What is an image?

A magic number

file

w

h

width

height

payload (pitch)

w x h pixels

(size 2D array)

This array pixel
header:

P6

# comment (0+ of them)

will

make (more color)

→ when parsing header, read char by char...

get

etc

get

etc
PPM binary
- color : 3 channel

PGM binary
- grayscale : 1 channel

PBM binary
- binary : 1 channel (0/1)

B1, B2, B3 ASCII encoded
- image (not binary)
We use P6, P5
we *read*, *write*
to *read*, write 1 byte
(unsinged char) at a
time. P5

Image: P6
header

1 channel per pixel
(1 float)

2 channels per pixel
(2 float)
When reading the J (grayscale) channel over the R, G, B (color) channel:

1) Cast to float \([0, 255]\)
2) Divide by \((\text{float})\max\) pixel channel value

Normalize pixel channel value to \([0, 1]\) same as float
Image planes

(1D array per channel.)
Pjm_read
Pjm_alloc(row, col)
Pjm_write
Pjm_free

Pjm_recul
Pjm_alloc
Pjm_write
Pjm_free

Pjm.h
Pjm.c
as $q ightarrow m$

$0.75 ightarrow 0.6$

$0.1 + 0.3 + 0.5 = 0.9$

in pd or rpm?

Mm extends from class (008 technology)
as $q \to 1$:

$$I_g(x,y) = \frac{1}{3} I_r(x,y) + \frac{1}{3} I_s(x,y) + \frac{1}{3} I_b(x,y)$$

$$I(x,y) = I_i(i,i)$$
\[(f * g)(c) = \sum_{j=1}^{m} g(c) f(c - \frac{j}{2}) \]

**Filter**

\[g(c) = [1, 1.5, .5, .1] \]

\[f(c) = [1, 2, 3, 4, 5, 6, 7, \ldots] \]

**NumPy:** save, full, value!
separability of filters:
Solve: $\int dx$

$dy$

$Jx$

$1 \begin{array}{cc}
1 & 0 \\
0 & 1 \\
\end{array}$

$I \begin{array}{cc}
1 & 0 \\
0 & 1 \\
\end{array}$

$I \times Jx$

$\binom{g_x}{g_y}$

$(g_x, y)$ outside $2x$
Solved for $x$

2D fit for

\[ g_x = dx \]

\[ g_y = dy \]
\[ r_x(x) = \int g_x(\tilde{x}) \times I \, dx/dt \]

\[ r_y(x) = \int g_y(\tilde{x}) \times I \, dy/dt \]

\[ \nabla r(x) = \begin{pmatrix} r_x(x) \\ r_y(x) \end{pmatrix} \]

\( \nabla : \text{gradient} \)
\[ \theta = \arctan \left( \frac{v_y(x)}{v_x(x)} \right) \]

\[ \| \nabla \phi(x) \| = \sqrt{v_x(x)^2 + v_y(x)^2} \]

\[ \nabla \phi \text{ gradient} \]

\[ \text{magnitude} \]

\[ \text{div} \]

\[ \hat{\mathbf{u}}(\theta) = (\cos(\theta), \sin(\theta)) \]

LiWedge + 

\[ \text{sing}_N \in \mathbb{S}_w \]
Signal: $u_1 = 7$

Filter:

Let $u_2 = 3$

$u_1 + u_2 = 10 = 9$

Full

$Numpy$

$0, 1, 2.5, 4, 5, 5, 7, 8.5, 10, 7.5$

$u_1 + u_2 = 12 - 3 = 9$
\[ u_1 + u_2 - 1 = 9 \]

'full'

\[ \max (u_1, u_2) = 7 \text{ [same third or signal]} \]

\[ \max (u_1, u_2) - \]

'valid'

\[ \min (u_1, u_2) + 1 = 5 \]

\[ g(c) = [0, 1, 5] \quad f(i) = [1, 2, 3, 4, 5, 6, 7] \]

\[ (f \times g)(i) = \sum_{j=1}^{m} g(c) \cdot f(i-j) \]
\[ \sum_{k=0}^{m_1} q(c_i) f(i-k) \]

**How to code?**

**Initial:***

1. Signal filter length of signal

2. For \( n = 0; n < n_1 + n_2 - 1; n++ \)

3. For \( k = 0; k < n \& \& k \)

4. \( k < \text{max}(n_1, n_2); k++ \)

5. \( f_v1[n] += (k < n1 ? v1[k] : 0) \times ((n-k < n2 ? v2[n-k] : 0)) \)
\[ f(i) : \text{the filter} \quad v_2 \text{ in code} \]
\[ g(i) : \text{the signal} \quad v_1 \text{ in code} \]

\[
(f \ast g)(r) = \sum_{k=1}^{\infty} f(i) g(i_k-r)
\]

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

like holding filter in place, sliding signal along.
When to 'hold forth' in place?

When streaming data, in real-time

\[ \text{past} \quad \text{future} \]

\[ \text{data streaming in} \]

\[ \text{data} \quad \text{past} \quad \text{future} \]

\[ \text{depth of context} \]

\[ \text{101-1} \]

\[ \text{for position} \quad \text{fun fun fun} \]
derivative of $f(x)$. 
Back to arg ϕ2: use separate filter

Step 1: along rows

Step 2: along cols

Gx
By:

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
\]
\[ C \times \]

\[
\begin{array}{ccc}
1 & 0 & 2 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{array}
\]
for (i = 0; i < rows; i++)
  for (j = 0; j < cols; j++)
    for (k = 0; k < m; k++)
      for (l = 0; l < n; l++)
        separate fitter doi away with \( O(\text{rows} \times \text{cols} \times \text{m} \times \text{n}) \)
$O(\text{row} \times \text{col} \times m) + O(\text{row} \times \text{col} \times n)$

a little bit of savings

In theory, separable filters are a little bit more efficient
$G_x, G_y$ images: you get 2

one with horizontal

" or vertical edge

- to output these as

'verible' image don't

forget to normalise

1) find max value

2) Divide each pixel by max

then write out to file
Normalizing: semi-final step before output

to use edge data (e.g., getting gradient info)
then (it normalize)

Cx, Cy images
\[ \sqrt{G_x^2 + G_y^2} \]

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

same i, j

\[ \text{mag.} \]

\[ \text{dir.} \]

\[ \text{disorient this into 4 dir., N, E, W, S} \]
can doometry like colorize image to show dir of edge
asg Ø1:
- couple of instances of not compiling
- couple of C++ implementations
- only 1 subclass implementation.

