpenCV.**

# Kinematik

**Rörelse utan krafter.**

**Forward kinematics: Ange vinklar, position ges av vinklarna. Lätt att implementera, svårt att använda.**

**Inverse kinematics: Ange position, räkna ut vinklar för att uppnå denna position.**

**Viktigt i robotik.**

# Invers-kinematik

- **Analytisk beräkning**
- **Iterativ beräkning med inversa Jacobianen**
- **CCD (Cyclic coordinate descent)**

# Invers-kinematik analytiskt

**Önskad slutposition anges**

**Varje led har ett antal frihetsgrader (vinklar)**

**Överbestämt, olinjärt system**

**Fungerar för små system, ohanterligt för stora.**

# Iterativ metod

**Beräkna derivata av alla utdata m.a.p. invärden.**

**Ger en NxM-matris - Jacobianen.**

**Denna beskriver framåt-kinematiken lokalt.**

**V = J$\theta$'**

**V derivata av utdata (position), $\theta$' derivata på indata**

**$J^{-1}V = \theta'$**

**ger oss derivatan på vinklarna.**

# Iterativ metod

**Jacobianen normalt inte symmetrisk! Ej inverterbar!**

**Pseudoinversen används.**

**Även en variant baserad på transponat av Jacobianen finns.**

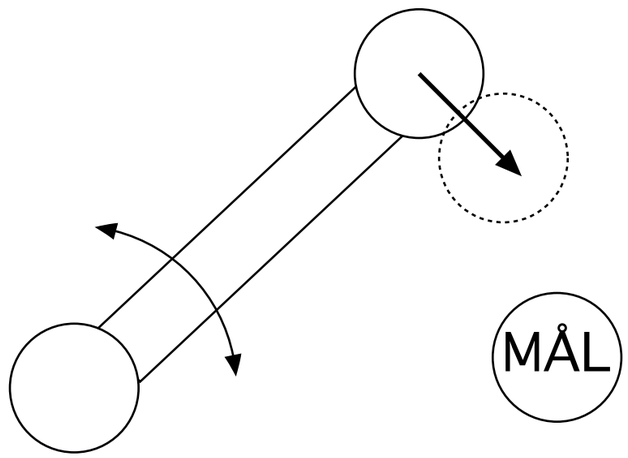
# Iterativ metod (informellt)

**Målposition m**
**Beräkna nuvarande position cp**
**Beräkna position med litet steg för varje vinkel i: $cpd_i$**
**Skillnad mellan avstånd mål-ny position och mål-nuvarande position**

**Möjliga nya vinklar $tr_i$**

**$tr_i := r_i + |\, m-cp \,| - |\, m - cp_i \,|$**

**Vad gjorde jag? Transponat av Jacobianen!**



```
  vec3 cp = ClawPosition(r0, r1, r2, r3, r4);
  vec3 cpd0 = ClawPosition(r0+diff, r1, r2, r3, r4); // Claw position
if r0 is changed
  vec3 cpd1 = ClawPosition(r0, r1+diff, r2, r3, r4);
  vec3 cpd2 = ClawPosition(r0, r1, r2+diff, r3, r4);
  vec3 cpd3 = ClawPosition(r0, r1, r2, r3+diff, r4);
  vec3 cpd4 = ClawPosition(r0, r1, r2, r3, r4+diff);

 // cpd 0-4 - bpd is the Jacobian! Describes the change for xyz
for each set of angles.
 // Change angle in the direction that improves
  float tr0 = r0 + Norm(goal - cp) - Norm(goal - cpd0);
  float tr1 = r1 + Norm(goal - cp) - Norm(goal - cpd1);
  float tr2 = r2 + Norm(goal - cp) - Norm(goal - cpd2);
  float tr3 = r3 + Norm(goal - cp) - Norm(goal - cpd3);
```

