

Ch6 Image Transforms

老師：連震杰 教授

Robotics Lab
Institute of Computer Science and Information Engineering,
National Cheng Kung University, Tainan, Taiwan, R.O.C.

1. Introduction

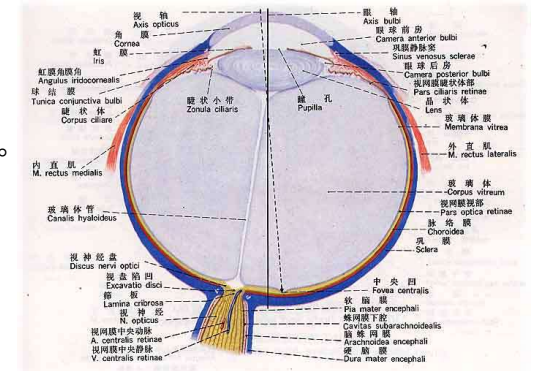
- Hough Transform是圖像處理中識別幾何形狀的一種方法，在圖像處理中有著廣泛應用，霍夫變換不受圖形旋轉的影響，易於進行幾何圖形的快速變換。
 - 簡單來說有三種
 - 1) Circle Hough Transform
 - 2) Line Hough Transform
 - 3) Generalize Hough Transform

2. Iris Detection Using Circle Hough Transform (CHT) jj, ??

◆ 2.0 Iris

➤ 虹膜(Iris)眼睛構造

- 虹膜(Iris)屬於眼球中層，位於血管膜的最前部，在睫狀體前方，
- 虹膜中央有瞳孔(Pupil)，由虹膜調節瞳孔的大小進入眼內光線的作用。



➤ 虹膜(Iris)識別系統 - 以手機的內置的拍照功能來獲得高畫質的瞳孔影像。

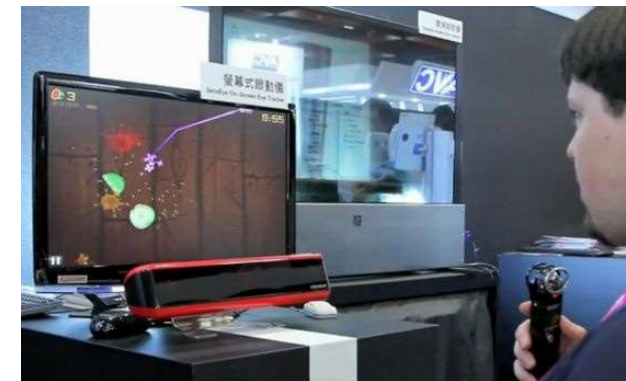


此應用程序需要用戶先利用手機相機拍下使用者眼睛圖片。
然後分析該圖片的虹膜紋理圖案，與存儲的範本核對，
以確認個人的真實身份。

➤ 虹膜(Iris)控制系統 - 由高畫質攝影機取得瞳孔影像，

經由瞳孔偵測與定位取得座標，

由座標的改變進而達到人機互動的效果。



2.0 Iris Detection using Circle Hough Transform

3.1 Canny Edge Detection

- 1) Image Convert to Grayscale
- 2) Noise Reduction/Smoothing
- 3) Compute Gradient Magnitude and Angle
- 4) Non-Maximum Suppression
- 5) Hysteresis Thresholding: Low and High Thresholds



E_t : Eye region image
 C_t : Binary image of Canny edge detection result

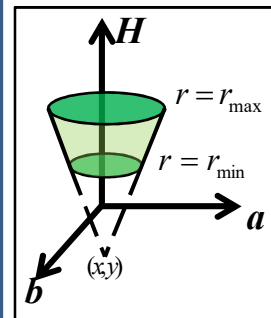
$$C_t = \text{canny}(E_t)$$

$$C_t(x, y) = \begin{cases} 1, & \text{edge} \\ 0, & \text{not edge} \end{cases}$$

Canny Result
 $C_t(x, y)$

3.2~3.5 Iris Detection using Circle Hough Transform

3.2 Parameter Space



(x, y) transform to (a, b)

$$\begin{aligned} a &= x + r \cos \theta \\ b &= y + r \sin \theta \end{aligned}$$

$$\begin{cases} H: \{r \mid r_{\min} \leq r \leq r_{\max}\} \\ 0^\circ \leq \theta < 360^\circ \end{cases}$$

3.3 For all edge point (x, y) in image space, calculated

- 1) Select radius which start from $r = r_{\min}$, $r \in (r_{\min}, r_{\max})$
- 2) Calculated (a, b) of all the circle which via (x, y)
- 3) For each edge point (x, y) , increment the corresponding elements in the accumulator (a, b) .

-Find the next edge point (x, y) , repeat steps 2

and 3 until all edge points are calculated.

-Repeat 1 until $r = r_{\max}$

3.4 Set the threshold of accumulator: Find all center

(a, b) which exceed this threshold in accumulator.

3.5 ??Delete/Merge Similar Circles: Collect candidate center, then delete similar circles.

2.1 Canny Edge Detection Algorithm **jj3**

➤ Canny Algorithm Steps:

Canny比 Sobel 多做了2.4.5項

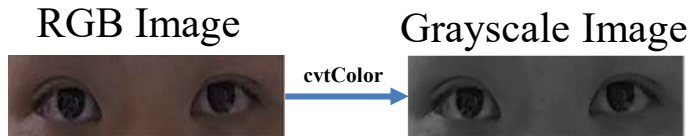
1. Image Convert to Grayscale
2. **Smoothing:** Noise Reduction (Smoothing)
3. **Edge Detection:** Compute Gradient Magnitude and Angle using Sobel
4. **Thinning:** Non-Maximum Suppression
5. **Connectivity:** Hysteresis Thresholding: Low and High Thresholds

➤ OpenCV Canny Function Call

- ### ➤ Edge detection performance:
- Wavelets edge detection (slow) > canny > sobel (fast)

2.1 Canny Algorithm – Steps (1/3) **jj, ??**

1. Image Convert to Grayscale



`cvtColor(image, gray_image, CV_BGR2GRAY)`

Parameters:

- 1) **image** – a source image (image)
- 2) **Gray_image** - a destination image (gray_image),
in which we will save the converted image.
- 3) **CV_BGR2GRAY** - an additional parameter that indicates
what kind of transformation will be performed.

High-pass filter, sum=0.0

3. Compute Gradient Magnitude and Angle

- Compute the derivatives ($D_x(x, y)$ and $D_y(x, y)$)
of the image in the x and y directions.

Ex: Sobel filter

Derivative =
slope,
High freq.

$$D_x(x, y) = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$D_y(x, y) = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

OpenCV 2.4.8.0 documentation » OpenCV Tutorials :

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Low-pass filter, sum=1.0

2. Noise Reduction (Smoothing)

-Note that this 5×5 filter is roughly
equivalent to the Gaussian filter

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$$\frac{1}{159} *$$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

J: -Usually 3x3 is enough, normally, 7x7
is too big. We don't want it filters out the
high-frequency components. So wavelet
is one solution but not in this case.

- Compute the gradient magnitude:

$$D = \sqrt{D_x^2(x, y) + D_y^2(x, y)}$$

J:?? Should D normalizes between 0~255 in order fro
later threshold?

- Compute the angle of the gradient:

$$\theta = \arctan\left(\frac{D_y(x, y)}{D_x(x, y)}\right)$$

After this process, what will the edge looks like?

2.1 Canny Algorithm – Steps (2/3) jj, ??

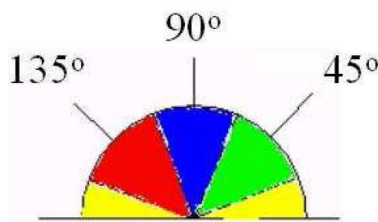
4. Non-Maximum Suppression (thinning)

➤ Goal: The edges it finds can be either very thick or very narrow depending on the intensity across the edge and how much the image was blurred first.

➤ Non-Maximum Suppression:

Three pixels in a 3×3 around pixel (x, y) are examined, there are 4 conditions:

- If $\theta'(x, y) = 0^\circ$, then the pixels $(x + 1, y)$, (x, y) , and $(x - 1, y)$ are examined.
- If $\theta'(x, y) = 90^\circ$, then the pixels $(x, y + 1)$, (x, y) , and $(x, y - 1)$ are examined.
- If $\theta'(x, y) = 45^\circ$, then the pixels $(x + 1, y + 1)$, (x, y) , and $(x - 1, y - 1)$ are examined.
- If $\theta'(x, y) = 135^\circ$, then the pixels $(x + 1, y - 1)$, (x, y) , and $(x - 1, y + 1)$ are examined.



Compute θ' by rounding the angle θ to one of four directions $0^\circ, 45^\circ, 90^\circ$, or 135° .

Obviously for edges, $180^\circ = 0^\circ$, $225^\circ = 45^\circ$, etc.

➤ Choose edge pixel :

- If pixel (x, y) has the highest gradient magnitude of the three pixels examined for any one condition, it is kept as an edge.
- If one of the other two pixels has a higher gradient magnitude, then pixel (x, y) is not

on the “center” of the edge and should not be classified as an edge pixel. (如果周圍比自己大就忽略)

J: Will this kind of judgment cause slow performance?

After this process, what will the edge looks like?

2.1 Canny Algorithm – Steps (3/3) jj

5. Hysteresis Thresholding: Two thresholds – t_{low} and t_{high} (connectivity)

- Existing problem (solve by canny): A simple threshold may actually remove valid parts of a connected edge, leaving a disconnected final edge image.

J: It wants to keep connected edge/components.

➤ Hysteresis Thresholding

- If pixel (x, y) has gradient magnitude less than t_{low} discard the edge (write out black).
- If pixel (x, y) has gradient magnitude greater than t_{high} keep the edge (write out white).
- If pixel (x, y) has gradient magnitude between t_{low} and t_{high} and any of its neighbors in a 3×3 region around it have gradient magnitudes greater than t_{high} , keep the edge (write out white).
- If none of pixel (x, y) 's neighbors have high gradient magnitudes but at least one falls between t_{low} and t_{high} , search the 5×5 region to see if any of these pixels have a magnitude greater than t_{high} . If so, keep the edge (write out white).
- Else, discard the edge (write out black).

J: Will this kind of judgment cause slow performance?