PWM — no such image
1 ppm
p ppm
1 ppm
p ppm
- Boundary conditions for convol:
  - use either 0 padding
  - use periodic padding (wrap-around indices)

    a la Python [-1]
given \( r, e \) dimensions of image

\begin{align*}
&\text{define } I(i, r) \left( (((r) + (i)) \% (r)) \right) \\
&\text{define } J(j, c) \left( (((c) + (j)) \% (c)) \right) \\
&\text{define } M2A(\cdot,\cdot, r, c) (I(i, (r)) \times (c) + J((j), (c))))
\end{align*}
Periodic extension of image
in ppm.c:  // why in main.c?

PM * ppm_to_grey (PM * cimg)

in + c, j;

PM * gimg = NULL;

float r, q, b, l;  // l for luminance

fimg = ppm_alloc (cimg -> rows, cimg -> cols);
for (c = 0; c < cimg; c++)
for (j = 0; j < cimg; j++)

r = cimg - rpx[c * cimg + j];
y = " qpx ";
5 = " hlx ";

l = 0.3 * r + 0.7 * y + -0.5;
ji = 83 * ppx[c * pimg + j] = l;

return ji;
- Fourier Transform, etc. → DWT (Discrete Wavelet Transform)

- Key aspects:
  - Both FT & DWT are lossless
  - Image in frequency space
  - Projection onto Basis Function
  - "Perfect reconstruction" (vectors)

**Formulas:**

\[ I(i,j) \leftrightarrow F(I(i,j)) \]
- Texts to consult:
  - Szeliski (Section 3.4)
  - Glassner: Digital Image Synthesis
  - Comp. graphics: ray tracing
Often we pose the question, 

"Given the function \( h(x, y) \) or \( h(k, l) \) smooth, find its Fourier transform \( \mathcal{F} \)."

\[
H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} \, dx \, dy
\]

\( \omega_x, \omega_y \) are frequency Omega i.e. frequency
- Discrete form:

\[ H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j 2 \pi (k_x x + k_y y)} \]

- I used to be quite confused by this.

- What's the \( e^{-j (wx + wy)} \) mean?

- Rewrite as \( e^{-j (wx + wy)} \) \((i \leftrightarrow j)\)

  Using Euler's formula:

  \( e^{ix} = \cos wx + i \sin wy \)
\[ \cos, \sin : \]

\[ \text{waveform} \]

\[ \text{freq vs.} \]

\[ \text{sin freq.} \]

\[ \text{same cos function, different parameters} \]

\[ \text{different frequencies} \]

\[ \text{it's a basic function!} \]
- Plot just project spatial info \((x, y)\) and cos, sin basis (free info)
- also, don't forget \(e^{ix} = \cos x + is\sin x\)

Reminiscent of polar coordinates

\[(\rho, \theta)\]

just a different coordinate system

\(r, \theta\)
- FT : takes a signal (e.g. wave) and represents it in frequency space.

- FT is a mathematical operator that decomposes a signal to a sum of weighted sines and cosines (basis functions).
Any signal can be represented as a combination of basis functions — even our Euclidean space.

- **Orthogonal vectors** (⊥ perpendicular)

\[
\begin{align*}
V_x &= [1, 0]^T \\
V_y &= [0, 1]^T
\end{align*}
\]

**Normalized**

\[
\|V_x\| = \sqrt{1^2 + 0^2} = 1 \\
\|V_y\| = \sqrt{0^2 + 1^2} = 1
\]

**Orthogonal and normal / orthonormal**

\[
\begin{bmatrix}
1 \\
0
\end{bmatrix} \cdot \begin{bmatrix}
0 \\
1
\end{bmatrix} = 1 - 0 + 0 \cdot 1 = 1
\]

**Dot product**
- test for orthogonality (are they \( \perp \))?
dot product: sum of element product

\[
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix} \cdot \begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix} = x_1y_1 + x_2y_2 + \ldots + x_ny_n = \sum_{i=1}^{n} x_iy_i
\]

\[
\begin{bmatrix}
  1 \\
  0
\end{bmatrix} \cdot \begin{bmatrix}
  1 \\
  1
\end{bmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0 + 0 = 0
\]

\[
\cos(\theta) = 0 
\Rightarrow \theta = \cos^{-1}
\]

\[
\theta = 90^\circ 
\Rightarrow \cos = 0
\]

\[
\theta = 0^\circ 
\Rightarrow \cos = 1
\]
any vector \( \vec{v} = (x, y) \) can be represented as a combination of orthogonal basis vectors.

\[ \vec{v} = (x, y) = (V \cdot V_x)V_x + (V \cdot V_y)V_y \]

Dot product which is a scalar (just a number, not a vector)
Example: \[
\left( \begin{array}{c}
4 \\
3
\end{array} \right) \cdot \left( \begin{array}{c}
0 \\
0
\end{array} \right) + \left( \begin{array}{c}
4 \\
3
\end{array} \right) \cdot \left( \begin{array}{c}
0 \\
1
\end{array} \right)
\]

\[
\left( \begin{array}{c}
(4 \cdot 0) + (3 \cdot 0) \\
(4 \cdot 0) + (3 \cdot 1)
\end{array} \right) + \left( \begin{array}{c}
4 \\
3
\end{array} \right) \cdot \left( \begin{array}{c}
0 \\
1
\end{array} \right)
\]

\[
4 \left( \begin{array}{c}
1 \\
0
\end{array} \right) + 3 \left( \begin{array}{c}
0 \\
1
\end{array} \right) = \left( \begin{array}{c}
4 \\
0
\end{array} \right) + \left( \begin{array}{c}
0 \\
3
\end{array} \right) = \left( \begin{array}{c}
4 \\
3
\end{array} \right)
\]

\[\text{Ans.}\]
\[ \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{n} v_i w_i = \cos(\theta) \]

\[ \theta = \cos^{-1} \left( \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} \right) \]