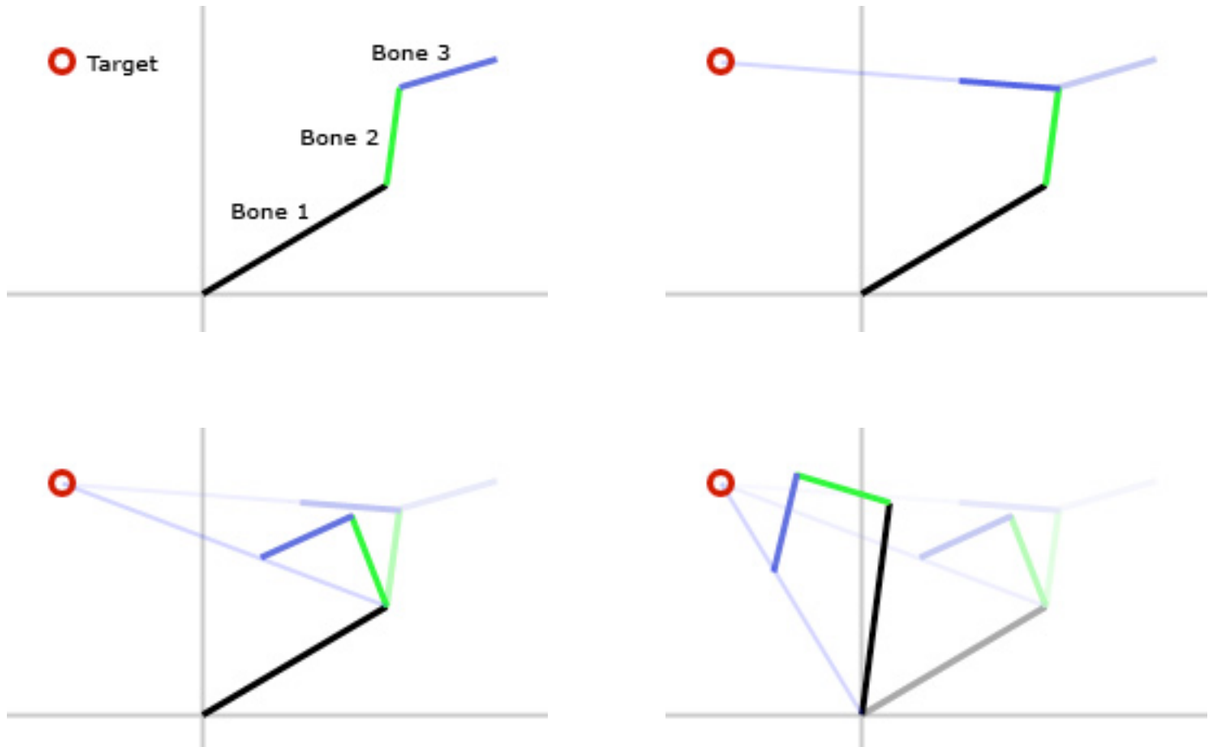
# Cyclic Coordinate Descent

**Iterativ metod som riktar leder explicit. En led i taget vrids åt rätt håll. Arbetar utifrån och in.**

**Vrid en led i taget så att man kommer så nära målet som möjligt. Starta med den yttersta, jobba mot roten.**

**Upprepa tills målet nås eller avståndet är litet nog.**

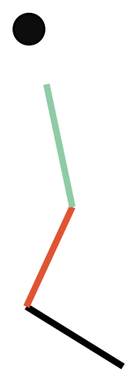
# CCD, exempel




(http://www.ryanjuckett.com)

# Problemfall

**Ofta ger CCD bra resultat med snabb konvergens. Dock, det finns vissa situationer som ger långsam konvergens, och extremfall som inte konvergerar alls (lokal extrempunkt).**



**Konvergerar långsamt**

**Konvergerar inte alls**

# Transponat av Jacobian och Cyclic Coordinate Descent

**Heuristiska metoder, enkla approximationer,
visar sig vara minst lika bra som en analytisk!**

**Både CCD och transponat av Jacobian kan ni implementera.
Enkla, intuitiva, rättframma, ger bra resultat! 😀 Dvs, ofta.**

# Ytterligare villkor för IK

- Begränsningar i möjliga vinklar
- Begränsningar i positioner i rymden
- Säkerhetsavstånd; möjliga positioner som bör undvikas eller kräver långsammare rörelse

Även ytterligare möjliga frihetsgrader: Kolvar/ pistonger som kan ändra längden på en led.

# Frågor?