2.1 Canny Algorithm – OpenCV Function Call **jj, ??**

➤ **Void Canny**(InputArray **detected_edges**, OutputArray **detected_edges**,
double **lowThreshold**, double **highThreshold**, int **kernel_size**)

Parameter:

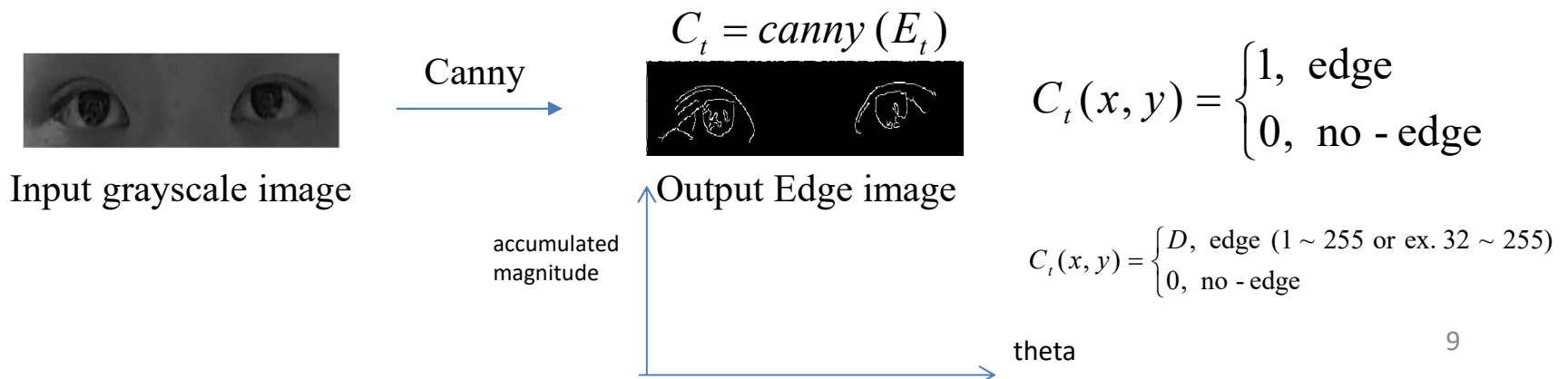
- 1) **detected_edges**: Source image, grayscale
- 2) **detected_edges**: Output of the detector (can be the same as the input)
- 3) **lowThreshold**: The value entered by the user moving the Trackbar
- 4) **highThreshold**: Set in the program **as three times** the lower threshold
(following Canny's recommendation)
- 5) **kernel_size**: We defined it to be 3 (the size of the Sobel kernel to be used internally)

Ex:

-1	0	+1
-2	0	+2
-1	0	+1

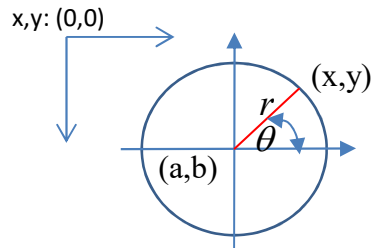
+1	+2	+1
0	0	0
-1	-2	-1

Kernel = window(表格) = filter



2.2 Circle Hough Transform (CHT) – Parameter Space (1/2) **jj**

- If an image (x-y domain or space) contains many points, some of which fall on perimeters of circles, then the job of the search program is to **find parameter triplets (a, b, H)** to describe each circle.
- x-y space: A circle with radius r and center (a, b) can be described with the parametric equations



Parametric circle in x-y domain

$$\text{General form : } (x - a)^2 + (y - b)^2 = r^2$$

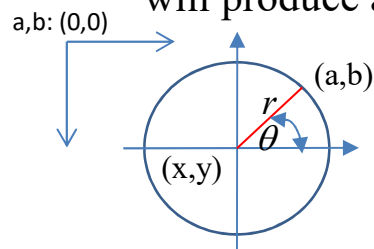
$$\text{Parametric form : } \begin{cases} x = a + r \cos \theta \\ y = b + r \sin \theta \end{cases} \quad \begin{cases} H\{r \mid r_{\min} \leq r \leq r_{\max}\} \\ 0^\circ \leq \theta < 360^\circ \end{cases}$$

(In x-y domain)

r (已知) , 固定(a, b)求(x, y)

When the angle θ sweeps through the full 360 degree range
the edge points (x, y) trace the perimeter of a circle.

- a-b parameter space: Each edge point (x, y) on the perimeter of a circle will produce a cone surface in parameter space.



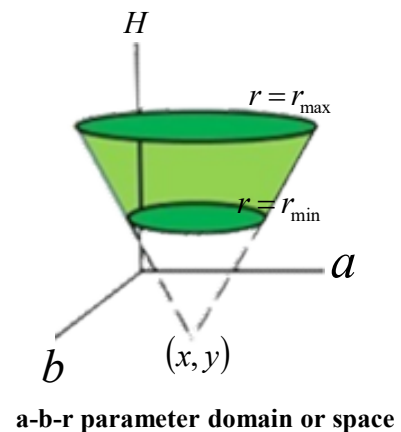
Parametric circle in a-b domain

$$\text{General form : } (a - x)^2 + (b - y)^2 = r^2$$

$$\text{Parametric form : } \begin{cases} a = x + r \cos \theta \\ b = y + r \sin \theta \end{cases} \quad \begin{cases} H\{r \mid r_{\min} \leq r \leq r_{\max}\} \\ 0^\circ \leq \theta < 360^\circ \end{cases}$$

(In a-b domain)

r (已知) , 固定(x, y)求(a, b)



a-b-r parameter domain or space

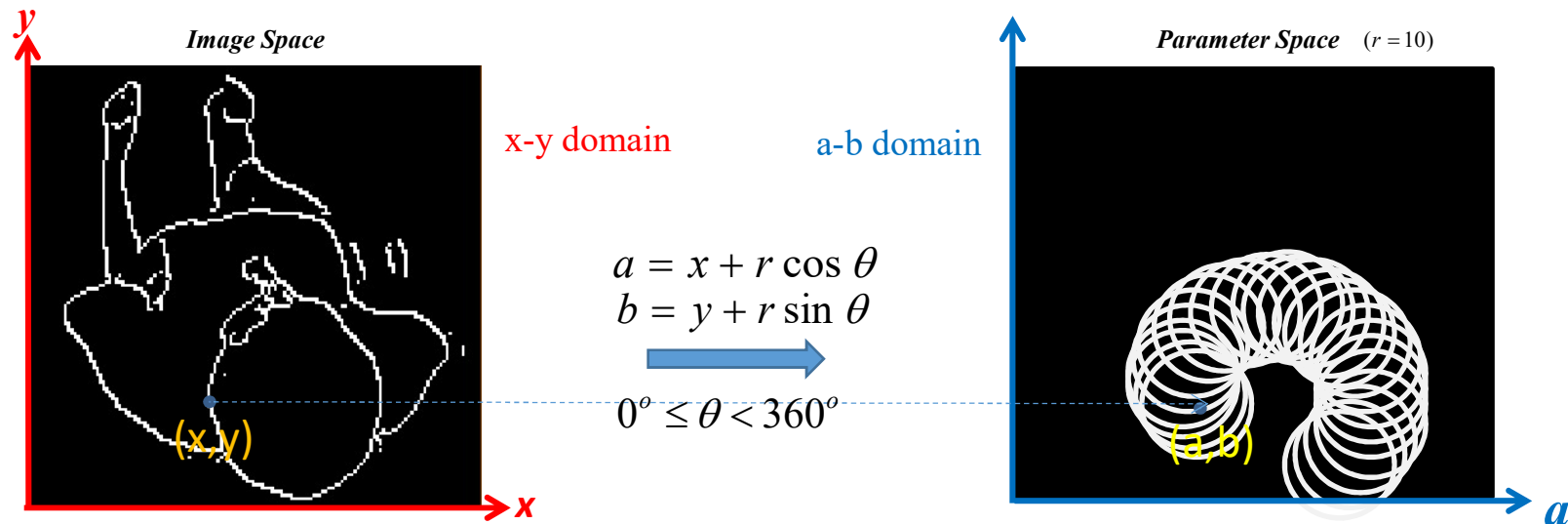
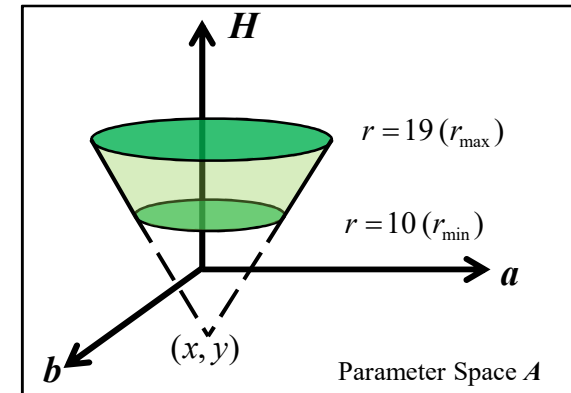
2.2 Circle Hough Transform (CHT) – Parameter Space (2/2) **jj**

➤ Parameter Space for Example

if ($C_i(x, y) = 1$)

$$\begin{aligned} a &= x + r \cos \theta \\ b &= y + r \sin \theta \end{aligned} \quad \left\{ \begin{array}{l} H\{r \mid r_{\min} \leq r \leq r_{\max}\} \\ 0^\circ \leq \theta < 360^\circ \end{array} \right.$$

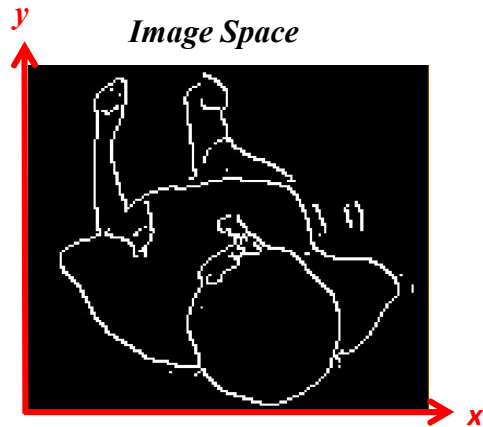
$A[r, a, b]$: Accumulator's value



2.3 Circle Hough Transform (CHT) – Accumulator (1/2) **jj**

➤ Accumulator for Example

For each feature point (x,y)



if ($C_t(x, y) = 1$)

$$C_t(x, y) = \begin{cases} 1, & \text{edge} \\ 0, & \text{no - edge} \end{cases}$$

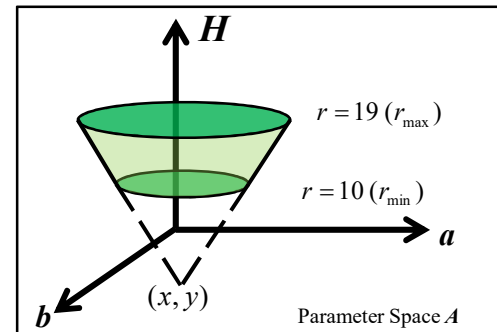
A: 做canny的目的是
我們只針對canny的
edge上的所有點進行
參數空間的轉換與投票



$$\begin{aligned} a &= x + r \cos \theta \\ b &= y + r \sin \theta \end{aligned}$$

$$0^\circ \leq \theta < 360^\circ$$

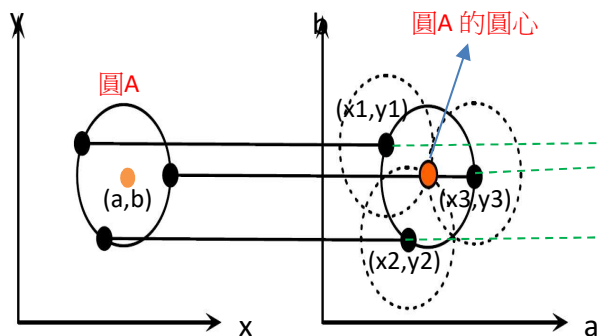
1. Select radius $r \in (r_{\min}, r_{\max})$ $H\{r \mid r_{\min} \leq r \leq r_{\max}\}$



2.3 Circle Hough Transform (CHT) – Accumulator (2/2) jj

- J: 1. 在 (x,y) domain 的一個點,經過轉換式,在 (a,b) domain 是同心圓的圓心 ,
此原有個虛擬的圓心 (x,y) , 也就是跟原來 (x,y) 位置一樣但現在在 (a,b) domain.
2. 在 (x,y) domain 裡 , 如果是同一個 圓A 的點 , 假設 N 個點,在 (a,b) domain 裡會產生 N 個圓 ,
這些圓會產生一個 voting 值最大的交點 , 此交點就是為原來 (x,y) domain 圓A 的圓心.