**Ex.**

\[
\begin{bmatrix}
1 & 3 & -5
\end{bmatrix}
\begin{bmatrix}
4 \\
-2 \\
-1
\end{bmatrix}
= (1)(4) + (3)(-2) + (-5)(-1)
\]

\[
\begin{bmatrix}
1 \\
2 \\
1
\end{bmatrix}
\begin{bmatrix}
2 \\
3
\end{bmatrix}
= 2 + 6 + 3
\]

\[
1 \times 2 \\
2 \times 1 \\
1 \times 1
\]

\[
= 3
\]
key observation is that basis vectors were scaled

\[ V = (\mathbf{v} \cdot \mathbf{v}_x) \mathbf{v}_x + (\mathbf{v} \cdot \mathbf{v}_y) \mathbf{v}_y \]

\underline{scalar} \quad \underline{scalar}

projection of vector onto space defined by basis vectors (or functions)

- for discussion usually stand with \( f(x) \) function

\[ y = f(x) \]
\[- y = f(x) \]

Partial domain

\[ f(t) \text{ can be written as linear combination of weighted basis functions:} \]

\[ f(t) = c_1 \phi_1(t) + c_2 \phi_2(t) + \ldots + c_n \phi_n(t) \]

Temporal domain

\[ x \in (0, t) \]

Weight or coefficients
Example: 'bar chart' function:

\[ f(x) = (1, 4, 2, 3, 2) \]
\[ x = 1, 2, 3, 4, 5 \]

\[ f(x) < 1.5 \] \[ \Rightarrow \quad f(x) = \begin{cases} f(x), & x < 5 \\ 0 & \text{otherwise} \end{cases} \]
\[ Sx = \begin{bmatrix} 1 & 0 & -1 \\ -2 & 0 & -2 \\ 0 & 1 & -1 \end{bmatrix} \]

\[
3 \times 3
\]

\begin{align*}
&\text{for} (i = 0; i < \text{rows}; i++) \\
&\quad \text{for} (j = 0; j < \text{cols}; j++) \\
&\quad \quad f(k = 0; k < 3; k++) \\
&\quad \quad \text{for} (l = 0; l < 3; l++) \\
&\quad \quad \quad \text{Diag} [i*\text{cols}+j]++
\end{align*}

i+k, j+t
\[ G_x = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \end{bmatrix} \]
\[ G_y = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 & 1 \end{bmatrix} \]
result: org image (unfiltered)
dtx image } there are
dty image } edge-detected
images,

at this point you UNNORMALIZED
can use dstx, dsty for gradient
image info if you wish

BUT, for output, must normalize dstx

dsty
\[ x' = \frac{(x-a)}{(b-a)} \times (d-c) \times (d-c) + c \]

(normalization) (scaling) (shifting)
\[ e^{-\frac{x}{\delta^2}} \]
\[ e^{-\frac{(x-2)}{\delta^2}} \]
\[ e^{-\frac{x}{2\sigma^2}} \]
\[ \left( \begin{array}{c} 195 \\ 25 \end{array} \right) \] is still normalized? Yes

\[ \left( \begin{array}{c} 255 \\ 9 \end{array} \right) \]

\[
\begin{align*}
\text{max} & \quad \text{min} \\
1 & \quad 0
\end{align*}
\]
Darle to basin function

$f(t) \in \{1, 4, 2, 5\} \subset \{f(5)\}

E \in \{1, 2, 3, 4, 5\}

f(t) \in [1, 5] \subset \{f(t), 1 \leq f(t) \leq 5\}, \text{ elsewhere}
basis functions $\varphi_i(t) \in \varphi_5(t)$

$$\varphi_i(t) = \begin{cases} 
0, & t < i-1 \\
1, & i-1 \leq t < i \\
0, & t \geq i 
\end{cases}$$

$i \in \{1, 2, 3, 4, 5\}$
- Draw chart \( p(t) \)

\[
p(t) = \beta_1 \phi_1(t) + \beta_2 \phi_2(t) + \cdots + \beta_5 \phi_5(t)
\]

\[
= \sum_{i=1}^{5} \beta_i \phi_i(t)
\]

- Normally, we'd just have 1 basis function & shift it

- Basis functions

- Coefficients
\[ \delta(t) \quad \frac{i}{\pi} \quad \text{then} \]

\[ \text{must be} \]

where \( \psi \) \( \text{need to sample} \).

\[ \delta(t-k) \]

\[ \phi(t-k) \text{ needs to be mutually orthogonal to span space} \]
In general,

\[ \int_{t_0}^{t_1} \phi_i(t) \phi_j(t) dt = \begin{cases} \delta_{ij} & \text{if } i = j \neq 0 \text{ or } i = j = 0, \\ 0 & \text{otherwise}. \end{cases} \]
For vectors \( \mathbf{v}_x, \mathbf{v}_y \):

\[
\langle \mathbf{v}_x, \mathbf{v}_y \rangle = \begin{bmatrix} 1^0 \end{bmatrix} \cdot \begin{bmatrix} 1 \end{bmatrix} = 1 \neq 0
\]

(Overlap \( \Rightarrow \) Orthogonal)

\[c \neq j\]

When \( i = j \):

\[
\langle \mathbf{v}_x, \mathbf{v}_x \rangle = \begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \end{bmatrix} = 1(1) + 0(0) = 1
\]

\[\text{Diagonal} \Rightarrow \text{Orthogonal}\]

