Image Analysis Exercise 1:

Fourier Descriptors

(Due 13.11.2009)

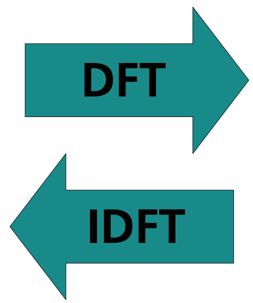
Computer Vision and Remote Sensing

The Discrete Fourier Transform in MATLAB (1)

Time domain
(complex signal)

$$a(x) \;=\; \square\square\square\square\square\square$$

$$a(0),\; a(1),\; \ldots,\; a(N-1)$$



Frequancy domain
(complex spectrum)

$$A(\mu) \;=\; \square\square\square\square\square\square$$

$$A(0),\; A(1),\; \ldots,\; A(N-1)$$

$$A(\mu) \;=\; \sum_{x=0}^{N-1} a(x)\;\exp\left(-\frac{2\pi\imath}{N}\mu x\right)$$

Forward DFT
(MATLAB `fft`)

$$\boxed{a(x)} \;=\; \sum_{\mu=0}^{N-1} \boxed{A(\mu)}\;\boxed{\exp\left(\frac{2\pi\imath}{N}\mu x\right)}$$

Inverse DFT
(MATLAB `ifft`)

Signal

Reconstructed from the spectrum $A(\mu)$.

Spectrum

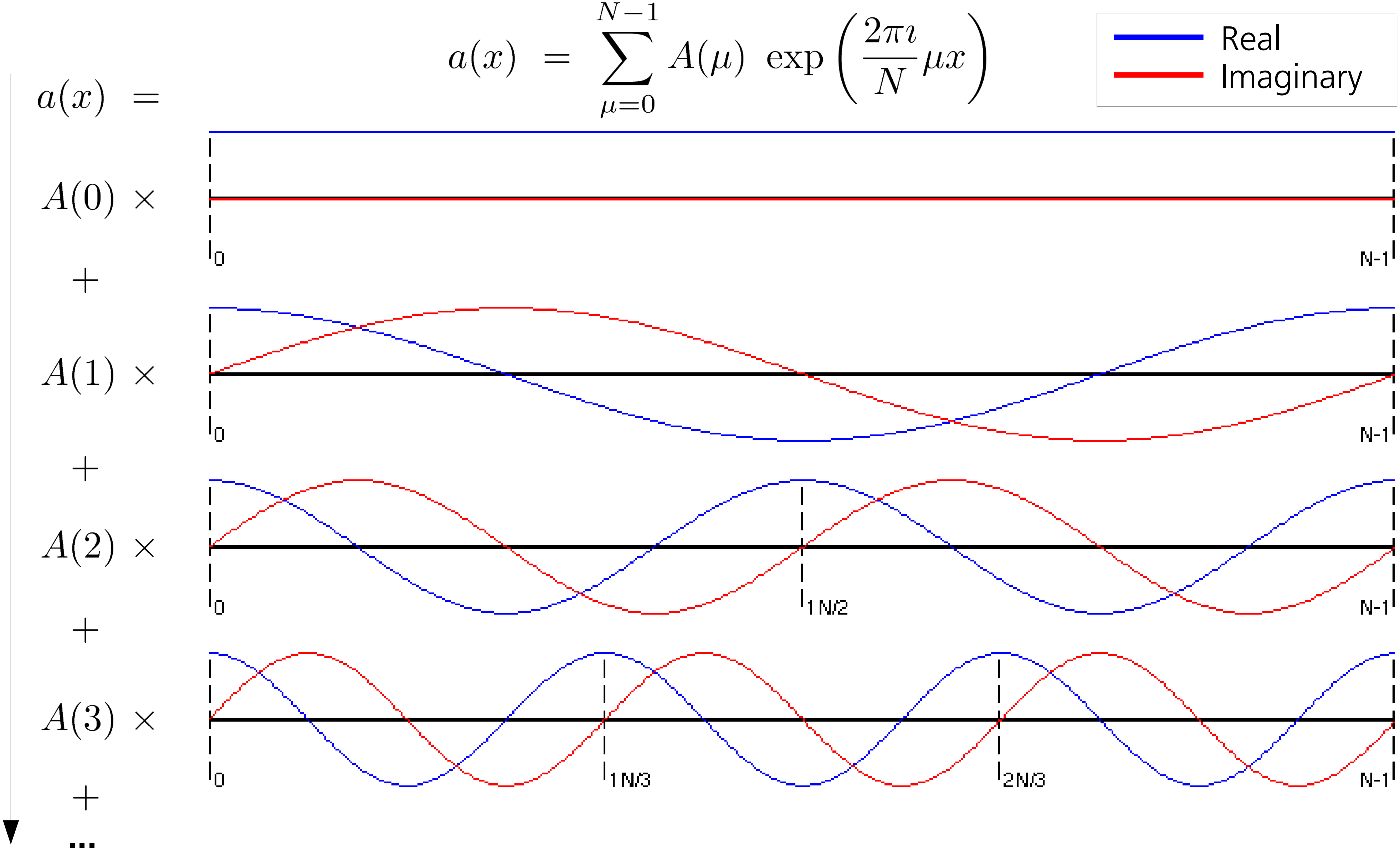
Frequency domain representation.
Weight and phase of frequency μ.

Basis Function

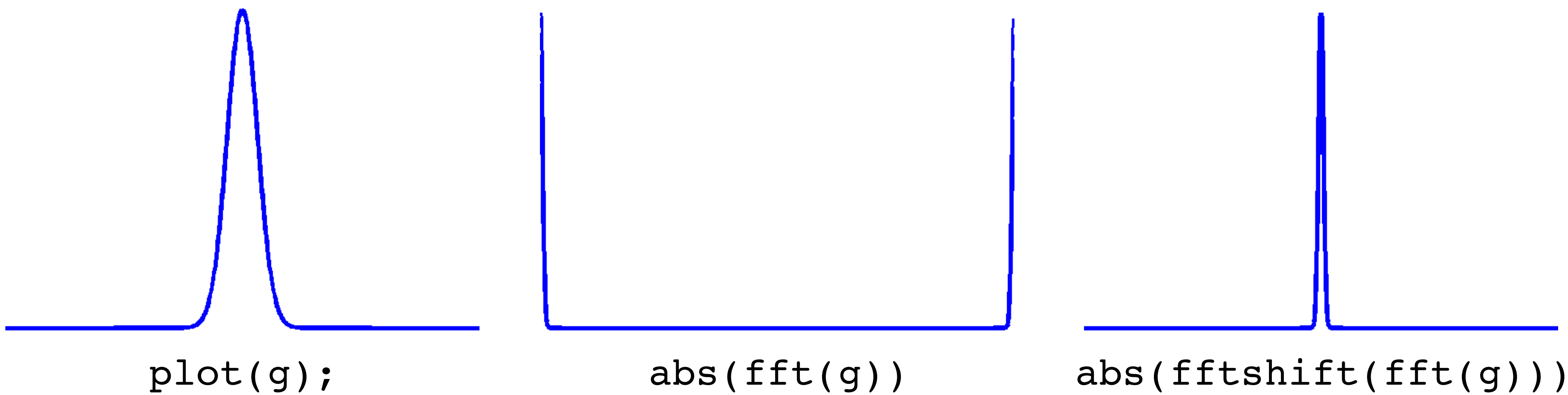
Complex exponential(oscillation).
Frequency μ/N over entire signal $0 \leq x < N$.