2. Calculated (a, b) of all the circle which via (x, y) .



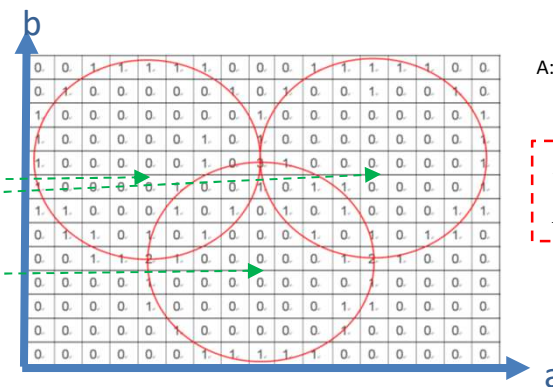
Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

A: 圓形定義: 固定一個圓心 (a,b)
與一個半徑 r , 找出與這個圓心
有相同距離 r 的所有點, 所形成
的圖形就是相對於 (a,b) 的一個圓)

$$\text{General form : } (x - a)^2 + (y - b)^2 = r^2 \quad x, y \in I$$

$$\begin{aligned} \text{Parametric form : } x &= a + r \cos \theta \\ y &= b + r \sin \theta \\ 0^\circ &\leq \theta < 360^\circ \end{aligned} \quad x, y \in I$$

3. Calculated Accumulator



A: 每一個格子表示一個 pixel

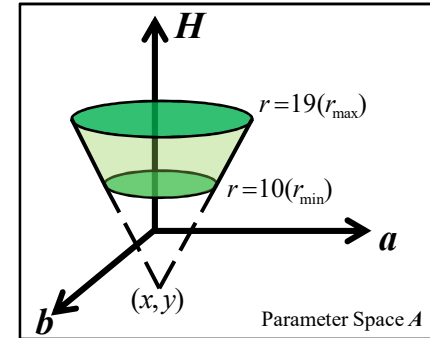
$A[r, a, b]$:
Accumulator's value

A: 圓形的參數空間: 逆其道而行, 這次改由固定 (x,y) 點
假使固定半徑 r , 目的是找出有通過 (x,y) 點, 距離為 r
的所有圓心, 此所有圓心構成的圖抑是一個圓, 假使
我們要在 (x,y) 平面上找出此三點的共同圓心, 則在
他們的參數空間找出它們所構成的參數於是否交集
在同一個點, 以此 case 所測得圓心就是計數器中得
最多票 (3 票) 的點.

2.4 Circle Hough Transform (CHT) – Threshold **jj**

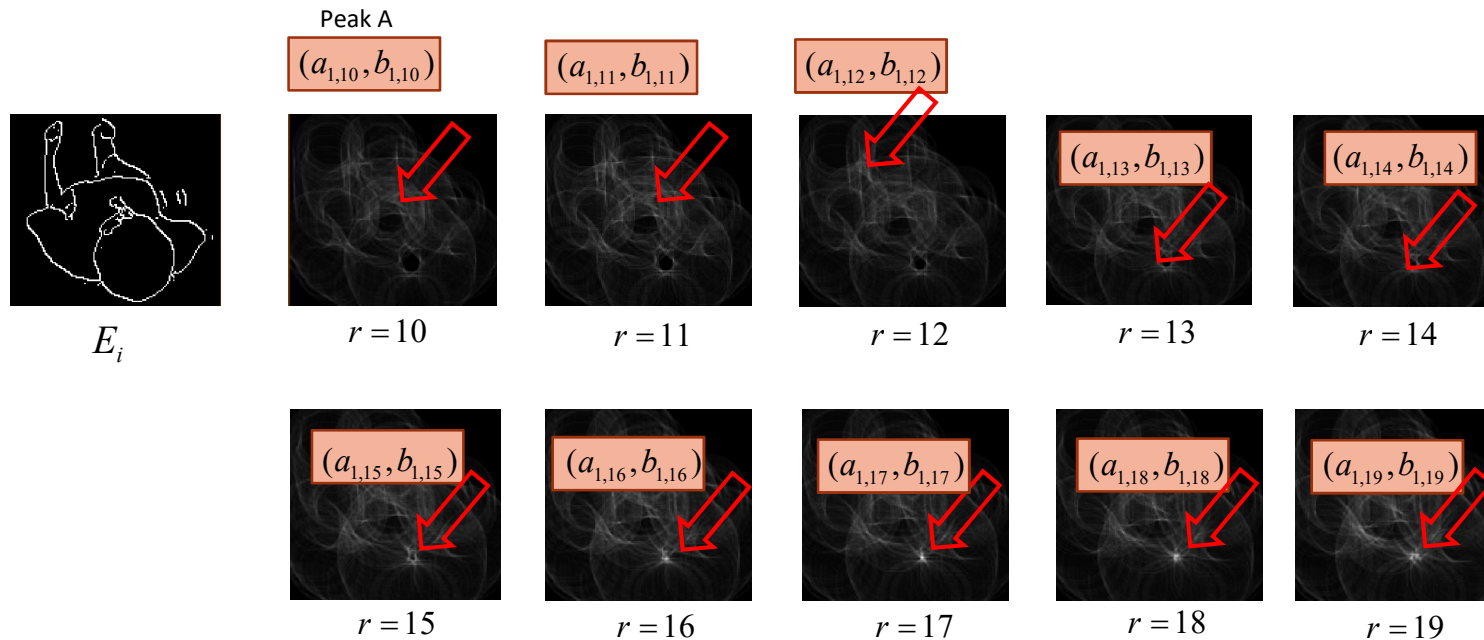
➤ Set the threshold in Accumulator

Find candidate position by **Accumulator's value**
which exceed the threshold at each layer in Space A



$$\begin{aligned} a &= x + r \cos \theta \\ b &= y + r \sin \theta \quad a, b \in I \end{aligned}$$

$$\begin{cases} H\{r \mid r_{\min} \leq r \leq r_{\max}\} \\ 0^\circ \leq \theta < 360^\circ \end{cases}$$



- 1) At the same radius $r=17$, merge neighbor centroids to be one final centroid by voting weights.
- 2) Find max voting value, ex. $r=17$

2.5 Circle Hough Transform (CHT) – Delete/Merge Similar Circles **jj**

1. Set threshold for 圓心點最短距離
2. Same r: Check each center, 如果該可能的圓心點和已經確認的圓心點的距離小於threshold,則表示圓心點和已經確認的圓心點是**同一個點**, 就把此點移除.
J: Merge method:
1) Max
2) Weighting based on accumulator values
3. Different r: 假使同一個圓心但半徑不一樣, 則會比較兩者的 accumulator's value, 並**留下最大者**.
4. Check 完所有的圓心點, 將還留下的圓心與半徑顯示在圖上.

2.6 Circle Hough Transform (CHT) – OpenCV Function Call **jj, ??**

- **Void HoughCircles** (InputArray **src_gray**, OutputArray **circles**, int **method**, double **dp**, double **min_dist**,
double param1(CannyThreshold), double **param2**, int **minRadius**, int **maxRadius**)

Parameters:

1. **src_gray**: Input image (grayscale).
2. **circles**: A vector that stores sets of 3 values: vector<Vec3f> (xc; yc; r)for each detected circle.
3. **CV_HOUGH_GRADIENT**: Define the detection method.
Currently this is the only one available in OpenCV.
4. **dp** : Resolution of the accumulator used to detect centers of the circles.
For example, if it is 1, the accumulator will have the same resolution as the input image,
if it is 2 - accumulator will have twice smaller width and height, etc.
5. **min_dist** : Minimum distance between centers of the detected circles.
If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one.
If it is too large, some circles may be missed.
6. **param_1** : The first method-specific parameter.
In case of CV_HOUGH_GRADIENT it is the higher threshold of the two passed to Canny edge detector
(the lower one will be twice smaller).
7. **param_2** : The second method-specific parameter.
In case of CV_HOUGH_GRADIENT it is accumulator threshold at the center detection stage.
The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values,
will be returned first. **J: Can be x % of 2 Pi r**
8. **min_radius** : Minimum radio to be detected. If unknown, put zero as default.
9. **max_radius** : Maximum radius to be detected. If unknown, put zero as default.

Function: Detects circles of different sizes in the input image.

The detected circles are returned as a list of storage.

Example: `HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT,
1, src_gray.rows/8, 200, 100, 98, 102)`

dp = 1 – 累加器圖像的解析度, 此參數允許輸入一個比原
圖像低的解析度, 設置為1表示與原圖解析度相同
, 假使此參數越大, 則受到解析度的影響就越小.

min_dist – 用來區分兩個不同圓的圓心最小距離.
`src_gray.rows/8`表示兩個圓心超過此距離則
判定是不同圓的圓心.

param_1 = 200 - 為canny的max threshold = 200

Param_2 = 100 – 為累加器的threshold = 100

min_radius = 98 – 偵測圓的最小半徑

max_radius = 102 – 偵測圓的最大半徑

3. Lane Detection Using Hough Line Transform (1/2)

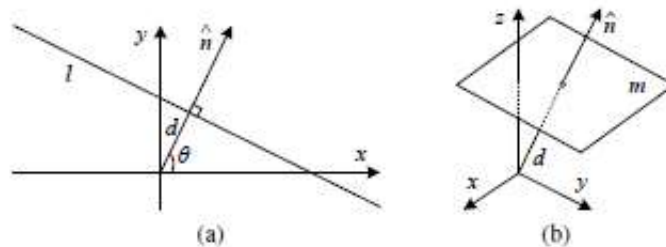


Figure 2.2 (a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal \hat{n} and distance to the origin d .

2D points. 2D points (pixel coordinates in an image) can be denoted using a pair of values, $\mathbf{x} = (x, y) \in \mathcal{R}^2$, or alternatively,

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}. \quad (2.1)$$

(As stated in the introduction, we use the (x_1, x_2, \dots) notation to denote column vectors.)

2D points can also be represented using *homogeneous coordinates*, $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$, where vectors that differ only by scale are considered to be equivalent. $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$ is called the 2D *projective space*.

A homogeneous vector $\tilde{\mathbf{x}}$ can be converted back into an *inhomogeneous* vector \mathbf{x} by dividing through by the last element \tilde{w} , i.e.,

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\tilde{\mathbf{x}}, \quad (2.2)$$

where $\tilde{\mathbf{x}} = (x, y, 1)$ is the *augmented vector*. Homogeneous points whose last element is $\tilde{w} = 0$ are called *ideal points* or *points at infinity* and do not have an equivalent inhomogeneous representation.

3. Lane Detection Using Hough Line Transform (2/2)

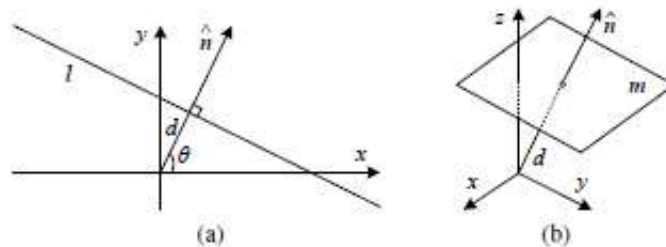


Figure 2.2 (a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal \hat{n} and distance to the origin d .