(Same for \( \langle \mathbf{v}_y, \mathbf{v}_y \rangle \))
For $\phi$, system is normalized if

$$\langle \phi_i, \phi_i \rangle = 1$$

$$\langle \phi_i, \phi_i \rangle = 0 \quad \text{orthogonal}$$

One can normalize by dividing by the norm or length

$$\langle \phi_i, \phi_i \rangle = \| \phi_i \|$$
asg 82:
- generally well done
- I appreciate included images
- C++ ok, but... try to update Makefile
- one person used 2D kernel
- do use the object-oriented C approach
  (avoid the huge main() function)
Let's say \( f(t) = \sum_{i=1}^{n} c_i \phi_i(t) \)

is an approximation (mapping w/ basis func. coeffs. (weights))

- How to minimize the error?
- Define Mean-Squared Error (MSE):

\[
MSE = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \left[ f(t) - \sum_{i=1}^{n} c_i \phi_i(t) \right]^2 dt
\]

over interval \((t_1, t_2)\)
- rewrite in long form:

\[ \frac{1}{\epsilon_2 - \epsilon_1} \int_{\epsilon_1}^{\epsilon_2} \left[ f(t) - c_1 \phi_1(t) - c_2 \phi_2(t) - \ldots - c_n \phi_n(t) \right]^2 dt \]

- take the square:

\[ \int_{\epsilon_1}^{\epsilon_2} \left[ f^2(t) + c_1^2 \phi_1^2(t) + \ldots + c_n^2 \phi_n^2(t) - 2c_1 f(t) \phi_1(t) - \ldots - 2c_n f(t) \phi_n(t) \right] dt \]

\[ \Gamma_{a-b}[a-b] = a^2 - 2ab + b^2 = a^2 + b^2 - 2ab \]
\[
\frac{1}{\epsilon_2 - \epsilon_1} \left\{ \left( \int_{\epsilon_1}^{\epsilon_2} f^2(t) \, dt \right) + c_1^2 \xi_1 + c_2^2 \xi_2 + \cdots + c_n^2 \xi_n - 2c_1 \gamma_1 - 2c_2 \gamma_2 - \cdots - 2c_n \gamma_n \right\}
\]

where
\[ k_i = \langle \phi_i, \phi_i \rangle \]
\[ \xi_i = \langle f, \phi_i \rangle \]
the \( c_1 x_1 + c_2 x_2 + \cdots + c_n x_n - 2c_1 y_1 - 2c_2 y_2 \cdots - 2c_n y_n \)

can be written as

\[
\left( c_i^2 k_i - 2 c_i \sqrt{k_i} \right) = \left( c_i \sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}} \right)^2 - \frac{\gamma_i^2}{k_i}
\]

to complete the square,

\[
\left( c_i \sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}} \right) \left( c_i \sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}} \right) - \frac{\gamma_i^2}{k_i}
\]

\[
= c_i^2 k_i - 2c_i \gamma_i + \frac{\gamma_i^2}{k_i} - \frac{\gamma_i^2}{k_i}
\]
\[ \text{MSE} = \frac{1}{t_2 - t_1} \int_{t_2 - t_1}^{t_2} f^2(t) \, dt + \sum \left( c \sqrt{k_i} - \frac{c_i}{\sqrt{k_i}} \right)^2 \]

To minimize, set

\[ c_i \sqrt{k_i} = \frac{c}{\sqrt{k_i}} \]

so that for each \( i \), the squared error goes to 0.
- divide both sides by \( \sqrt{\kappa} \) and solve for \( \xi' \):

\[
\xi' = \frac{\gamma_i}{\kappa} = \frac{\int_{\xi_1}^{\xi_2} f(\xi) \phi_i(\xi) d\xi}{\int_{\xi_1}^{\xi_2} \phi_i^2(\xi) d\xi} = \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle}
\]

- is the end, we finally convolve \( f(\xi) \) with basis function \( \phi_i(\xi) \)

- basically like \( (\mathbf{u} \cdot \mathbf{v}_x) \mathbf{v}_x + (\mathbf{u} \cdot \mathbf{v}_y) \mathbf{v}_y \)
- In other words, we just project \( f(t) \) into the space defined by \( \phi_s(t) \).

- Basically a change of coordinate space from pixels to frequency space.
The FT uses \( \Psi_n(f) = e^{2\pi i n f} \) for \( n \in \mathbb{Z} \) is a set of digitons (as opposed to \( \mathbb{R} \) set of reals).

Wavelet transform projects onto \( \phi \) and \( \psi \).

Scaling function \( \varphi \)

Wavelet function \( \psi \)
- Fourier series
  \[ x(t) = \sum_{k} a_k e^{ik\omega t} = \langle x, \psi_k \rangle \]

- Signal
- Plot of \( f(t) \)

- Fourier series coefficients
  \[ a_k = \langle x, \psi_k \rangle \]

- Over interval \( T = \frac{2\pi}{\omega} \)

- Plots for different \( n \):
  - \( n = 0, \omega = 0 \)
  - \( n = 1, \omega = 1 \)
  - \( n = 2, \omega = 2 \)
- Fourier Transform
\[ X(\omega) = \frac{1}{\sqrt{2\pi}} \int x(t) e^{-i\omega t} \, dt = \frac{1}{\sqrt{2\pi}} \langle x, \psi \rangle \]
\[ x(t) = \frac{1}{\sqrt{2\pi}} \int X(\omega) e^{i\omega t} \, d\omega = \frac{1}{\sqrt{2\pi}} \langle \tilde{x}, \psi \rangle \]
\[ X(\omega) = F(x(t)) \]
\[ x(t) = F^{-1}(X(\omega)) = F^{-1}(F(x(t))) \]
\[ x(t) \xrightarrow{F} X(\omega) \]
- key: Fourier basis functions have infinite support - they "have fun" exist everywhere in the signal domain.

- FT integrates over entire signal.