The Discrete Fourier Transform in MATLAB (2)

$$a(x) = \sum_{\mu=0}^{N-1} A(\mu)\ \exp\left(\frac{2\pi \imath}{N}\mu x\right)$$

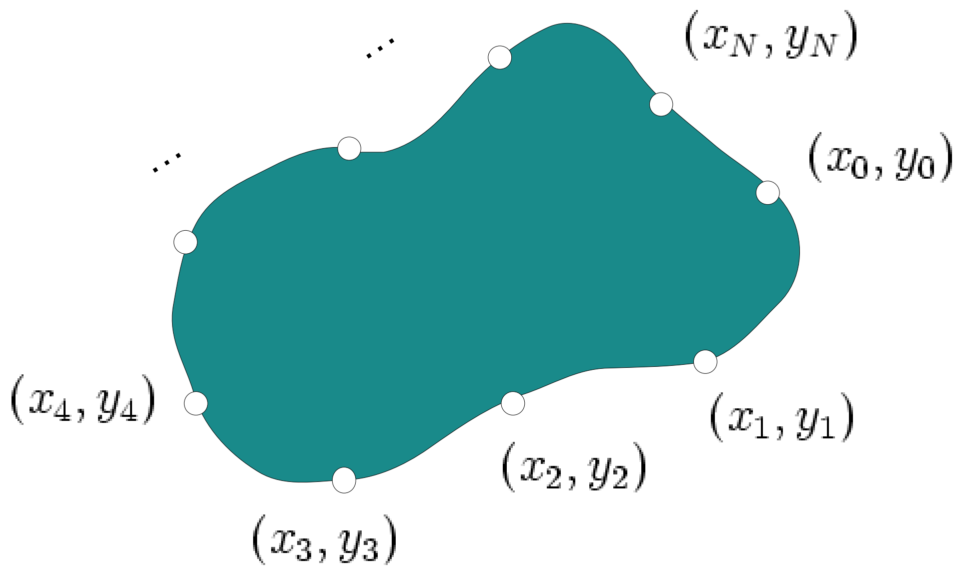


The Discrete Fourier Transform in MATLAB (4)



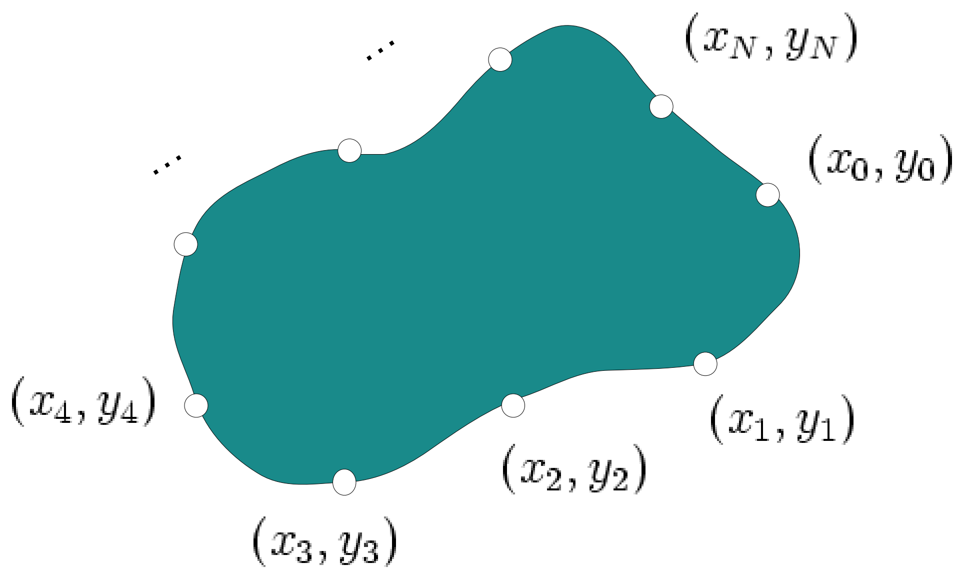
- Forward transform: `fft` (`fft2` for 2D signals)
- Inverse transform: `ifft` (`ifft2` for 2D signals)
- The transforms adhere strictly to the definition of the DFT
- The zero frequency (DC component) is at ***A(0)***
- The MATLAB function `fftshift` centres the spectrum
  - Largest “effective” negative frequency in ***A(0)***
  - Largest “effective” positive frequency in ***A(N-1)***
- `ifftshift` reverses the effects of `fftshift`

Fourier Descriptors: Overview



- Concise and description of (object) contours
  - ➔ Contours are represented by vectors
- Numerous application
  - ➔ Contour Processing (filtering, interpolation, morphing)
  - ➔ Image analysis: Characterising and recognising the shapes of object

Representing a Contour using the DFT



(x_N,y_N): Coordinates of the N^{th} point along the circumference

Pixels on the contour are assumed to be ordered (e.g. clockwise)!

1st Step

Define a complex vector using coordinates (x,y).

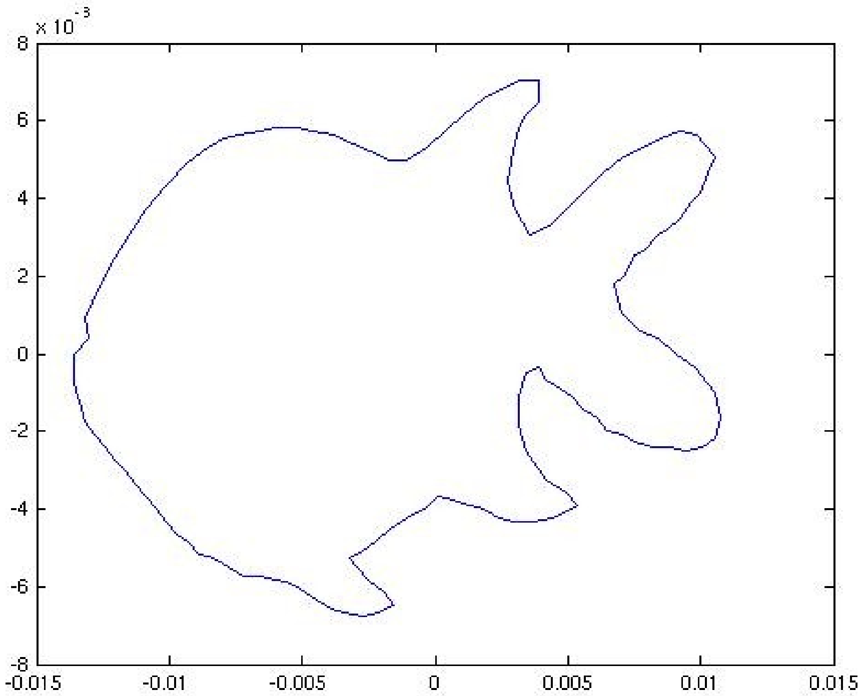
$$\tilde{\mathbf{U}} = \begin{pmatrix} x_0 + \imath y_0 \\ x_1 + \imath y_1 \\ \vdots \\ x_N + \imath y_N \end{pmatrix}$$