2D lines. 2D lines can also be represented using homogeneous coordinates $\tilde{l} = (a, b, c)$. The corresponding *line equation* is

$$\bar{x} \cdot \tilde{l} = ax + by + c = 0. \quad (2.3)$$

We can normalize the line equation vector so that $l = (\hat{n}_x, \hat{n}_y, d) = (\hat{n}, d)$ with $\|\hat{n}\| = 1$. In this case, \hat{n} is the *normal vector* perpendicular to the line and d is its distance to the origin (Figure 2.2). (The one exception to this normalization is the *line at infinity* $\tilde{l} = (0, 0, 1)$, which includes all (ideal) points at infinity.)

We can also express \hat{n} as a function of rotation angle θ , $\hat{n} = (\hat{n}_x, \hat{n}_y) = (\cos \theta, \sin \theta)$ (Figure 2.2a). This representation is commonly used in the Hough transform line-finding algorithm, which is discussed in Section 4.3.2. The combination (θ, d) is also known as polar coordinates.

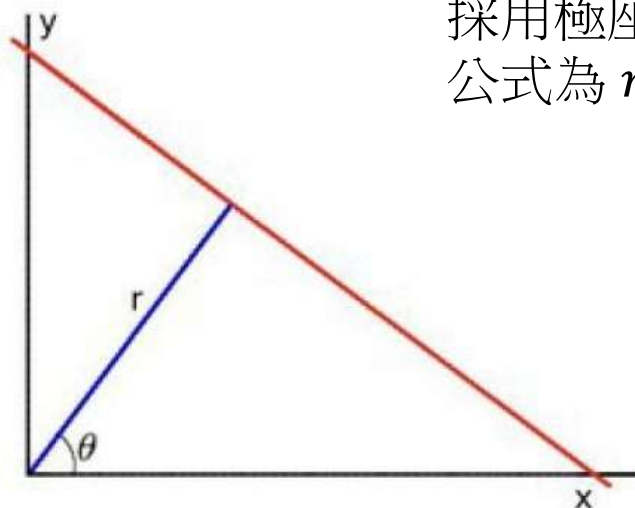
3. Lane Detection Using Hough Line Transform

3.1 Hough Line Transform Theory

How does it work?

1. As you know, a line in the image space can be expressed with two variables. For example:

- (a) In the **Cartesian coordinate system**: Parameters: (m, b) .
- (b) In the **Polar coordinate system**: Parameters: (r, θ)



採用極座標 (r, θ) ，極座標與直角坐標轉換公式為 $r = x \cos \theta + y \sin \theta$

For Hough Transforms, we will express lines in the *Polar system*. Hence, a line equation can be written as:

$$y = \left(-\frac{\cos \theta}{\sin \theta} \right) x + \left(\frac{r}{\sin \theta} \right)$$

Arranging the terms: $r = x \cos \theta + y \sin \theta$

3.1 Hough Line Transform Theory

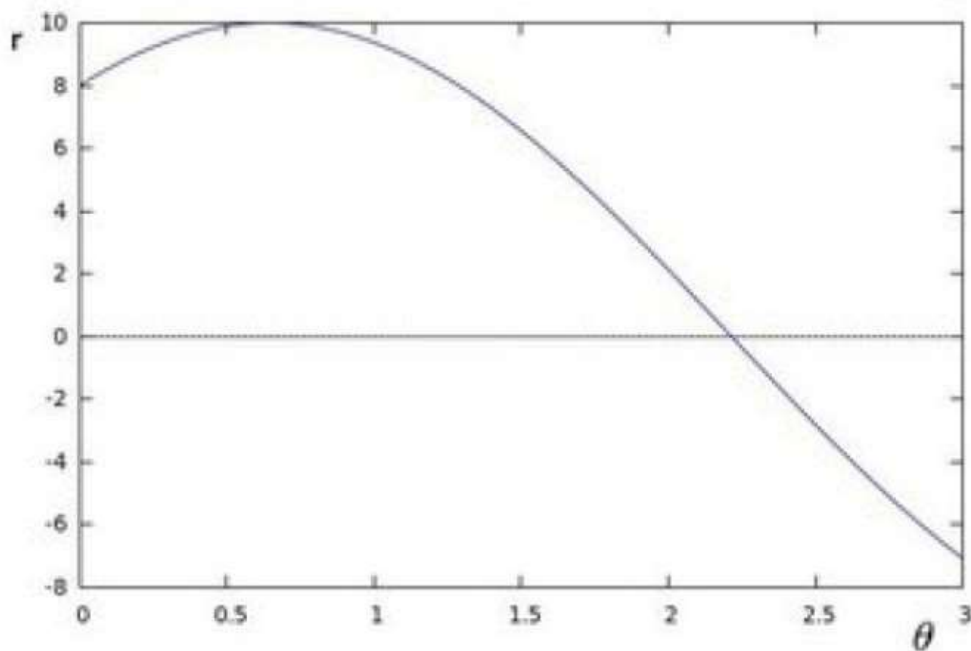
1. In general for each point (x_0, y_0) , we can define the family of lines that goes through that point as:

$$r_\theta = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$$

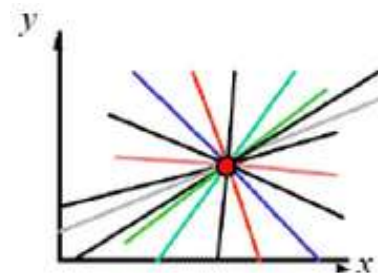
Meaning that each pair (r_θ, θ) represents each line that passes by (x_0, y_0) .

用 (r_θ, θ) 代表通過 (x_0, y_0) 的線段

2. If for a given (x_0, y_0) we plot the family of lines that goes through it, we get a sinusoid. For instance, for $x_0 = 8$ and $y_0 = 6$ we get the following plot (in a plane $\theta - r$):



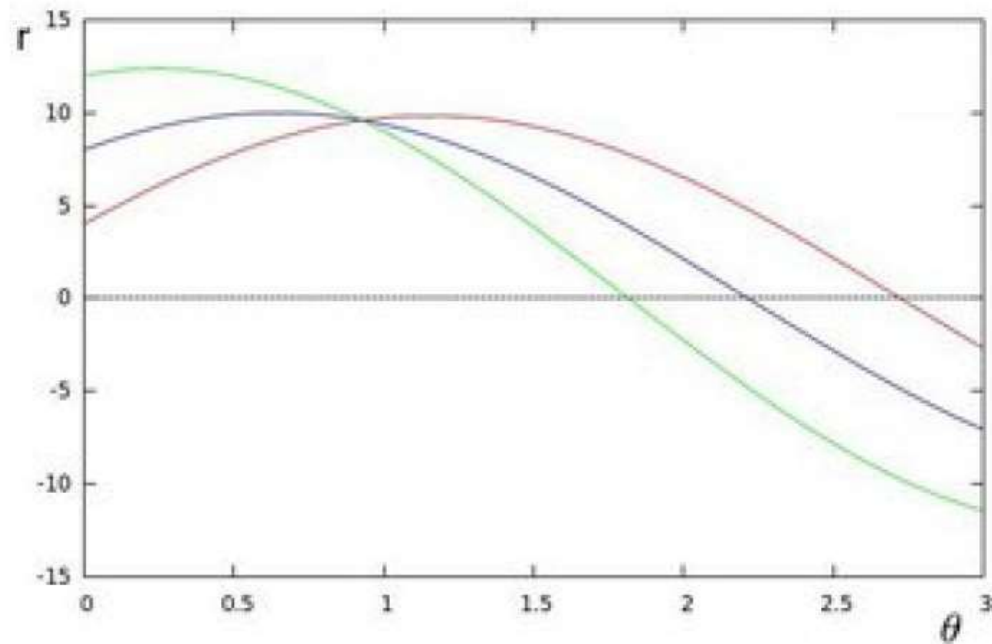
← $(x,y)=(8,6)$ 時通過此點
所有線段畫出來的圖



We consider only points such that $r > 0$ and $0 < \theta < 2\pi$.

3.1 Hough Line Transform Theory

3. We can do the same operation above for all the points in an image. If the curves of two different points intersect in the plane $\theta - r$, that means that both points belong to a same line. For instance, following with the example above and drawing the plot for two more points: $x_1 = 9, y_1 = 4$ and $x_2 = 12, y_2 = 3$, we get:



The three plots intersect in one single point $(0.925, 9.6)$, these coordinates are the parameters (θ, r) or the line in which (x_0, y_0) , (x_1, y_1) and (x_2, y_2) lay.

3.2 Near Field Lane Detection – Hough Transform??, jj2 (1/3)

3.2.1) Gradient weighted Hough transform for Line/Lane Detection/Equation

(1) Hough transform

➤ 用於二值化影像的形狀偵測

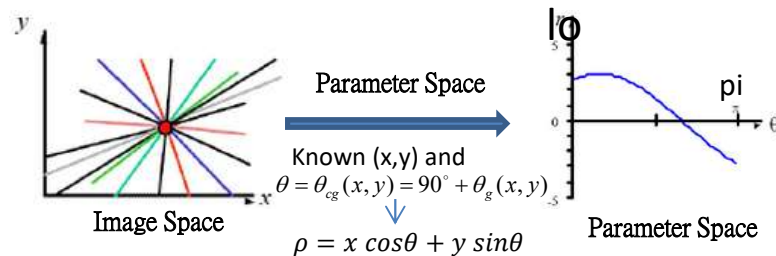
➤ 原理：

1. 找出影像中分散的點特定形狀(例如直線或圓)的參數值
2. 每一點藉由影像空間映射到參數空間產生所有參數的可能值
3. 累計全部點產生的參數值，最後得以在參數空間決定表現最明顯的形狀參數

➤ 舉例：直線 Hough transform

1. 採用極座標 (ρ, θ) ，極座標與直角坐標轉換公式為 $\rho = x \cos \theta + y \sin \theta$
2. ?? 影像空間每一點映射到 (ρ, θ) 參數空間為一條曲線??

??
Circle $x = \rho \cos \theta$
 $y = \rho \sin \theta$

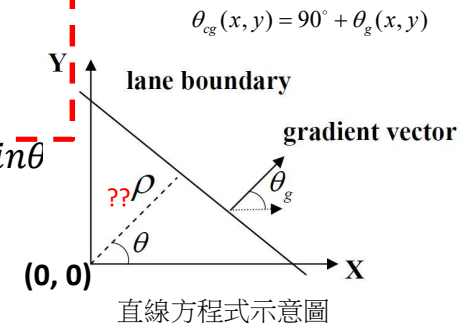


Known (x,y) , try to obtain: Unknown (a,b) ??