- If there is a high freq. burst somewhere, FT will say it is there but not where it is.
- wavelet basis functions: compact support

$\phi$ to outside local support

Daubechies
- Wavelet:
  - 2D family of functions derived from scaling function $\phi$ - the one most smoothly reduces signal in resolution

$\phi$  $\psi_d$  $\psi_l$  $\psi_h$

$\phi$ (scaling function) $\psi_l$ (low-pass filter)
$\psi_d$ (detail coefficients) $\psi_h$ (high-pass filter)
- 2D implementation (using 1D wavelet filters) e.g., for wavelet browser page

\[ \phi \approx h \]

\( h : 3 \left( \frac{1}{12}, \frac{1}{12} \right) \quad g : \left\{ \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right\} \)

normalized Haar filter

[Diagram of normalized Haar filters]

\[ \frac{1}{\sqrt{2}} \]

\[ \frac{1}{\sqrt{2}} \]

[Diagram of important notes]
What is the effect of \(H\) so far?

\[ HH = \phi \otimes \phi^T : \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \]

...tensor product ...

\[ = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \]

\(2 \times 2\) matrix.
argues evidence:

1) use error origin vs. ident (e.g.)

2) threshold wavelet coefficient before reconstruction

... (not the smoothed delta)
The 2006 simulation image comparison of wind sounding
- wavelets: start with scaling function

\[ \phi(x) = \sum_{k=0}^{N} c_k \phi(2x-k) \]

- dilatation is applied recursively:

\[ \phi_j(x) = \sum_{k} c_k \phi_{j-1}(2x-k) \]
Example: unit square

\[ s(x, y, lx, ly, s) = \begin{cases} 1, & lx \leq x < lx + s \\ ly \leq y < ly + s \\ 0, & \text{otherwise} \end{cases} \]

\[ s(x, y, lx, ly, s) = s(x, y, x + \frac{s}{2}, y, \frac{s}{2}) + \\
  s(x, y, x + \frac{s}{2}, y + \frac{s}{2}, \frac{s}{2}) + \\
  s(x, y, x + \frac{s}{2}, y + \frac{s}{2}, \frac{s}{2}) + \\
  s(x, y, x + \frac{s}{2}, y + \frac{s}{2}, \frac{s}{2}) \]
box function: \( y_0(t) = \begin{cases} 1, & 0 \leq t \leq 1 \\ 0, & \text{otherwise} \end{cases} \)

\[
y_1(t) = y_0(2t) + y_0(2t-1)
\]
\[ y_n(t) = y_0(2^t) + y_0(2^t - 1) + y_0(2^t - 2) + y_0(2^t - 3) \]

Scaling function: \[ \phi_j(x) = \sum_{k \in \mathbb{Z}} c_k \phi_{j-1}(2^x - k) \]

"Recipe" for nonrecurrent \( \phi \):

\[ y = \sum_{k} (-1)^k c_{k+1} \phi(2^x - k) \]

Some texts use \( c_{1-k} \)
\( \phi : \{ 1, 2, 3, 4 \} \)

\( \psi : \)

a) Reword \( \phi(1-k) \)

b) Negate even odd value

c) 

\[
\begin{array}{ccc}
0 & 1 & 2 & 3 \\
4 & 3 & 2 & 1 \\
\end{array}
\]

5) 

\[
\begin{array}{ccc}
1 & -3 & 2 & -1 \\
\end{array}
\]

Quadratic mirror on test
\( \psi(x) \): \\
\( \phi(x) \): \\
- low pass \\
- deriv. at a certain freq.
and \( \langle \phi, \psi \rangle = 0 \) zero they are orthogonal via above construction

in general,

\[
\phi_{j,k}(x) = 2^{j/2} \phi(2^j x - k)
\]

\[
\psi_{j,k}(x) = 2^{j/2} \psi(2^j x - k)
\]

-factors of 2 mean dyadic dilations

(the \( \ll 1 \) shift operator in code)
- How does this get applied?

\[
\text{scaling} = \sum_{i,k} c_{i,k} \phi(2^j x - k)
\]

\[
g_k(x) = \sum_{i,k} d_{i,k} \psi(2^j x - k)
\]

Deconvolution

Convolution
- Recursively,

\[ f^n(x) = g^{n-1}(x) + g^{n-2}(x) + \ldots + g^{n-m}(x) + f^{n-m}(x) \]

- 1-level decomposition

\[ f'(x) = f'(x) + g^0(x) \]

- 2-level:

\[ f^2(x) = g'(x) + f'(x) = g'(x) + g^0(x) + f^0(x) \]
\[
\begin{align*}
\phi_j^{(s)}(\mathbf{r}) &= \sum_{\lambda_1} g^{(s)}(\lambda_1) \phi_j^{(s)}(\mathbf{r}) \\
\phi_{j+1}(\mathbf{r}) &= g_j(\mathbf{r}) + \phi_j^{(s)}(\mathbf{r}) \\
&= \sum d_{j,k} \phi_{j+k}(\mathbf{r}) + \sum c_{j,k} \phi_{j+k}(\mathbf{r}) \\
&\quad \text{for } j, k \in \mathbb{Z} \quad \text{(integers)}
\end{align*}
\]
- schematically

\[ C^N \xrightarrow{d^{N-1}} C^{N-2} \xrightarrow{d^{N-2}} \cdots \xrightarrow{d^{N-m}} C^{N-m} \}

\text{only deformed}

\text{scaled} f_{\infty}

\text{decomposition}

\text{renormalization}

\[ C^2 \xrightarrow{d^{N-1}} \cdots \xrightarrow{d^{N-m+1}} C^{N-m+1} \xrightarrow{d^{N-m}} \cdots \xrightarrow{d^{N-1}} C \xrightarrow{d^{N-1}} C^{N-1} \]
\[ h : \text{low-pass filter}, \ (\phi) \]

\[
\text{Haar } h_k : \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix}
\]

\[ g : \text{high-pass filter}, \ (\psi) \]

\[
\text{Haar } g_k : \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & -1 \end{bmatrix}
\]
The quadrature mirror function is defined as:

\( g_k = (-1)^k h_{1-k} \) for \( k = 1, 2, 3, \ldots \).