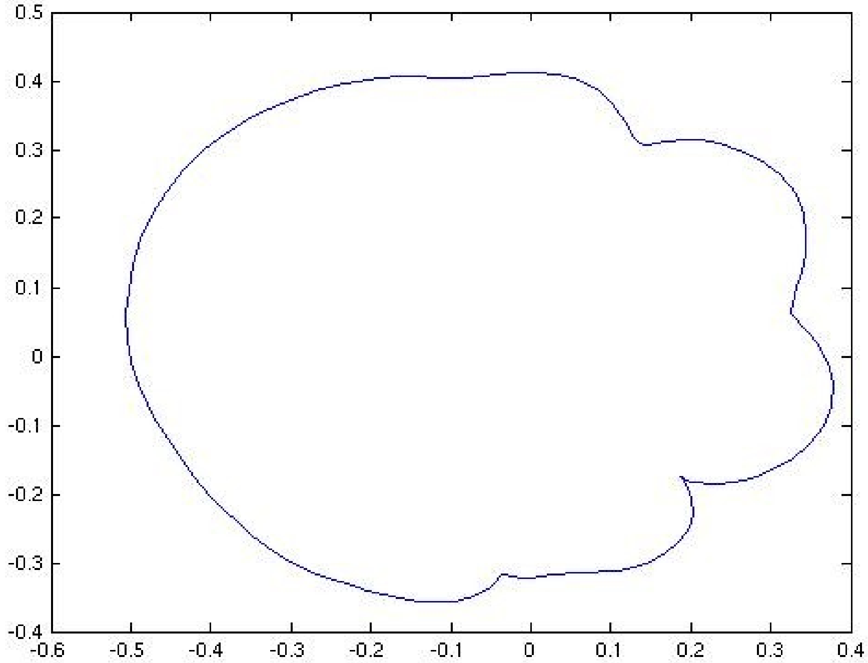
2nd Step

Apply the 1D DFT

$$\tilde{\mathbf{F}}_\mu = FFT[\tilde{\mathbf{U}}] = \sum_{k=0}^{N-1} \tilde{\mathbf{U}}_k \ \exp\left(-\frac{2\pi\imath}{N} k\mu\right)$$

Contour Processing





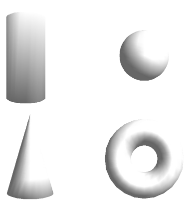
- In this example, a contour is processed using a low-pass filter
  - ➔ Other filters: Sharpening, Edge extraction, ...
- Morphing contours and interpolation are also easily achieved

Fourier Descriptors in Image Analysis

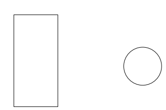
• Object Recognition using shape information

- Appropriate pre-processing steps make Fourier Descriptors invariant to common transformations
  - ➔ Translation, changes in scale, rotation
- The contour of a known object can therefore be recognised irrespectively of its position, size and orientation

1. Database with known object types



2. Extraction of object contours



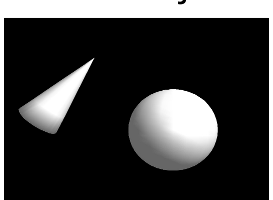
...

3. Computation of invariant Fourier Descriptors

F_μ F_μ

...

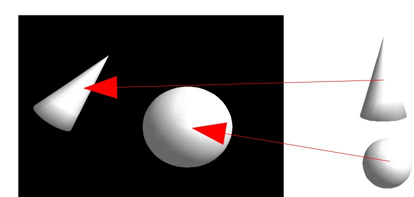
1. Image with unknown objects



2. Extraction of object contours and invariant descriptors

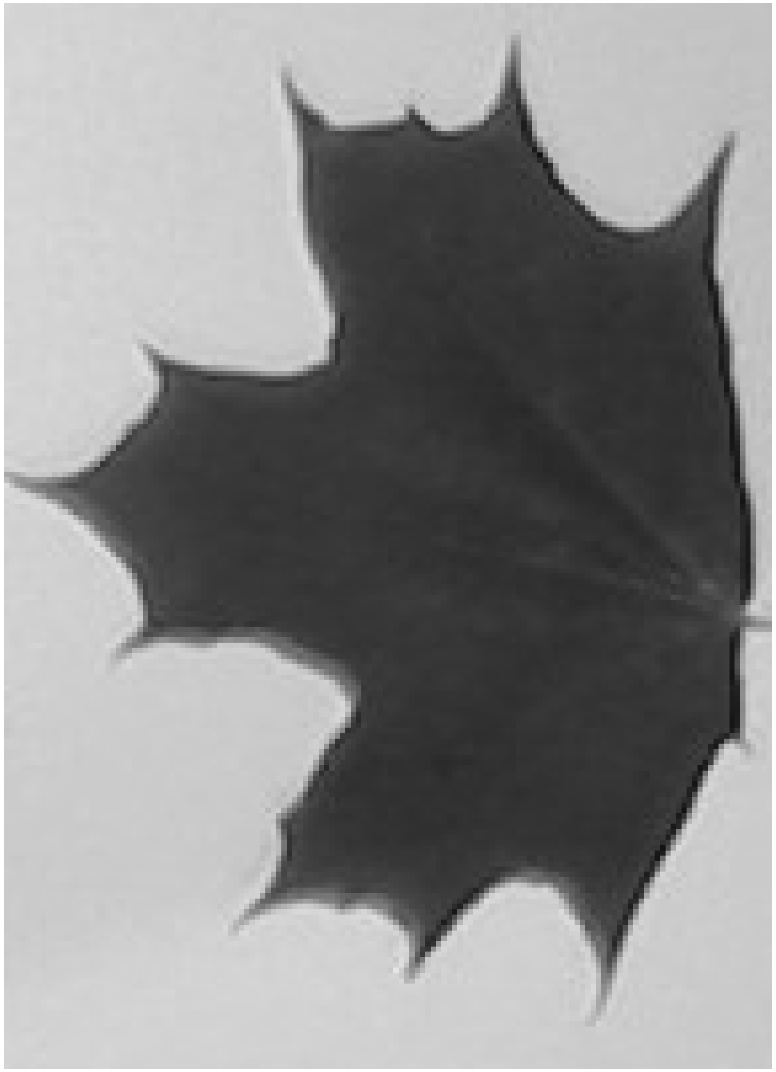
F_μ F_μ

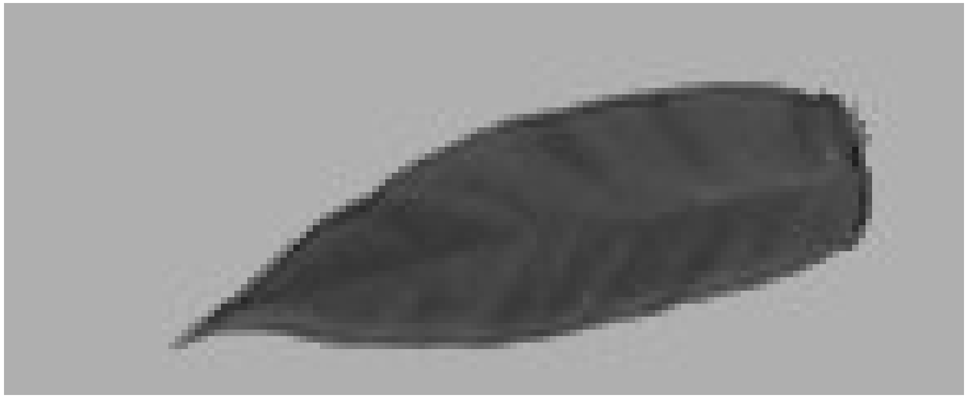
3. Recognition by comparison with database