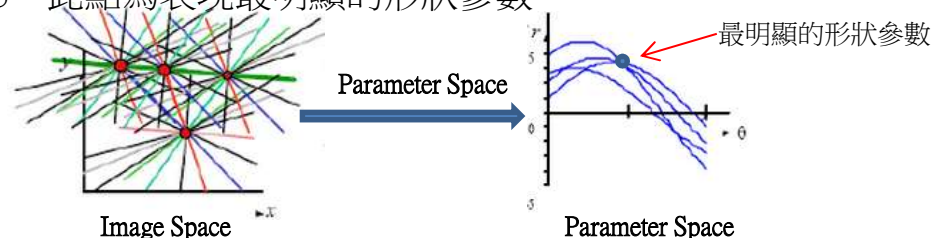
$$f_n(x) = y = a + bx$$

For each pixel (x,y) gradient direction $+ 90^\circ$: known

$$\theta = \theta_{cg} \quad ??$$

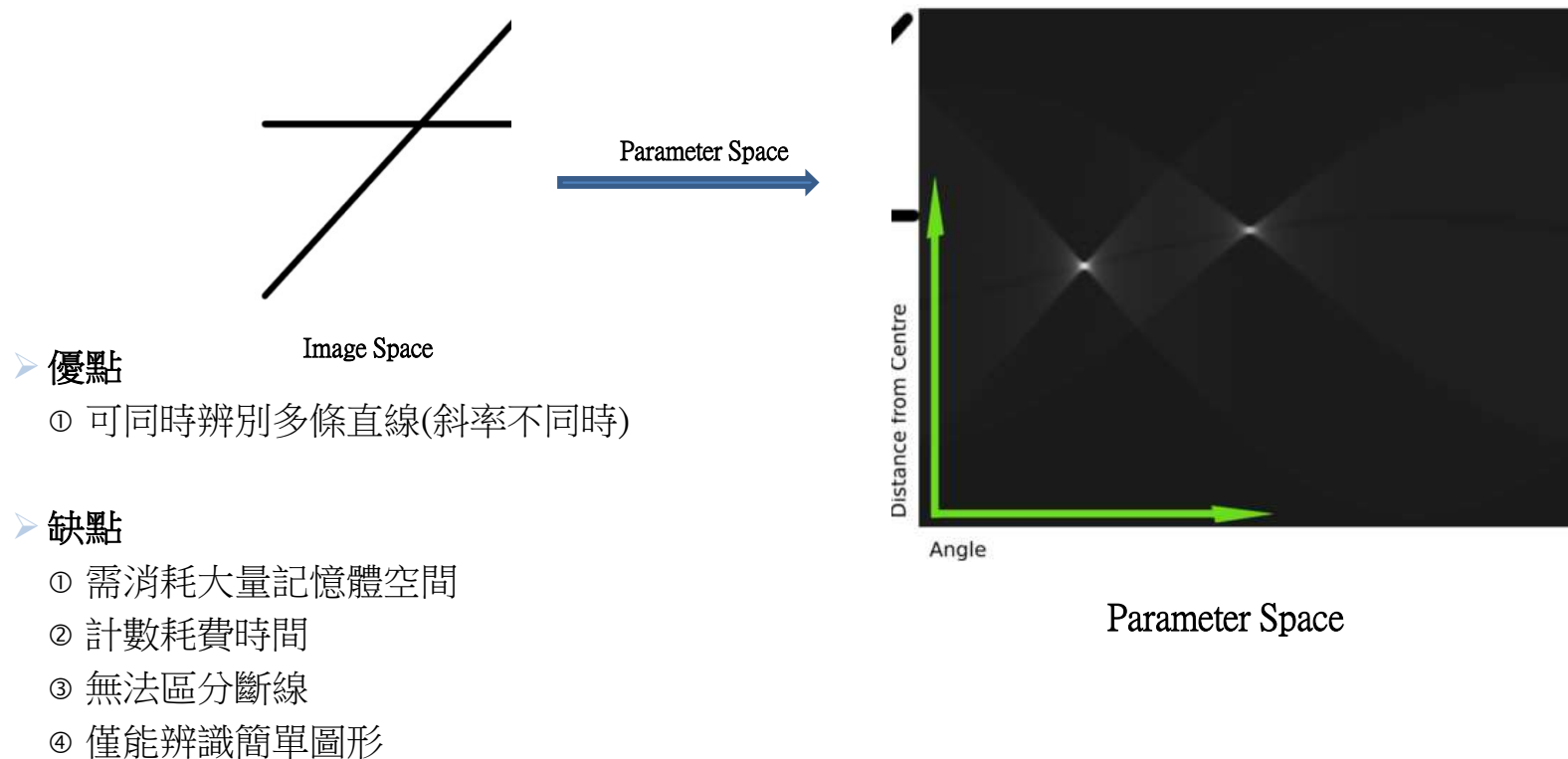


3. 若影像空間同一直線上的三個點，則在 (ρ, θ) 參數空間的三條曲線將交於一點，這一點的累加值為3，此點為表現最明顯的形狀參數



3.2 Near Field Lane Detection – Hough Transform (2/3)

4. 因此當我們的input image擁有的直線越明顯，在參數空間影像中越亮的地方曲線交集越多



3.2 Near Field Lane Detection – Hough Transform ?? (3/3)

(2) ?? Cumulative Gradient Weights:

$$\hat{C}(\rho_k) = \sum_{x \cos \theta_{cg} + y \sin \theta_{cg} = \rho_k} g(x, y)$$

k	1	2	3
θ_g	90°	90°	90°
ρ_k	5	6	7 = ρ_α
$\hat{C}(\rho_k)$	45	8	131

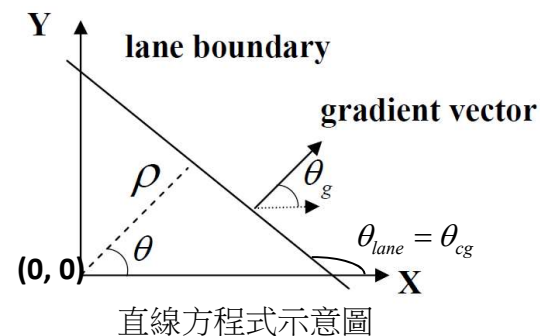
(3) Near Field Lane Equation (近端車道直線方程式)

$$y = \frac{\rho_\alpha}{\sin \theta_{cg}} + \left[-\cot \theta_{cg} \right] x \quad \xrightarrow{\tan \theta_g = \frac{1}{m} = b, \text{ where } m = \text{slope}} \begin{cases} f_n^{left}(x) = a^l + b^l x \\ f_n^{right}(x) = a^r + b^r x \end{cases}$$

ρ_α 使梯度權重函數值最大

a^l, b^l : Left lane's unknown linear function parameters

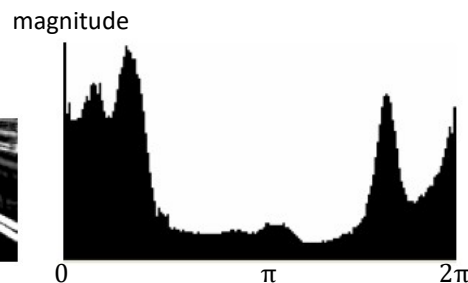
a^r, b^r : Right lane's unknown linear function parameters



原圖



邊緣影像



Cumulative gradient magnitude



近端車道偵測結果



符合車道角度的邊緣影像

3.3 Near Field Lane Detection – Using OpenCV (1/3) ??

◆ 所使用到OpenCV的Function：

1. void cvtColor(InputArray src, OutputArray dst, int code??)

- Input : Array (color image)
- Output : Array (gray image)
- 將RGB轉換為Gray value

RGB to Gray: $Y \leftarrow 0.299 * R + 0.587 * G + 0.114 * B$

2. void Sobel(InputArray src, OutputArray dst, int ddepth??, int dx??, int dy??)

- Input : Array (gray image)
- Output : Array (sobel)
- 計算Image的X方向、Y方向的灰階值差異量

-1	0	+1
-2	0	+2
-1	0	+1

G_x
分量

+1	+2	+1
0	0	0
-1	-2	-1

G_y
分量

3.3 Near Field Lane Detection – Using OpenCV (2/3)??

3. ??void cartToPolar(InputArray x, InputArray y, OutputArray magnitude, OutputArray angle)

- Input : Array*2 (sobel.x , sobel.y)
- Output : Array*2 (magnitude, angle)
- 利用2.Sobel出的X方向、Y方向的灰階值差異量計算梯度強度和方向

$$\begin{aligned} \text{magnitude}(I) &= \sqrt{x(I)^2 + y(I)^2}, \\ \text{angle}(I) &= \text{atan2}(y(I), x(I)) \cdot [180/\pi] \end{aligned}$$

4. ??void HoughLinesP(InputArray image, OutputArray lines, int threshold,
double minLineLength=0, double maxLineGap=0)

- Input : Array(binary image)
- Output : Array(two points of straight-line head and tail)
- 將一張二值化的image做Hough transform，得到直線兩端點的座標
- minLineLength : Minimum line length. Line segments shorter than that are rejected.
- maxLineGap : Maximum allowed gap between points on the same line to link them.

3.3 Near Field Lane Detection – Using OpenCV (3/3)??

5. `void Mat::convertTo(OutputArray m, int rtype)`

- Output : Array
- 將一個array轉換至其他data type
- rtype : Data type
 - CV_8U : 8-bit unsigned integer
 - CV_8S : 8-bit signed integer
 - CV_16U : 16-bit unsigned integer
 - CV_16S : 16-bit signed integer
 - CV_32S : 32-bit signed integer
 - CV_32F : 32-bit floating-point number
 - CV_64F : 64-bit floating-point number

3.4 Line Hough Function in OpenCV

HoughLinesP

Finds line segments in a binary image using the probabilistic Hough transform.

C++: `void HoughLinesP(InputArray image, OutputArray lines, double rho, double theta, int threshold, double minLineLength=0, double maxLineGap=0)`

Python: `cv2.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]]) → lines`

Parameters

image – 8-bit, single-channel binary source image. The image may be modified by the function.

lines – Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2) , where (x_1, y_1) and (x_2, y_2) are the ending points of each detected line segment.

rho – Distance resolution of the accumulator in pixels.

theta – Angle resolution of the accumulator in radians.

threshold – Accumulator threshold parameter. Only those lines are returned that get enough votes ($> \text{threshold}$).

minLineLength – Minimum line length. Line segments shorter than that are rejected.

maxLineGap – Maximum allowed gap between points on the same line to link them.