Negate every other element of the sequence:

\( h_k : 1, -2, 5, 7, 4, 12, -4 \)

Add an arbitrary random factor \( h_k \):

\( g_k : -4, -1, -4, -7, 5, 2, -1 \)
— mathematically, decomposition at resolution level $j$:

$$f_\phi^{j-1}(x) = \sum_k h_k f_\phi^j(2x+k)$$

$$f_\psi^{j-1}(x) = \sum_k g_k f_\phi^j(2x+k)$$

$h_k$, $g_k$ are the ID filters.

*the low-pass image at lower level*
- reconstruction is a bit trickier:

\[
\int_0^1 \theta(x+\rho) = (1-\theta) \int_0^{\rho} (x) + (\rho) \int_0^1 (x)
\]

for \( \rho \in [0,1] \)

above (a pre-processing step) reconstruction is:

\[
\int_0^\frac{n}{2} \theta(x+k) = (1-\theta) \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \tilde{h}_k + \frac{1-\rho}{\rho} \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} (x-k) + (\rho) \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \tilde{h}_k f_\theta(x-k)
\]

(Reconstruction finished).

\( \theta = 0 \) \quad \rho = 1
just a convenient way of writing:

\[ f^j_\phi (2\pi) = \sum_{k} \phi_k f^{j-1}_{\phi_k \psi}(\pi - k) \]

\[ f^j_\phi (2\pi+1) = \sum_{k} \phi_k f^{j}_{\phi_k \psi}(\pi - k) \]

\[ \phi_k : \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \]

\[ \phi_k : \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \]
- a numerical example in 1D:

\[ f(x) = x^2 \]

\[ x^2(1) + \frac{1}{x^2}(4) \]

\[ x^2(0) + \frac{1}{x^2}(-2) \]

\[ \frac{-2}{\sqrt{2}} \]

\[ \frac{\sqrt{2}}{\sqrt{2}} \]

\[ \frac{1}{\sqrt{2}} \]
\[ c^1 = \frac{5}{\sqrt{2}} - \frac{2}{\sqrt{2}} \]

\[ d^1 = \frac{1}{\sqrt{2}} \left( \frac{5}{\sqrt{2}} \right) + \frac{1}{\sqrt{2}} \left( \frac{-1}{\sqrt{2}} \right) \]

\[ \frac{9}{\sqrt{2}} \]

\[ c^0 = \frac{1}{\sqrt{2}} \left( \frac{5}{\sqrt{2}} \right) + \frac{1}{\sqrt{2}} \left( \frac{-1}{\sqrt{2}} \right) \]

\[ \frac{3}{\sqrt{2}} \]
\[ f'^2 = c^2 \]

\[ W(f) = \begin{pmatrix} \frac{3}{\sqrt{2}} & \frac{7}{\sqrt{2}} & -\frac{3}{\sqrt{2}} & \frac{2}{\sqrt{2}} \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & \sqrt{2} \end{pmatrix} \]

\[ \text{Reconstruct to get back to original} \]

\[ f = W^{-1}(W(f)) \]
\[ c_1 \cdot d_1: \frac{3}{\sqrt{2}} \quad -\frac{3}{\sqrt{2}} \quad -\frac{1}{\sqrt{2}} \quad \frac{2}{\sqrt{2}} \]

\[ \frac{1}{\sqrt{2}}(\frac{5-3}{\sqrt{2}}) \quad \frac{1}{\sqrt{2}}(\frac{5+3}{\sqrt{2}}) \quad \frac{1}{\sqrt{2}}(\frac{-2+3}{\sqrt{2}}) \quad \frac{1}{2}(\frac{-2-3}{\sqrt{2}}) \]

\[ \frac{2}{2} \quad \frac{8}{2} \quad \frac{0}{2} \quad \frac{-4}{2} \]

\[ C^2 : \]

\[ \begin{array}{cccc}
1 & 1 & 1 & 0 \\
1 & 1 & 0 & -2
\end{array} \]

\[ f^2 \checkmark \]
DWT code snippet
- max() is similar to Sobel

```
pgm_dwt2D(pgm, level)

// Visualization
pgm_copy(pgm)
pgm_normalize(pgm)

// Output
pgm
// Optional: do something interesting with pgm
```

"e.g. "simulate compression"
\texttt{\texttt{Rm} \_\texttt{idwt} \_\texttt{2D} (\texttt{ging}, \texttt{level})} // inverse transform

// note: you could copy \texttt{ging},

do \texttt{MSE} — is there a diff

between \texttt{orig} = \texttt{idwt} (\texttt{dwt} (\texttt{orig}))

\texttt{MSE} = \frac{1}{mn} \sum_{i,j} (I_t(i,j) - I_n(i,j))^2

— per-pixel squared difference
\[ - \rho m \cdot dv \cdot dt = ( \rho m \times \hat{v} \cdot t, \text{cut level}) \]

\[ \lim_{t \to \infty} h_i = i \log^2 \]
\[ \text{int hi} = \left\lfloor \log_2 \left( \frac{\log(n)}{\log(2.0)} \right) \right\rfloor \]

\[ \text{int lo} = \text{hi} - \text{levels} \]
Int hi, lo are levels of the axi-pot image pyramid (DUT is a pyramidal image transform)

\[
\text{Int levels } = \emptyset \\
\text{means to do it once} \\
(1 - \text{level decom})
\]
$\text{WVLT} \times wp = \text{make-wavelet} (\text{HAAR})$

wavelet browser $\rightarrow$ Haar

$\text{Bardgo-wl}$

symlet

$\text{dwt 2D} (\text{iimg} \rightarrow \text{split}, \text{iimg} \rightarrow \text{revis})$