Application: Recognising and classifying leaves

Database





Two types of leaves are to be recognised and classified

Application: Recognising and classifying leaves

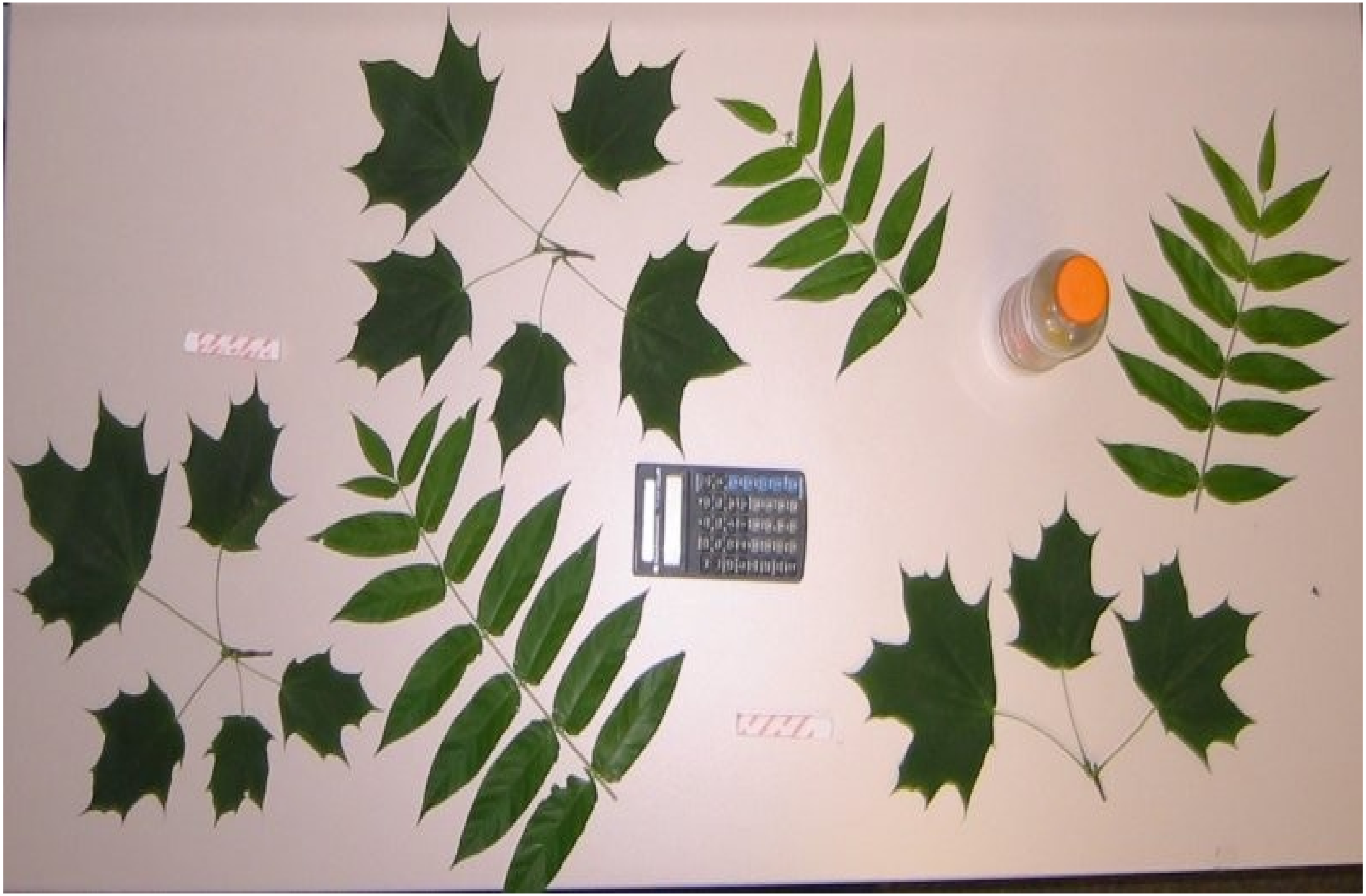
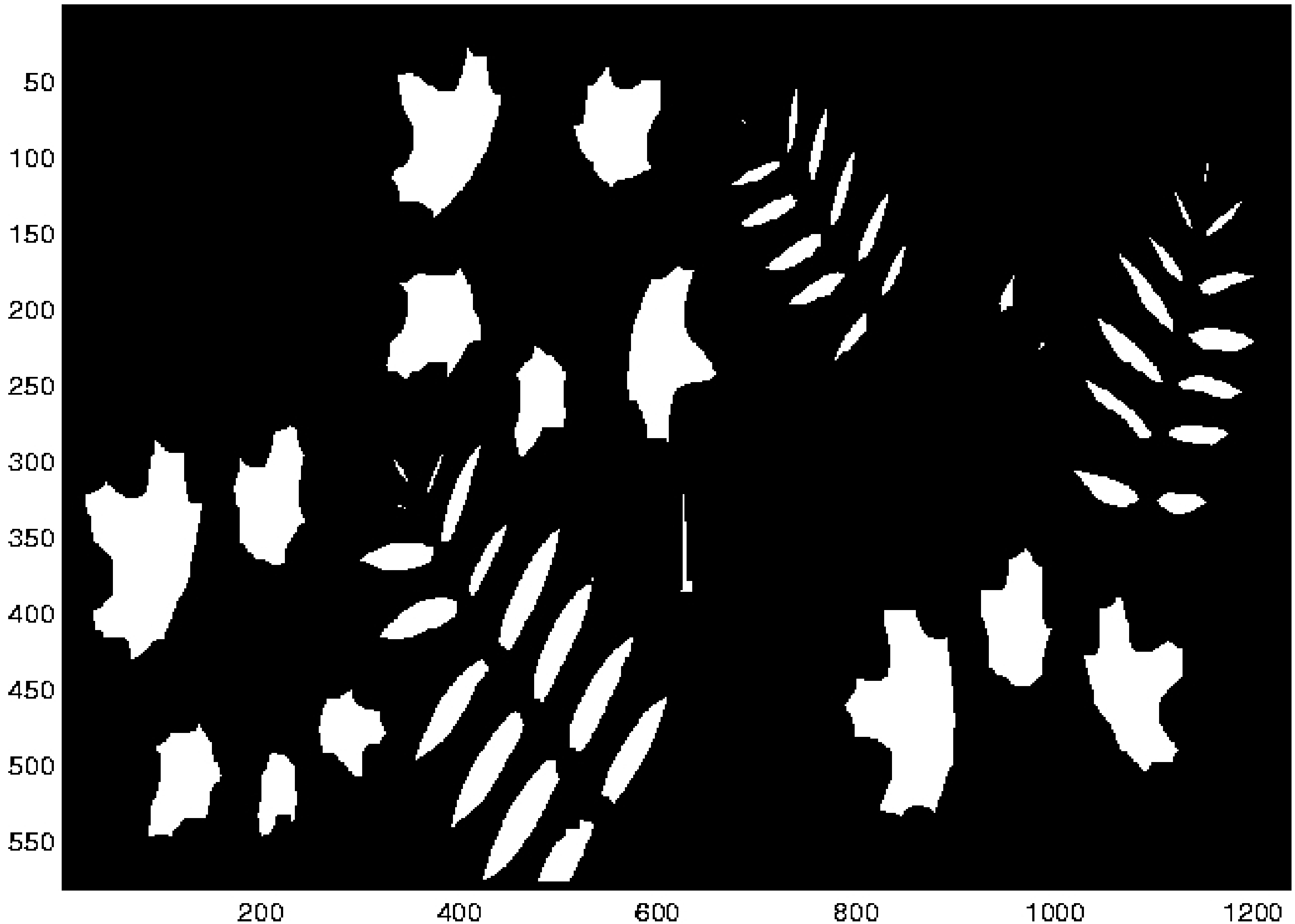