Input : Array(binary image)

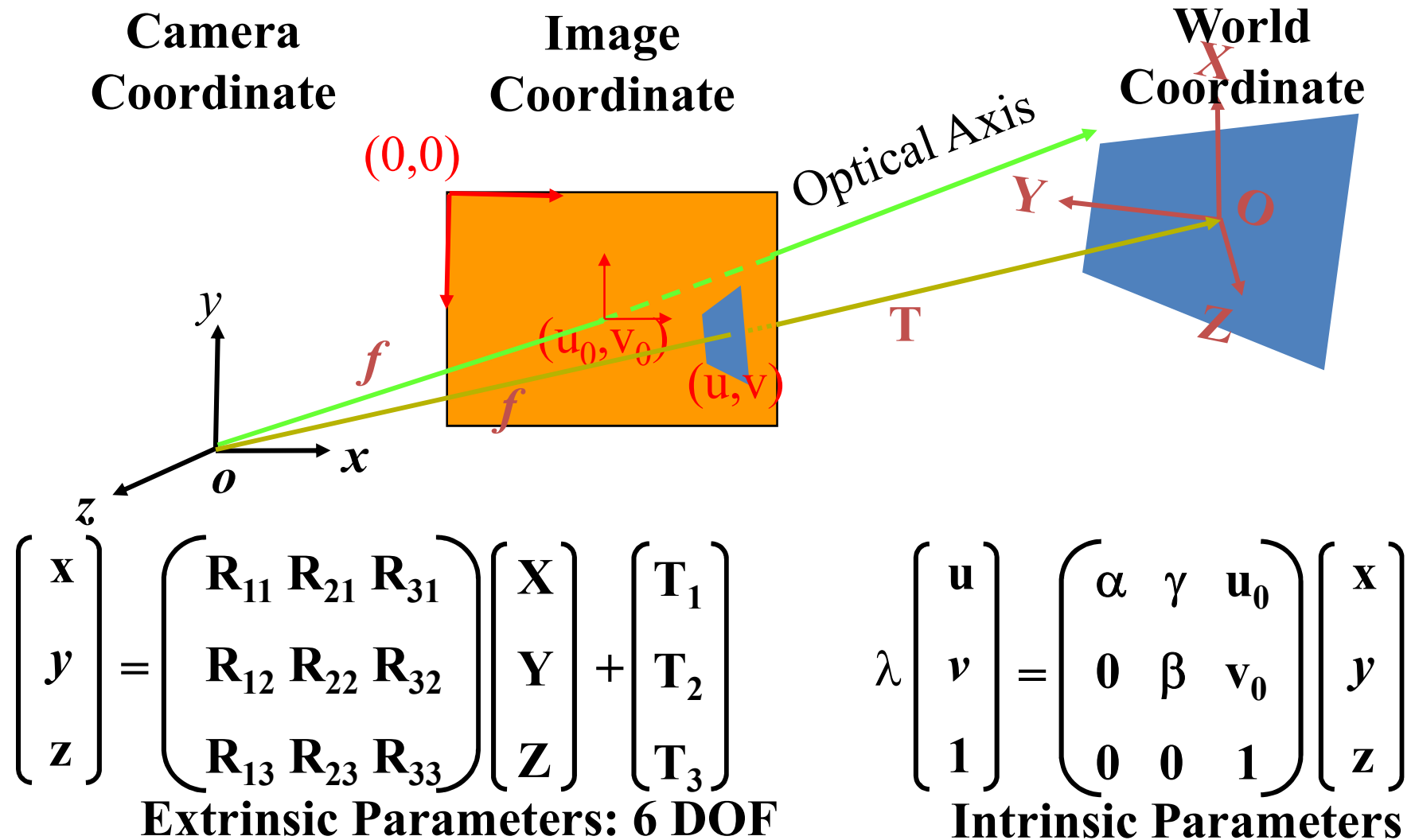
Output : Array(two points of straight-line head and tail)

將一張二值化的image做Hough transform，得到直線兩端點的座標

minLineLength：最小線段長度

maxLineGap：最大兩點差距可連起同一條線

World to Camera to Image Coordinates (1/3)



Camera Parameters: Intrinsic + Extrinsic Parameters (2/3)

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{21} & R_{31} & T_1 \\ R_{12} & R_{22} & R_{32} & T_2 \\ R_{13} & R_{23} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Scale Factor: λ

Normalize to unit vector

$$u = -f k_u (x/z) + \gamma (y/z) + u_0$$

$$v = -f k_v (y/z) + v_0$$

● Intrinsic Parameters:

- Scale Factor: $\alpha = -f k_u$, f : focal length
- Scale Factor: $\beta = -f k_v$
- Aspect Ratio = $\alpha/\beta = k_u/k_v$
- Skew Factor: γ
- Principal Point: (u_0, v_0)

● Extrinsic Parameters:

- Rotation: R
- Translation: T

Pinhole Camera Model: World \Leftrightarrow Camera \Leftrightarrow Image Coordinates

Framework: distortion vs undistortion??

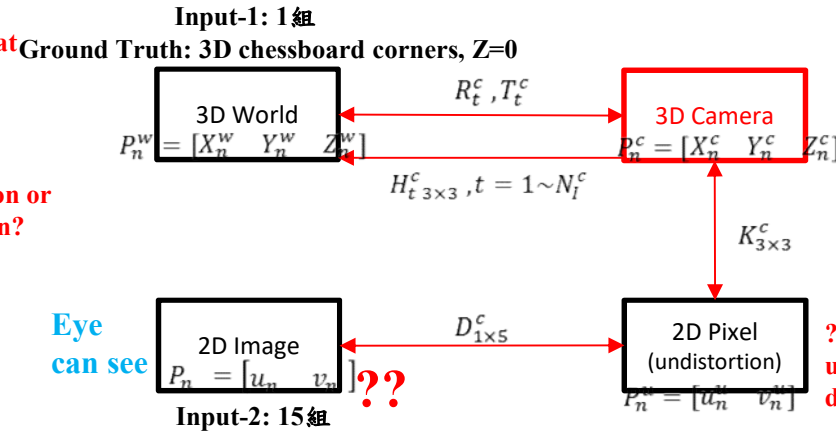
Chessboard: 棋盤格影像
Chessboard pattern: 投影棋盤格

3D camera: 1 chessbd images
3D DLP: 15 images
15 images
1 pattern at

$$\min \sum_{n=1}^N (p_n^u - p_n^u)^2$$

$$\begin{bmatrix} u_n \\ v_n \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{41} & h_{42} & h_{43} & 1 \end{bmatrix} \begin{bmatrix} X_n^w \\ Y_n^w \\ Z_n^w \\ 1 \end{bmatrix}$$

$Z_n^w = 0, N_{rc} = 48$
 $P_n^w = [X_n^w \ Y_n^w \ Z_n^w]^T$
 $P_n^c = [X_n^c \ Y_n^c \ Z_n^c]^T$
 $P_n^u = [u_n \ v_n]^T$
 $H_{t,3 \times 3}$
reprojection
Distortion or undistortion??



1) From 2D image (distorted image) coordinate (u_n, v_n) to 2D pixel (undistorted) coordinate $P_n^u = [u_n^u \ v_n^u]^T$

(1) Radial (barrel) distortion:

$$u_n^u = u_n (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$v_n^u = v_n (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

(2) Tangential distortion:

$$u_n^u = u_n + [2p_1 v_n + p_2 (r^2 + 2u_n^2)]$$

$$v_n^u = v_n + [p_1 (r^2 + 2v_n^2) + 2p_2 u_n]$$

where $r^2 = u_n^2 + v_n^2$

2) Perspective Projection: From 3D camera coordinate projects to 2D image coordinate

?? If this is correct then D doesnot calculate included??

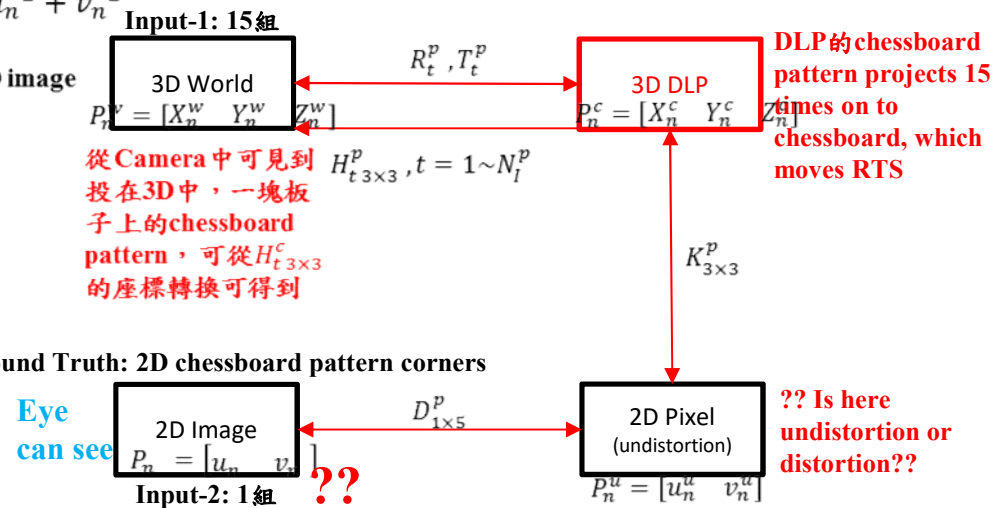
$$\begin{bmatrix} u_n \\ v_n \end{bmatrix} = \begin{bmatrix} f_x & \gamma = 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_n^c \\ Y_n^c \\ Z_n^c \\ 1 \end{bmatrix}$$

3) Affine: From 3D world coordinate to 3D camera coordinate

$$s \begin{bmatrix} X_n^c \\ Y_n^c \\ Z_n^c = 0 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_n^{w'} \\ Y_n^{w'} \\ Z_n^{w'} = 0 \\ 1 \end{bmatrix}$$

4) From 3D world coordinate to 2D coordinate

$$s \begin{bmatrix} u_n \\ v_n \\ l \end{bmatrix} = \begin{bmatrix} f_x & \gamma = 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_n^{w'} \\ Y_n^{w'} \\ Z_n^{w'} = 0 \\ 1 \end{bmatrix}$$



Camera Parameters:

Intrinsic + Extrinsic Parameters (3/3)

- Intrinsic parameters: Projection matrix
 - Is calculated via **camera calibration**
 - Most important intrinsic parameter is **focal length f** .
 - **3D-to-2D projection**: From 3D world/camera coordinate projects to 2D image coordinate.
 - **2D-to-3D back-projection/reconstruction**: From 2D image coordinate back-projects to 3D world coordinate.
- Extrinsic parameters: Affine transformation
 - Geometric transformation
 - **3D** world/camera coordinate Vs. **3D** world/camera coordinate
 - **2D** image coordinate Vs. **2D** image coordinate

Rotation Matrix Representation: Euler angles

Assume rotation matrix R results from successive Euler rotations of the camera frame around its X axis by ω , its once rotated Y axis by ϕ , and its twice rotated Z axis by κ , then

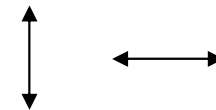
$$R(\omega, \phi, \kappa) = R_X(\omega)R_Y(\phi)R_Z(\kappa)$$

Different
orders have
different results

where ω , ϕ , and κ are often referred to as tilt, pan, and swing angles respectively.