$iimg \rightarrow \text{hi}, \text{wp}, \text{hi}, \text{lo}$
```
dwt2D() ; i_dwt2D() function

dwt2D(double *I, int row, int col,
       WVLT *up, fulvl, cirlv)

<table>
<thead>
<tr>
<th>HcHR_A</th>
<th>HcGR_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>GcHR_A</td>
<td>GcGR_A</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>A_HRA</th>
<th>HcRA</th>
<th>HcGR_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>GcHR_A</td>
<td>GcRA</td>
<td>GcGR_A</td>
</tr>
</tbody>
</table>

(Recursively diagnose upper-left corner at each level)
```
source helper indices:

\[
\text{int } c, i, m, r, c, k
\]

\[
\text{int } \text{ran, col, ind, filter}
\]

\[
\text{int } \text{row, col, offsets (+p)}
\]

\[
\text{int } c2, j2, r2, c2 \text{ i/2, j/2, r/2, c/2}
\]

\[
\text{int } i2, j2 \text{ i/2, c/2}
\]
\( r = \text{zeros} \); \text{loff} = r \gg \geq 1 \);  \text{ll} \% \\
\( c = \text{cols} \); \text{coff} = c \gg \geq 1 \);  \text{ll} \% \\
\text{for } k = \text{full} \% \; k \geq \text{cols} \% \; k -- \\
\quad r_2 = r \gg 1 \);  \\
\quad c_2 = c \gg 1 \);  \\
\text{image} ( [ r_1, c_2 ] \text{ array set to } \emptyset \\
\text{image} ( [ r_1, c_2 ] \text{ array set to } \emptyset)
for $i = 0; i < r; i++$
\[ \text{pass} \]

for $j = 0; j < c; j++$

for $m = 0; m < ap -> len; m++$

\[ jj = j < < 1; \]
\[ H_{rA}(c, jj) = \leq g_{m}(c, jj + m) \]
\[ G_{A}(c, jj) = \leq g_{m}(c, jj + m) \]
(still in K loop)

\[ \text{pass 2} \]

for \( i = 0 \); \( i < k^2 \); \( i++ \)

\[ \text{for } j = 0 \; ; \; j < c^2 \; ; \; j++ \]

\[ I(i, j) = 0.0; \]
\[ I(i + r, j) = 0.0; \]
\[ I(i, j + c \times f) = 0.0; \]
\[ I(i + r, j + c \times f) = \n\heartsuit \heartsuit \]
- Bottom of main loop:
  \[ \text{free} \ (HrA) \]
  \[ \text{free} \ (G1A) \]
  \[ r \gg r = 1; \quad \text{col} \gg \ = 1; \]
  \[ \text{c} \gg = 1; \quad \text{col} \gg = 1; \]
}
- \text{idnt2DC()} I leave to you.
  it's similar, but you need to 
  (interleave row), cols in between 
  (extra shift)

- otherwise, \text{idnt2DC()} works backwards

- I still use \text{id(A,6vA)} two images
- I also use \text{RA, CA} as interface temp range
- for asg 03:
  - implement idw + r
  - show you can reconstruct image
  - optional: try MSE
- Moving on to OpenCV, video processing
  some image processing
- mainly in Python 3 (3.7)
  I have this installed
  numpy
  scipy (need XCode + Command Line Tools)
  OpenCV? (X11 Quartz)
- ffmpeg - command-line tool for transcoding video

- what is video?
  .mpg - MPEG (Motion Pictures Expert Group)
  .mp4 - MPEG-4, most popular today
  .avi - ?? Windows favorite
  .mov - Mac popular
- Movie file extensions are not super meaningful
- Because they're all just containers
- Usually it's codec (and inside that matters)

Decoder - [Codec]

H.264 is one of the most popular
- What matters are synchronization (frame rate), coders
  (frame rate, bit rate), codecs

- Audio stream - Video stream - Audio stream
  - subtitles stream

- Audio coder - Video coder
  - AVC, MPEG, AV1
- Subtitles: record video, speak slowly, clearly
  - Upload to YouTube
  - Get it to transcription
  - Remove original audio
  - Download subtitles (.srt file - credit)
- Verbose into video
-4.264 : still DCT-based (?)

DCT: Discrete Cosine Transform
(not quite Fourier \{ \cos \})

\[ \text{coD as the Gari function.} \]
- every I-frame fully encoded in "space"

- 2:19-2:19 pattern on blocks, i.e., 8x8
I-frame  I-frame

delta's I-frame detail
- Why the zig-zag pattern? DCT
- Because longer runs of zero coefficients

"DC"
lowest freq.

Run-length encoding

= 197, 10, (2x0), 23, 5, (10x0), end

End of block
- Firefox: install from firefox.org (or via repositories or apt-get)
- get ffmpeg, ffprobe
- lots of programs built on top of this: VLC, HandBrake
OpenCV:

- Install: source code probably best
- Want: contrib module (arrow lib)

- Latest version? 4.5.1?
  4.4.0
- Source code install:
  - see pyImageSearch

- On Mac, often need to do this in root (sudo) in /usr/local/src

- get opencv.tar.gz & contrib

- with cmake
- quntip, un tar, put into

/usr/local/src/opencv

- contrib go into /module/contrib

- mkdir build

- cd build

- cmake ..
asg 07: extra stuff:

- MSE
- wavelet denoising \rightarrow compression sim.
- edge detection via detection of modulus maxima

(in my notes: PDF)
Modulus maxima: wavelet coefficients
larger or equal to neighbour: (I)

\[ \text{mod}(x - \Delta x) \leq \text{mod}(x) \geq \text{mod}(x + \Delta x) \]

AND

\[
\begin{cases}
    \text{mod}(x) > \text{mod}(x + \Delta x) \\
    \text{mod}(x) > \text{mod}(x - \Delta x)
\end{cases}
\]

\[ x - \Delta x \quad x \quad x + \Delta x \]
$y(x)$ is greater than or equal to both of its 1D neighbors, and strictly greater than one of them.