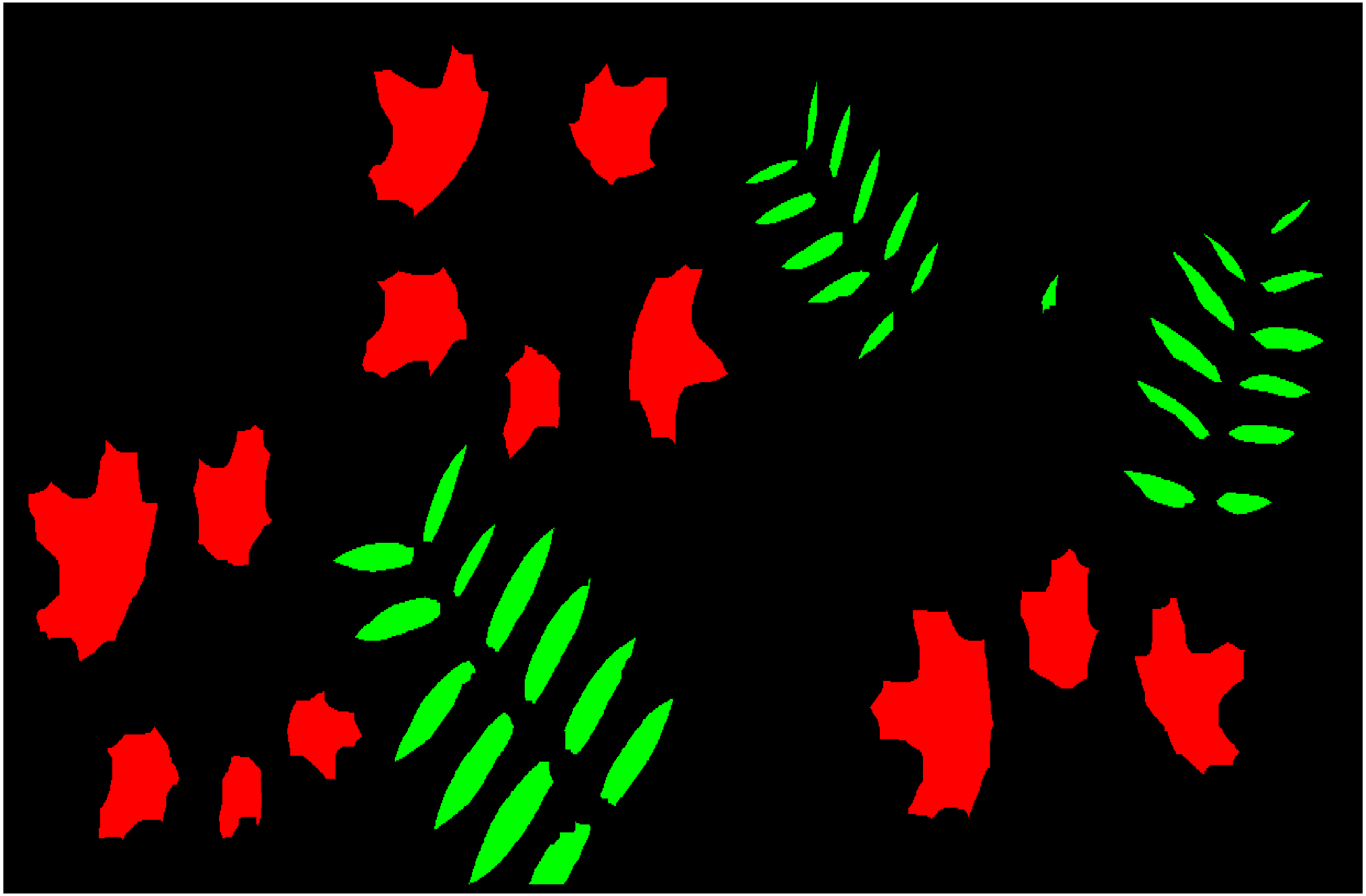
Image with unclassified objects

Application: Recognising and classifying leaves



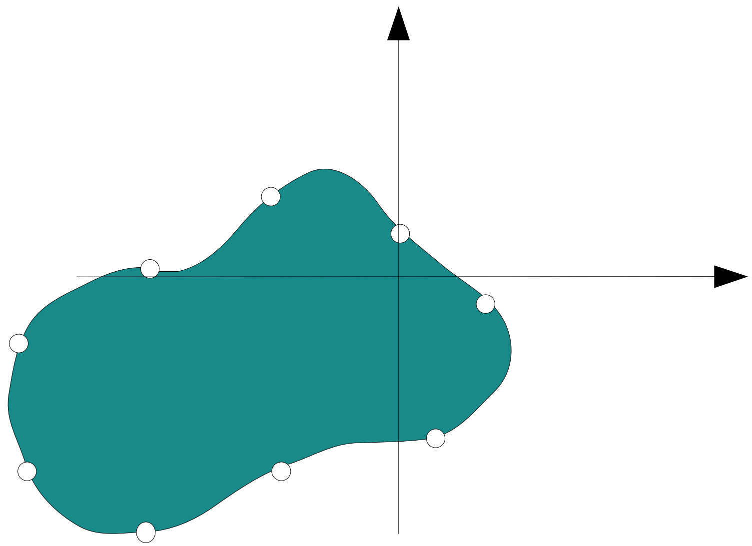
Segmented Objects
(Thresholding)

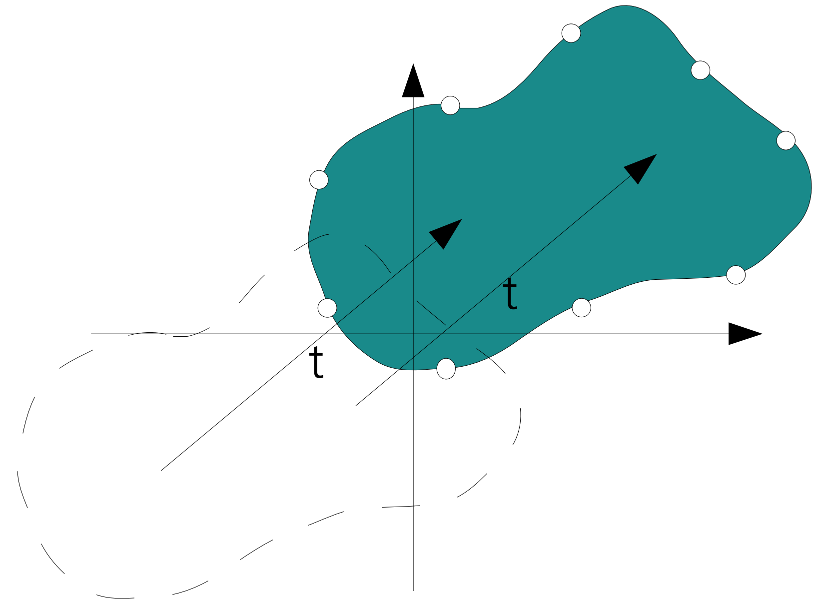
Application: Recognising and classifying leaves