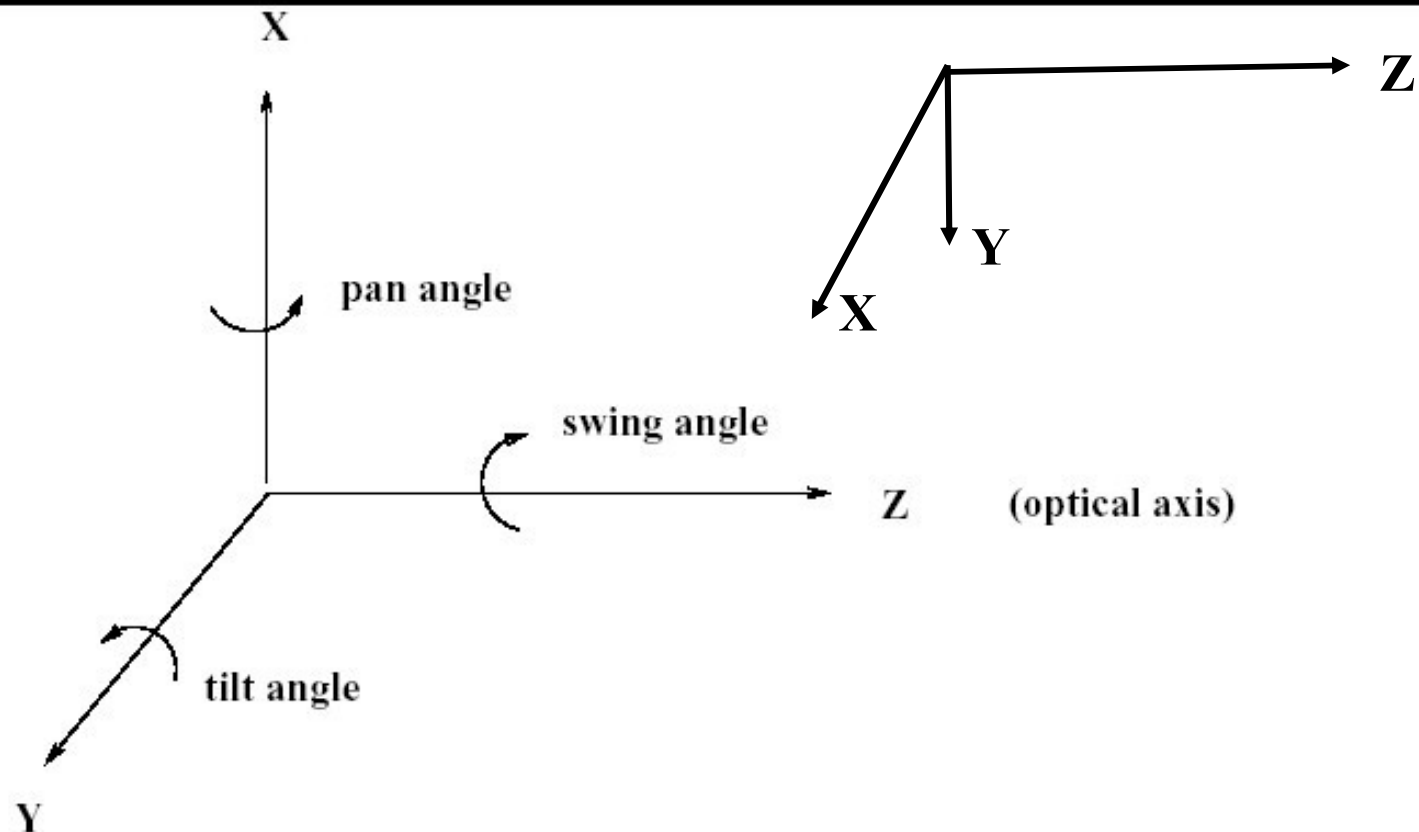


roll



pitch yaw

Rotation Matrix Representation: Euler angles



- **In-Plane Rotation/Motion:** **Roll (or Swing)**
- **Out-of-Plane Rotation Motion:** **Pitch (or Tilt) and Yaw (or Pan)**

□DIP book

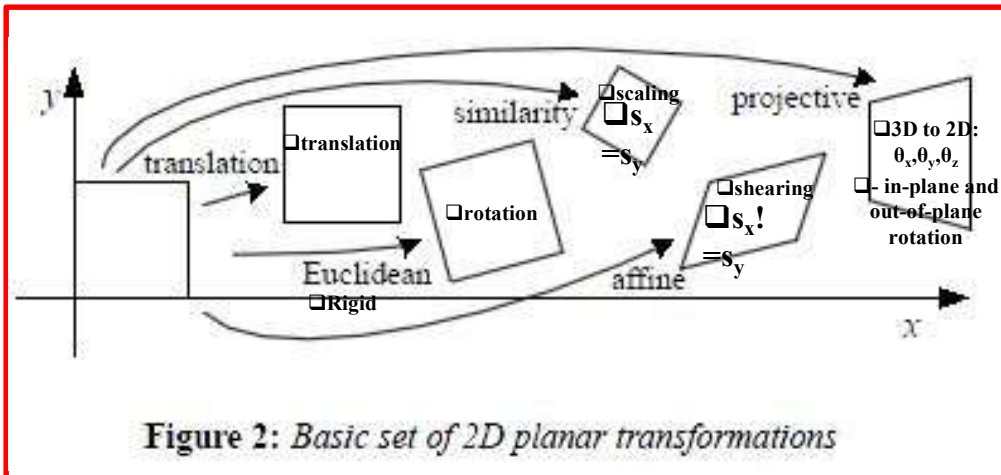
$$R_x(\omega) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \omega & \sin \omega \\ 0 & -\sin \omega & \cos \omega \end{pmatrix}$$

$$R_y(\phi) = \begin{pmatrix} \cos \phi & 0 & -\sin \phi \\ 0 & 1 & 0 \\ \sin \phi & 0 & \cos \phi \end{pmatrix}$$

$$R_z(\kappa) = \begin{pmatrix} \cos \kappa & \sin \kappa & 0 \\ -\sin \kappa & \cos \kappa & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2D (Planar) Motions (Transformation) (1/6) jj

◆ 2D Planar Motion Transformations:



I : Identity matrix (2x2).

$\tilde{x} (x, y, 1)$: Homogeneous or projective 2D coordinate

$\tilde{x}' (x', y')$: Transformed coordinate

$t (t_x, t_y)$: Translation.

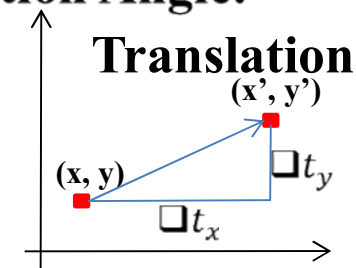
R : Rotation Matrix.

θ : Rotation Angle.

1) Translation Transformation: Translation

$$\tilde{x}' = \tilde{x} + t \quad \text{or} \quad \tilde{x}'_{2 \times 1} = [I_{2 \times 2} \ t_{2 \times 1}] \tilde{x}_{3 \times 1}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

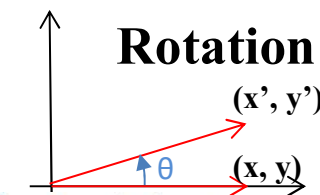


2) Euclidean Transformation (Rigid, 2D rigid body): Rotation + Translation

$$\tilde{x}' = R\tilde{x} + t \quad \text{or} \quad \tilde{x}' = [R \ t] \tilde{x}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

is an orthonormal rotation matrix with $RR^T = I$ and $|R| = 1$.



$$a = \cos \theta$$

$$b = \sin \theta$$

$$a^2 + b^2 = 1$$

2D (Planar) Motions (Transformation) (2/6) jj

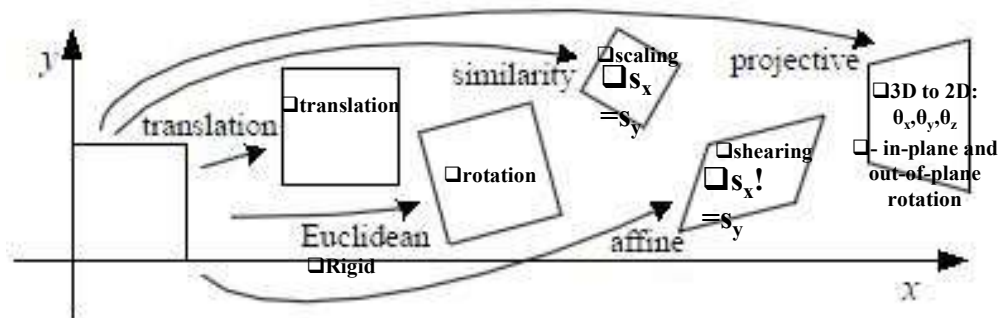


Figure 2: Basic set of 2D planar transformations

$\tilde{x} (x, y, 1)$: Homogeneous coordinates

$\tilde{x}' (x', y')$: Transformed coordinates.

$t (t_x, t_y)$: Translation.

R : Rotation matrix.

s : Scale factor.

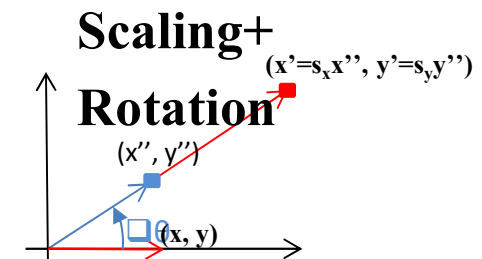
θ : Rotation Angle.

3) **Similarity Transformation** (Scaled rotation): Scaling ($s_x=s_y$) + Rotation + Translation

$$\tilde{x}' = sR\tilde{x} + t \quad \text{or} \quad \tilde{x}' = [sR \quad t]\tilde{x}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad sR = \begin{bmatrix} s \cos \theta & -s \sin \theta \\ s \sin \theta & s \cos \theta \end{bmatrix}$$

where we no longer require that $a^2 + b^2 = 1$.



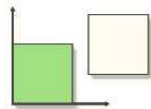
2D (Planar) Motions (Transformation) (3/6) jj

4) Affine Transformation: Scaling ($s_x \neq s_y$) + Shearing + Rotation + Translation any transformation that preserves **parallel lines**

$$\tilde{x}' = A\tilde{x} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

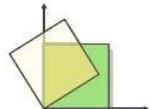
Translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



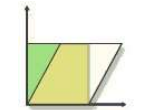
Rotation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Shear:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k_x \\ k_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Scaling:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$\begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$h_{20} = 0 \quad h_{21} = 0 \quad h_{22} = 1$

Linear Combination:

$$x' = \frac{x_1}{w_1} = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} = \frac{h'_{00}x + h'_{01}y + h'_{02}}{h'_{20}x + h'_{21}y + 1}$$

$$y' = \frac{y_1}{w_1} = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} = \frac{h'_{10}x + h'_{11}y + h'_{12}}{h'_{20}x + h'_{21}y + 1}$$

1. Adding Translation ($\tilde{x}' = \tilde{x} + t$):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

2. Adding Rotation ($\tilde{x}' = R \cdot \tilde{x} + t$):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

3.1. Adding Scaling ($\tilde{x}' = S \cdot R \cdot \tilde{x} + t$):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x \cos \theta & -S_x \sin \theta \\ S_y \sin \theta & S_y \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

3.2. Adding Shear ($\tilde{x}' = S \cdot R \cdot \tilde{x} + t$):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k_x \\ k_y & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta + k_x \sin \theta & -\sin \theta + k_x \cos \theta \\ k_y \cos \theta + \sin \theta & -k_y \sin \theta + \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

4. All ($\tilde{x}' = S \cdot S \cdot R \cdot \tilde{x} + t$):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} 1 & k_x \\ k_y & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x(\cos \theta + k_x \sin \theta) & S_x(-\sin \theta + k_x \cos \theta) \\ S_y(k_y \cos \theta + \sin \theta) & S_y(-k_y \sin \theta + \cos \theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Matrix form(2x3):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x(\cos \theta + k_x \sin \theta) & S_x(-\sin \theta + k_x \cos \theta) & t_x \\ S_y(k_y \cos \theta + \sin \theta) & S_y(-k_y \sin \theta + \cos \theta) & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$\tilde{x} (x, y, 1)$: Homogeneous coordinates.