In 2D:

\[
\begin{array}{c|c|c}
1 & 2 & 3 \\
\hline
2 & 3 & 1 \\
\hline
3 & 1 & 2
\end{array}
\quad \text{or} \quad
\begin{array}{c|c|c}
6 & 7 & - \\
\hline
8 & - & 6 \\
\hline
7 & 6 & -
\end{array}
\]
- can also get \( \Theta : \text{atom}^2(dx, dy) \)
- get another interesting
- anisotropic filtering
- not the same everywhere
- nice sharp

\[ e^{-\left(\frac{x^2 + y^2}{2}\right)} \]
- back to OpenCV (with Python)
- once you have OpenCV & python libs installed:

```python
import sys, math
import numpy as np
import cv2
```

img.jpg
Quick example: display image

```python
img = cv2.imread(argv[1], 0)
cv2.imshow('image', img)
k = cv2.waitKey(0)
if k == 27:
    cv2.destroyAllWindows()
```
- Next thing: display video:

```python
import cv2

# Capture from laptop camera
cap = cv2.VideoCapture(0)

# Or from file / e.g. file.mp4

# Use cap, interesting stuff:
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
```
$$fps = \text{cap}.\text{get}([\text{cv2}.\text{CAP_PROP_FPS}])$$

(might not be correct)

$$\text{frame\_rate} = 1 / fps$$

$$\text{frames} = \text{cap}.\text{get}([\text{cv2}.\text{CAP_PROP_FRAME_COUNT}])$$

$$\text{total\_time} = \text{frame\_rate} \times \text{frames}$$

sampled at 120 Hz

80 fps (80 Hz)
- I've done this before.

\[
dilation = \frac{\text{fps}}{\text{data\_duration()} \cdot (\text{frame\_rate} \times \text{frames})} \\
\hspace{1cm} \text{video duration}
\]
- display video: need frame
  
  if cap.isOpened():
    ret, frame = cap.read()
  
  if ret is True:
    frame = cv2.resize(frame, size, 0.0, cv2.INTER_CUBIC)
    grey = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('vid', grey)
ms = cap.get(cv2.CAP_PROP_POS_MSEC) * 1000 / 1000

for timestamp at bottom left
asg04

- video copy assignment using OpenCV
- Python3 (C++ ok)
  email me

- Choose A VIDEO — short, not too big dim (640 x 480 --)
- To output video:

```python
import cv2

cap = cv2.VideoCapture('input')
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
if output:
  fourcc = cv2.VideoWriter_fourcc(*'mp4v')
  fps = 25.0  # Why did I do this?
  # should be same as input
  out = cv2.VideoWriter('output.m4v',
                        fourcc, fps, size, True)
```
while (cap.isOpened()) {
    ret, frame = cap.read()
    if ret == True:
        # do stuff
        if output:
            out.write(frame)
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
}
# end of while

cap. release()

if output:
    out. release()

cv2. destroyAllWindows()
(as good - face detection) ( & tracking)

"face is there"

"history"

"frame coherence"

Simple approach + open CV

(5) zoom filter

→ remembering that we had face on previous frames
Face tracking: use object for face
and "history" (deque)
normal face? (like ring buffers)

Adaptive:
- you decide how much "face" to store

New frame?
new face detected
likely face
not face
- asgør:

  a) Face detection

  b) Facial features

  Individual detectors for each feature

  Haar cascading filter
- Haar cascading filter
- Viola-Jones 2001 paper
- machine learning approach
  - trains detectors for face, eyes, nose, mouth
  - xml file in OpenCV
  - dist
- Viola-Jones
- 3 advancements
  1. use an integral image
     allow for fast computation
  2. learning algorithm based on AdaBoost
     selects small no. of features
  3. combination of classifiers into a cascade
1. Integral image: based on Haar basis function.
   - Computed at many scales.
   - Sound familiar?
   - Haar cascade filters.
   - Haar basis features - not prices.
- 3 kinds of features
  - 2 rect
  - 3 rect
  - 4 rect

- Detector is 24 x 24
- Unlike wavelet haar bases, these overlap
  - Overcomplete
- Compute Haar-like sums, or integral image:

$$\tilde{i}(x, y) = \sum \tilde{e}(x', y'), x' \leq x, y' \leq y$$

- Similar to summed area tables in graphics.
2. Learning classification functions
   - need a feature set that, we family
   - a weak classifier is used

   \[ h_j(x) = \begin{cases} 1 & \text{if } f_j(x)(t) < \theta_j \\ 0 & \text{otherwise} \end{cases} \]

   - need AdaBoost to learn (classifier)
3. **Attentional cascade**
   - cascade is a decision tree
   - cascaded triggering of classifier

![Diagram of a decision tree cascade with decision points labeled 1, 2, 3, and a 'reject' decision node.]
- so where are those cascaded filters?

/usr/local/src/opencv/data/

haarcascades/*.xml

- for eyes - use Haar cascades for face detector - you get a lot
- then, within this box, use cascaded further for eyes, mouth, nose
1. [Smiley face drawing]

2. [Drawing of a robot]

You will get a bundle of false tetris.

Expect a list of rectangles for each feature.
- code looks like this:

```python
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')

left_cascade = cv2.CascadeClassifier('haarcascade_lefteye_2splits.xml')

right_cascade =

nose_cascade =
mouth_cascade =
```
- Then use classifier

\[ \text{faces} = \text{face cascade, detect Multi Scale} \text{ (frame, } 1:3, 5) \]

I forgot what these are

for \((x, y, w, h)\) in faces:

\[ \text{roi - gray} = \text{gray}[y:y+h, x:x+w] \]

... search space

\[ \text{eye = eye cascade, detect Multi Scale} \text{ (roi - gray, ...)} \]
... same for eye, nose, mouth

- you set lots of eye, nose, mouth keyers.

- this is done per frame

(The detection part of ours)