Leaves detected and classified

Translation





$$\tilde{\mathbf{U}}_k \iff \tilde{\mathbf{F}}_\mu$$

$$t + \tilde{\mathbf{U}}_k \iff ?$$
$$t = \delta x + \imath \delta y$$

Translating ***U*** by *t*:

$$\boxed{\tilde{\mathbf{F}}_0 \rightarrow \tilde{\mathbf{F}}_0 + N\, t}$$

Derivation (Translation)

$$\begin{aligned}\tilde{\mathbf{G}}_\mu &= \sum_{k=0}^{N-1} \left(t + \tilde{\mathbf{U}}_k\right) \exp\left(-\frac{2\pi\imath}{N}\mu k\right) \\ &= \mathrm{DFT}\left[\tilde{\mathbf{U}}_k\right] + \boxed{\sum_{k=0}^{N-1} t \exp\left(-\frac{2\pi\imath}{N}\mu k\right)}\end{aligned}$$

► ***μ* = *0***:

$$\sum_{k=0}^{N-1} t \exp\left(-\frac{2\pi\imath}{N}0\right) = N\,t \quad \Rightarrow \quad \tilde{\mathbf{G}}_0 = \tilde{\mathbf{F}}_0 + N\,t$$

► ***μ* > *0***:

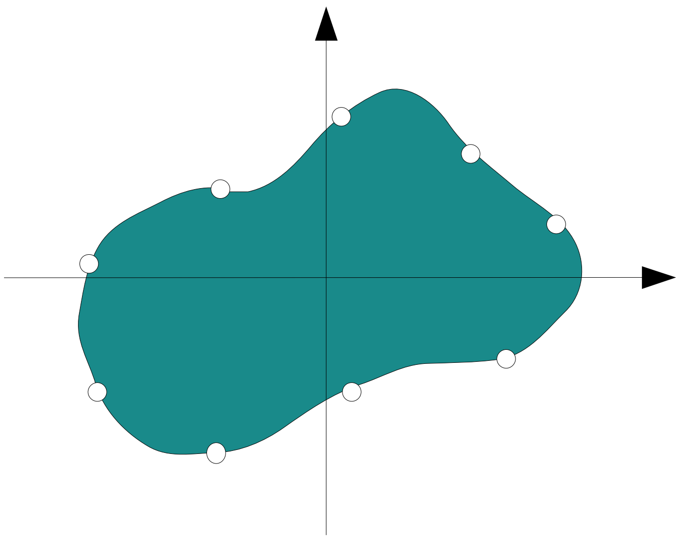
$$\sum_{k=0}^{N-1} t \exp\left(\frac{-2\pi\imath}{N}\mu k\right) = 0$$

(Summation over periodic signal!)

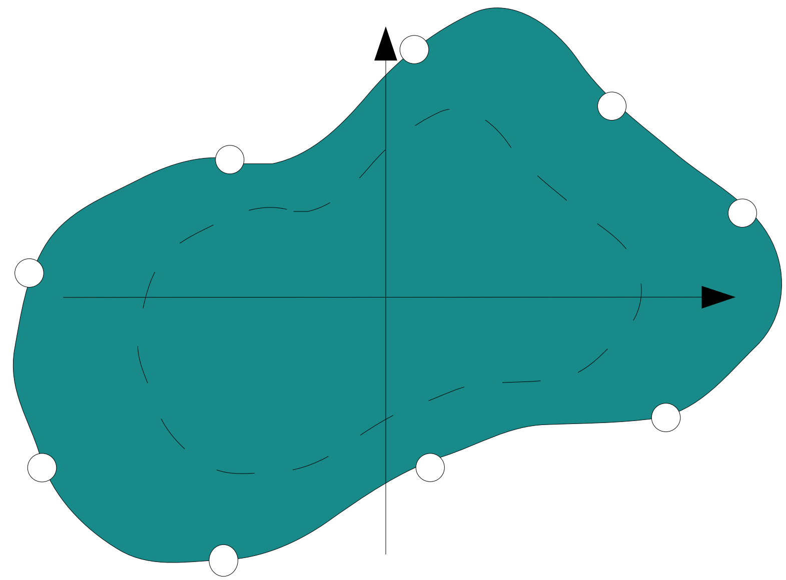
$$\Rightarrow \quad \tilde{\mathbf{G}}_\mu = \tilde{\mathbf{F}}_\mu$$

All information regarding (global) translation is contained in element 0 of the descriptor

Changes in Scale



$$\tilde{\mathbf{U}}_{\mathbf{k}} \iff \tilde{\mathbf{F}}_{\mu}$$



$$s\ \tilde{\mathbf{U}}_{k} \iff ?$$

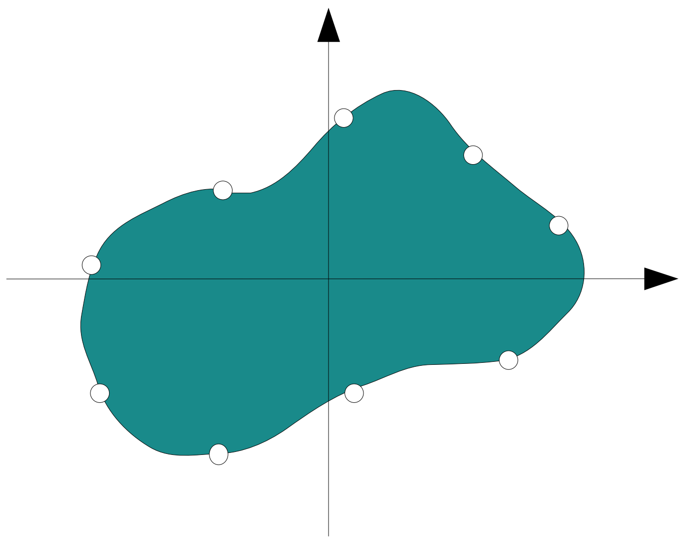
Magnification by factor *s*:

$$\tilde{\mathbf{F}}_{\mu} \ \rightarrow \ s\ \tilde{\mathbf{F}}_{\mu}$$

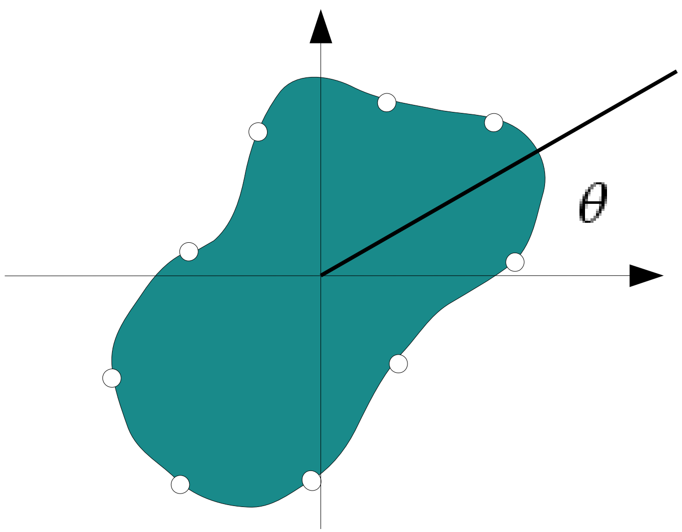
Derivation (Changes in Scale)

$$
\begin{aligned}
\tilde{\mathbf{G}}_\mu &= \sum_{k=0}^{N-1} \left(s\ \tilde{\mathbf{U}}_k \right)\ \exp\left(-\frac{2\pi\imath}{N}\mu k \right) \\
&= s \sum_{k=0}^{N-1} \tilde{\mathbf{U}}_k\ \exp\left(-\frac{2\pi\imath}{N}\mu k \right) \\
&= s\ \mathrm{DFT}\left[\tilde{\mathbf{U}}_k \right]\ =\ s\ \tilde{\mathbf{F}}_\mu
\end{aligned}
$$