$\tilde{x}' (x', y')$: Transformed coordinates.

$t (t_x, t_y)$: Translation.

R : Rotation matrix.

$$\tilde{x}' = A\tilde{x}$$

S_x : Scale factor to the x-axis.

S_y : Scale factor to the y-axis.

A : Affine Matrix.

θ : Rotation Angle.

k_x : Shear factor parallel to the x-axis.

k_y : Shear factor parallel to the y-axis.

H : Homogeneous matrix.

$$\begin{bmatrix} 1 & 1 \\ ky/kx & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \gamma & 1 \end{bmatrix}$$

?? Intrinsic skew factor??

Scale factor =? Aspect Ratio = $\alpha/\beta = k_u/k_v$

2D (Planar) Motions (Transformation) (4/6) jj

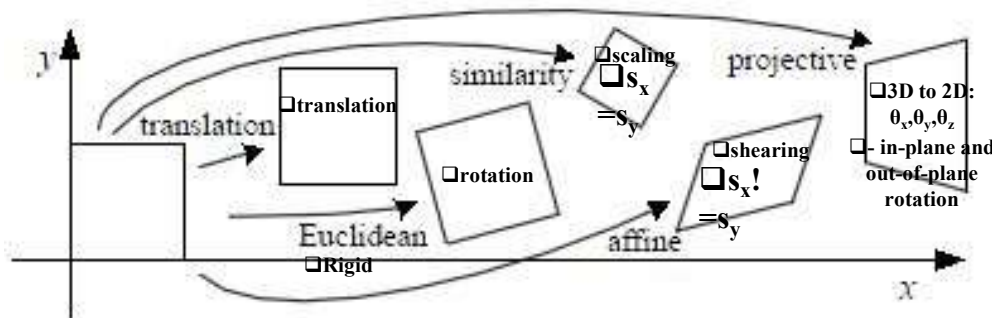


Figure 2: Basic set of 2D planar transformations

$\tilde{x} (x, y, 1)$: Homogeneous coordinates.

$\tilde{x}' (x', y')$: Transformed coordinates.

H : Homogeneous matrix.

Rotation:

In-Plane rotation + Out-of-plane rotation: $\theta_x, \theta_y, \theta_z$

Including 3x3 intrinsic (or projection) parameters + affine transform with in-plane and out-of-plane rotations

5) Projective Transformation (Perspective Transform): 3D project to 2D

Projective transformations preserve **straight lines**.

$$\tilde{x}' \approx \tilde{H} \tilde{x}$$

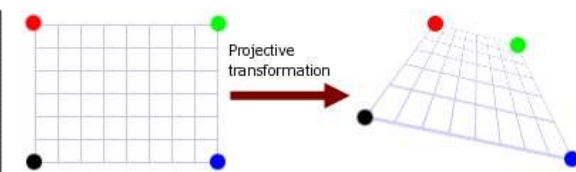
$$\begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \tilde{H} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

$$x' = \frac{x_1}{w_1} = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} = \frac{h'_{00}x + h'_{01}y + h'_{02}}{h'_{20}x + h'_{21}y + 1}$$

$$y' = \frac{y_1}{w_1} = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} = \frac{h'_{10}x + h'_{11}y + h'_{12}}{h'_{20}x + h'_{21}y + 1}$$

where $h'_{ij} = \frac{h_{ij}}{h_{22}}$

Non-Linear combination



2D (Planar) Motions (Transformation) (5/6-1/11)

6) 8-Parameter planar transformation Vs. perspective projection transformation

Motion	$u(x, y) = a_0 + a_1x + a_2y + p_0x^2 + p_1xy$	divergence = $a_1 + a_5 = (u_x + v_y)$,	(3)
flow	$v(x, y) = a_3 + a_4x + a_5y + p_0xy + p_1y^2$	curl = $-a_2 + a_4 = -(u_y - v_x)$,	(4)
Motion	$u(x, y) = a_0 + a_1x + a_2y$	deformation = $a_1 - a_5 = (u_x - v_y)$	(5)
flow	$v(x, y) = a_3 + a_4x + a_5y + cx^2$	Yaw = p_0 ,	
		Pitch = p_1 .	

a_i : Constants.

$u(x) = [u(x,y), v(x,y)]^T$: Horizontal and vertical components of **the flow** at the image point $x = (x,y)$.

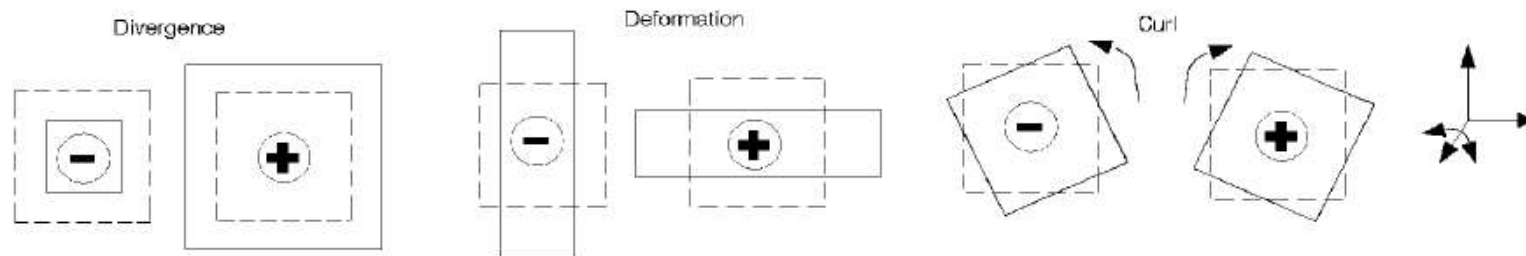


Figure 3. The figure illustrates the motion captured by the various parameters used to represent the motion of the regions. The solid lines indicate the deformed image region and the “-” and “+” indicate the sign of the quantity.

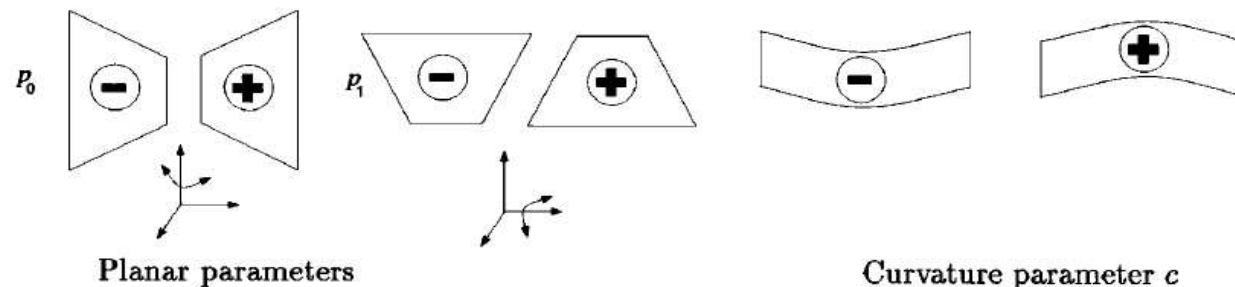


Figure 4. Additional parameters for planar motion and curvature.

2D (Planar) Motions (Transformation) (5/6-2/11)

$$\mathbf{X}(\mathbf{x}) = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy & 0 \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 & x^2 \end{bmatrix} \quad (10)$$

$$\mathbf{A} = [a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ 0 \ 0 \ 0]^T \quad (11)$$

$$\mathbf{P} = [a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ p_0 \ p_1 \ 0]^T \quad (12)$$

$$\mathbf{C} = [a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ 0 \ 0 \ c]^T \quad (13)$$

such that $\mathbf{u}(\mathbf{x}; \mathbf{A}) = \mathbf{X}(\mathbf{x})\mathbf{A}$, $\mathbf{u}(\mathbf{x}; \mathbf{P}) = \mathbf{X}(\mathbf{x})\mathbf{P}$, and $\mathbf{u}(\mathbf{x}; \mathbf{C}) = \mathbf{X}(\mathbf{x})\mathbf{C}$ represent, respectively, the affine, planar, and affine + curvature flow models described above.

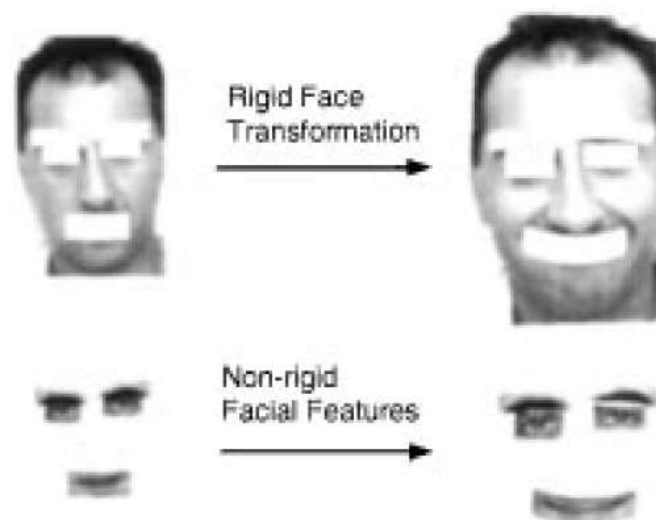
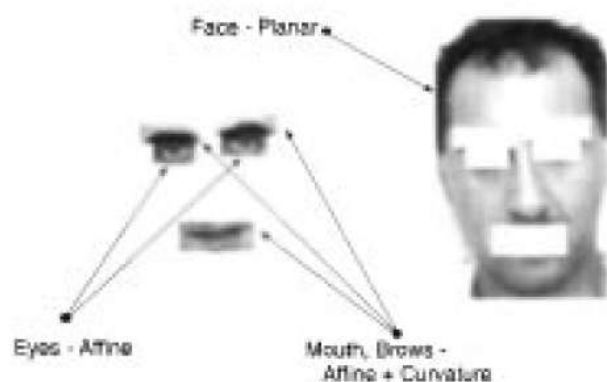


Figure 1. Illustration showing the parametric motion models employed and an example of a face undergoing a looming motion while smiling.

2D (Planar) Motions (Transformation) (6/6) jj






Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I & t \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} R & t \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity $S_x = S_y$	$\begin{bmatrix} sR & t \end{bmatrix}_{2 \times 3}$	4	angles + ...	
affine $S_x \neq S_y$	$\begin{bmatrix} A \end{bmatrix}_{2 \times 3}$	6	parallelism + ...	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3 \times 3}$	8	straight lines	

Table 1: Hierarchy of 2D coordinate transformations. The 2×3 matrices are extended with a third $[0^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

#D.O.F : Degrees Of Freedom

1) Translation : t_x, t_y

2) Euclidean : t_x, t_y, θ

3) Similarity : t_x, t_y, θ, s

4) Affine : $a_{00}, a_{01}, a_{02}, a_{10}, a_{11}, a_{12}$

5) Projective : $h'_{00}, h'_{01}, h'_{02}, h'_{10}, h'_{11}, h'_{12}, h'_{20}, h'_{21}$

Motion flow

6) 8-Parameter planar transformation: $a_0, a_1, a_2, a_3, a_4, a_5, p_0$ and p_1