Rotation



$$\tilde{\mathbf{U}}_{\mathbf{k}} \iff \tilde{\mathbf{F}}_{\mu}$$



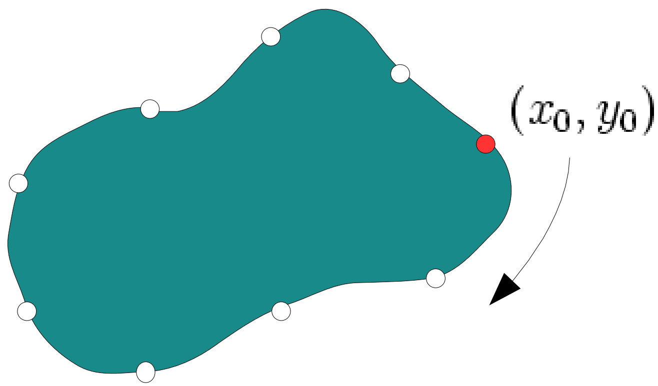
$$\exp(\imath\ \theta)\ \tilde{\mathbf{U}}_{k} \iff ?$$

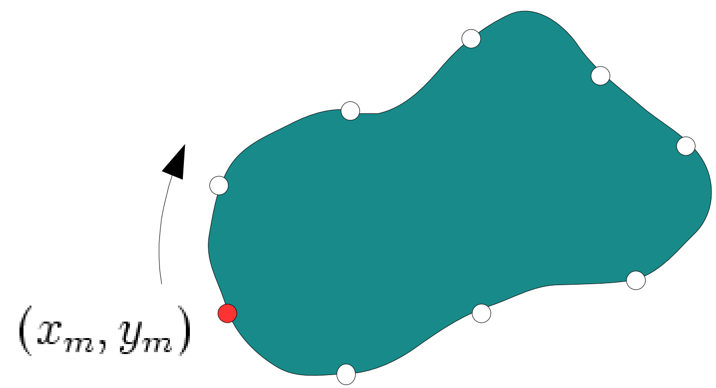
Rotation by an angle θ:

$$\tilde{\mathbf{F}}_{\mu} \rightarrow \exp(\imath\ \theta)\ \tilde{\mathbf{F}}_{\mu}$$

(Derivation identical to scale change: Multiplication by constant)

Starting point





$$\tilde{\mathbf{U}}_{\mathbf{k}} \iff \tilde{\mathbf{F}}_{\mu} \qquad\qquad \tilde{\mathbf{U}}_{k+m} \iff ?$$

Different staring points affect the order of elements in **U** and the **F** obtained.

Changing the starting point by *m* places (pixels):

$$\tilde{\mathbf{U}}_{k+m} \iff \exp\left(\frac{2\pi\imath}{N}\mu m\right)\tilde{\mathbf{F}}_{\mu}$$

Derivation (Starting Point and Mirroring)

- Shifting the starting point by *m* places (pixels)

$$
\begin{aligned}
\tilde{\mathbf{U}}_{k+m} &= \sum_{\mu=0}^{N-1} \tilde{\mathbf{F}}_\mu \, \exp\left(\frac{2\pi\imath}{N}\mu(k+m)\right) \\
&= \sum_{\mu=0}^{N-1} \left(\exp\left(\frac{2\pi\imath}{N}\,\mu m\right)\tilde{\mathbf{F}}_\mu\right) \, \exp\left(\frac{2\pi\imath}{N}\mu k\right) \\
&= \mathrm{IDFT}\left[\exp\left(\frac{2\pi\imath}{N}\mu m\right)\,\tilde{\mathbf{F}}_\mu\right]
\end{aligned}
$$

- Mirroring the contour (left/right or top/bottom or clockwise/anticlockwise)

$$\tilde{\mathbf{U}}_{\mathbf{k}} \iff \tilde{\mathbf{F}}_\mu \qquad\qquad \tilde{\mathbf{U}}_{N-k} \iff ?$$

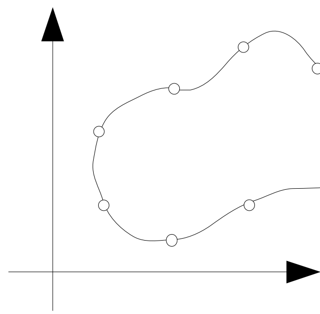
$$
\begin{aligned}
\tilde{\mathbf{U}}_{N-k} &= \sum_{\mu=0}^{N-1} \tilde{\mathbf{F}}_\mu \, \exp\left(\frac{2\pi\imath}{N}\mu(N-k)\right) \\
&= \sum_{\mu=0}^{N-1} \tilde{\mathbf{F}}_\mu \, \exp\left(-\frac{2\pi\imath}{N}\,\mu k\right) \\
&= \mathrm{IDFT}\left[\tilde{\mathbf{F}}_\mu^*\right]^*
\end{aligned}
$$

(* indicates complex conjugation)

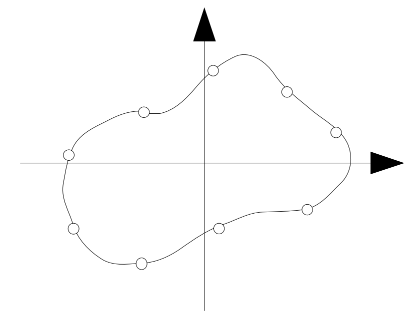
Normalisation (1)

- **Translation Invariance:** Centre the contour at the origin

Translation by t:
$\tilde{\mathbf{F}}_0 \rightarrow \tilde{\mathbf{F}}_0 + Nt$



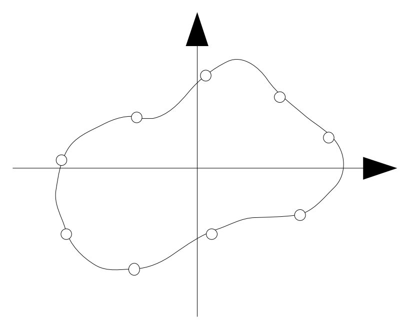
$$\Longrightarrow \quad \boxed{\tilde{\mathbf{F}}_0 := 0}$$



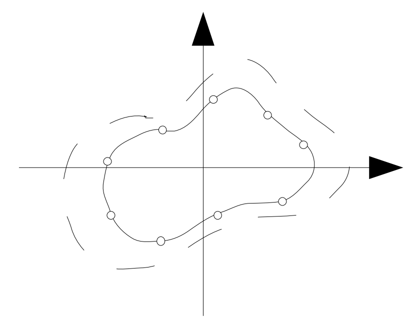
(The 0 frequency contains all and only the information related to translation)

- **Scale Invariance:** Standardise the size of the contour

Scale change by s:
$\tilde{\mathbf{F}}_\mu \rightarrow s\tilde{\mathbf{F}}_\mu$



$$\Longrightarrow \quad \boxed{\tilde{\mathbf{F}}_\mu := \frac{\tilde{\mathbf{F}}_\mu}{|\tilde{\mathbf{F}}_1|}}$$



Consider: $\tilde{\mathbf{G}}_\mu = a\,\tilde{\mathbf{F}}_\mu \quad \tilde{\mathbf{H}}_\mu = b\,\tilde{\mathbf{F}}_\mu$ then $\tilde{\mathbf{G}}_\mu / \left|\tilde{\mathbf{G}}_1\right| = \tilde{\mathbf{H}}_\mu / \left|\tilde{\mathbf{H}}_1\right| = \tilde{\mathbf{F}}_\mu / \left|\tilde{\mathbf{F}}_1\right|$

Normalisation (2)

- **Rotation** and **changes in starting point:** Affect only the phase of the descriptor

Rotation by angle θ:

$$\tilde{\mathbf{F}}_\mu \rightarrow \exp(\imath\theta)\tilde{\mathbf{F}}_\mu$$

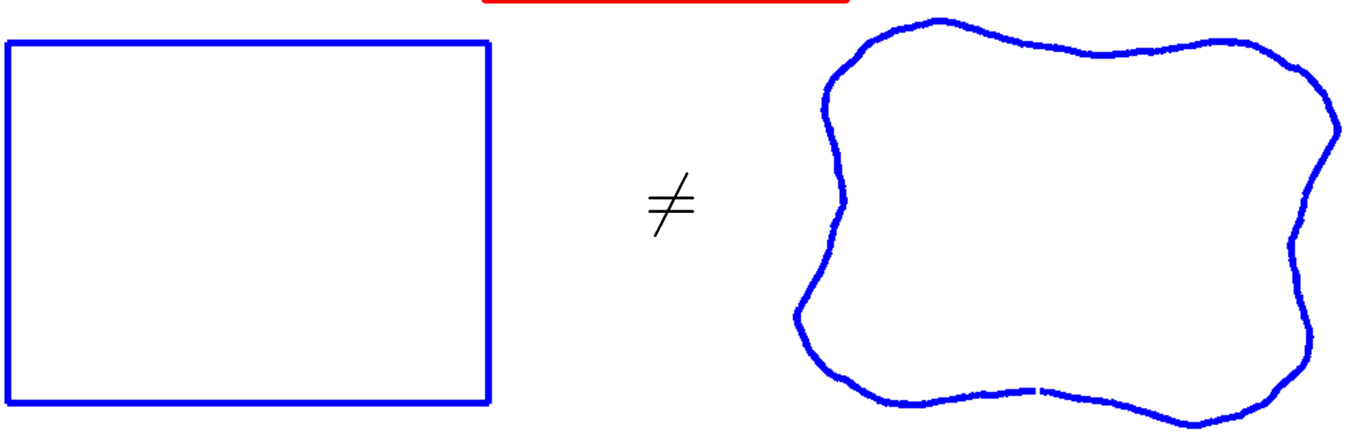
Moving the starting point by *m* places:

$$\tilde{\mathbf{U}}_{k+m} \Longleftrightarrow \exp\left(\frac{2\pi\imath}{N}\mu m\right)\tilde{\mathbf{F}}_\mu$$

- ➔ Simple Solution: Simply remove all phase information
- ➔ Consider only absolute values of the descriptor elements:

$$\boxed{\tilde{\mathbf{F}}_\mu := \left|\tilde{\mathbf{F}}_\mu\right|}$$

But:



(Information loss, both shapes have the same amplitude spectrum)

Exercises

- **`function plotFD(F)`**
  - → `F`: The (non-normalised) Fourier descriptor of a contour
  - Plots the contour `U` described by `F`
  - (Note: the MATLAB `plot` command also accepts complex numbers)

- **`function [G]=shiftFD(F, x, y)`**
  - → `F`: The (non-normalised) Fourier descriptor of a contour
  - → `x`: X-Translation
  - → `y`: Y-Translation
  - → `G`: The (non-normalised) Fourier descriptor of the shifted contour
  - Translates the contour corresponding to `F` by (`X,Y`) Pixels
  - This operation does not require an (inverse) DFT!

- **`function [G]=scaleFD(F,scaleFactor)`**
  - → `F`: The (non-normalised) Fourier descriptor of a contour
  - → `scaleFactor`: Change in contour scale
  - → `G`: The (non-normalised) Fourier descriptor of the scaled contour
  - Scales the contour corresponding to `F` by `100*scaleFactor %`.
  - This operation does not require an (inverse) DFT!

- **`function [G]=resizeFD(F, n)`**
  - → **`F:`** The Fourier descriptor of a contour
  - → **`G:`** The a resized Fourier descriptor with n elements
  - Processes **`F`** to obtain **`G`**, which has been shortened to contain only `n` elements (assume `F` has more than `n` elements).
  - Use `fftshift` to move high „effective“ frequencies to the beginning and end of `F`
  - Remove elements from beginning and end until the length equals `n`.

- **`function [G]=normaliseFD(F)`**
  - → `F`: The (non-normalised) Fourier descriptor of a contour
  - → `G`: The normalised, invariant Fourier descriptor of the contour
  - Processes `F` to obtain `G`, which is invariant to translation, rotation and scaling of the original contour.

- **`function [diff]=compareFD(F, G)`**
  - → `F`: Normalised, invariant Fourier descriptor
  - → `G`: Normalised, invariant Fourier descriptor
  - → `diff`: A measure of the difference between `F` and `G`
  - Quantifies the difference between `F` and `G` (see lecture notes)

- **`function [F]=extractFD(U)`**
  - `U`: A vector of complex pixel coordinates on the contour of an object **(in arbitrary order!)**
  - `F`: The non-normalised Fourier Descriptor of the contour
- Extracts the Fourier Descriptor of a contour, described by a vector of coordinates
- The vector must be sorted before further processing (MATLAB function `sort`)
- The order of pixels can be determined using the direction, or angle, from the object centre to each pixel (assume mostly convex objects).

Recognising and Classifying Leaves (Training Phase)

The following approach to recognition and classification is to be implemented in one, or several (better), MATLAB functions

- Load the example grayscale images for both types of leaf and segment them using a threshold
  - The threshold can be a constant, determined empirically
  - Thresholding in MATLAB is a single line of code! (Logical Operator)
  - The resulting images have the value of 1 (true) inside leaves, and 0 (false) outside.
- Extract the one-pixel thick contour of the segmented leaves
  - Use morphological operators (i.e. MATLAB functions `imerode/imdilate`)
- Use `extractFD`,`normaliseFD` and, finally, `resizeFD` to characterise both types of leaf by two invariant Fourier Descriptors of equal length

Recognising and Classifying Leaves (Classification Phase)

- Load the grayscale test image and segment it by thresholding
  - The result is a binary image in which leaves, branches and other objects are assigned 1 (true) and the background is 0 (false)
- Use a morphological operator to separate leaves from branches and other objects (e.g. MATLAB function `imopen`)
- Extract the one-pixel thick contour of the remaining objects
  - Morphological operators again
- Enumerate the obtained contours of objects using the MATLAB function `bwlabel`
- For each contour:
  - Determine the coordinates of pixels on the contour using `find`
  - Extract an invariant description (with a standard length) of the contour using `extractFD, normaliseFD` and `resizeFD`, and use `compareFD` to compare the description to both types of leaf
  - Assign the contour to the more similar type of leaf
- Plot the classified contours, using colour to indicate the classification result.