POLITECNICO
MILANO 1863

IMAGE AND SOUND
ISPG
PROCESSING GROUP

CREATIVE PROGRAMMING AND COMPUTING

Lab: Tools & the World

WHO I AM

- Assistant: **Michele Buccoli**
  - E-mail: michele.buccoli@polimi.it

2013 **M. Sc.**

- *A music search engine based on semantic text-based queries*

2013 – 2016 **PhD**

- *Linking signal and semantic representation of musical content for Music Information Retrieval*

2016 – 2018 **Post-doc researcher**

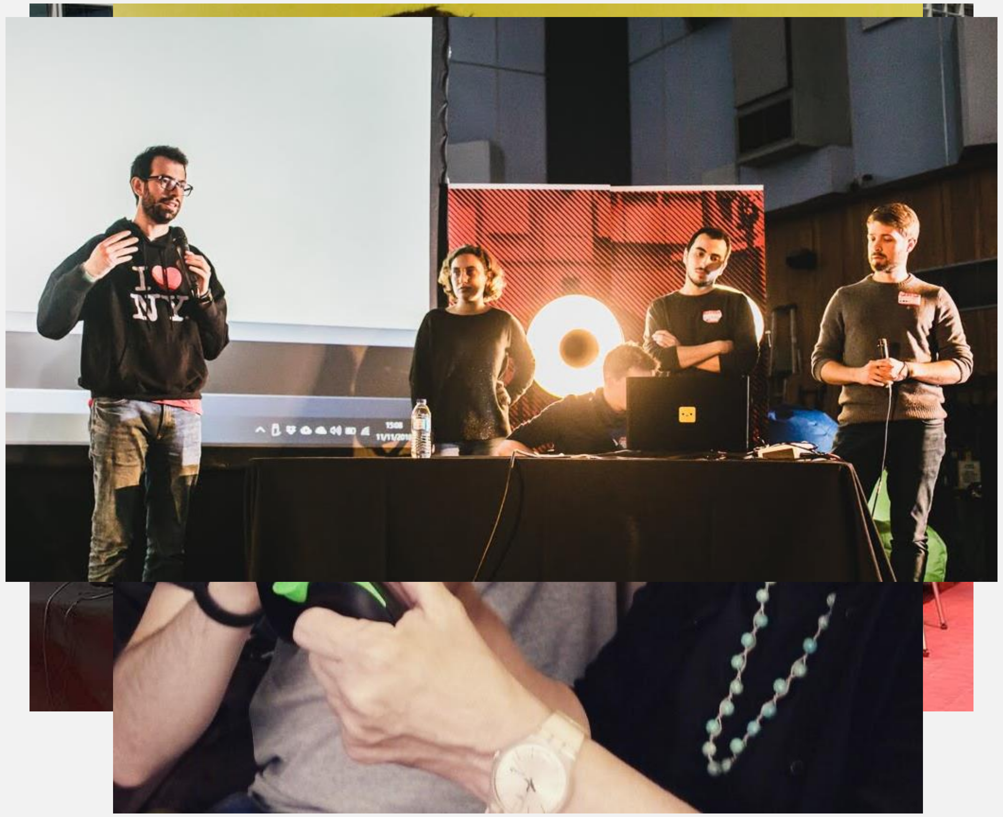
- Researcher for the WholoDance European project

2019 **Researcher for BdSound S.r.l.**

- Audio solutions based on machine and deep learning

WHY I AM HERE

- Assistant: **Michele Buccoli**
  - E-mail: michele.buccoli@polimi.it



2013 – 2016 **PhD**

- Present lab's activities: Music Information Retrieval is easier to present than heavy-math space-time processing

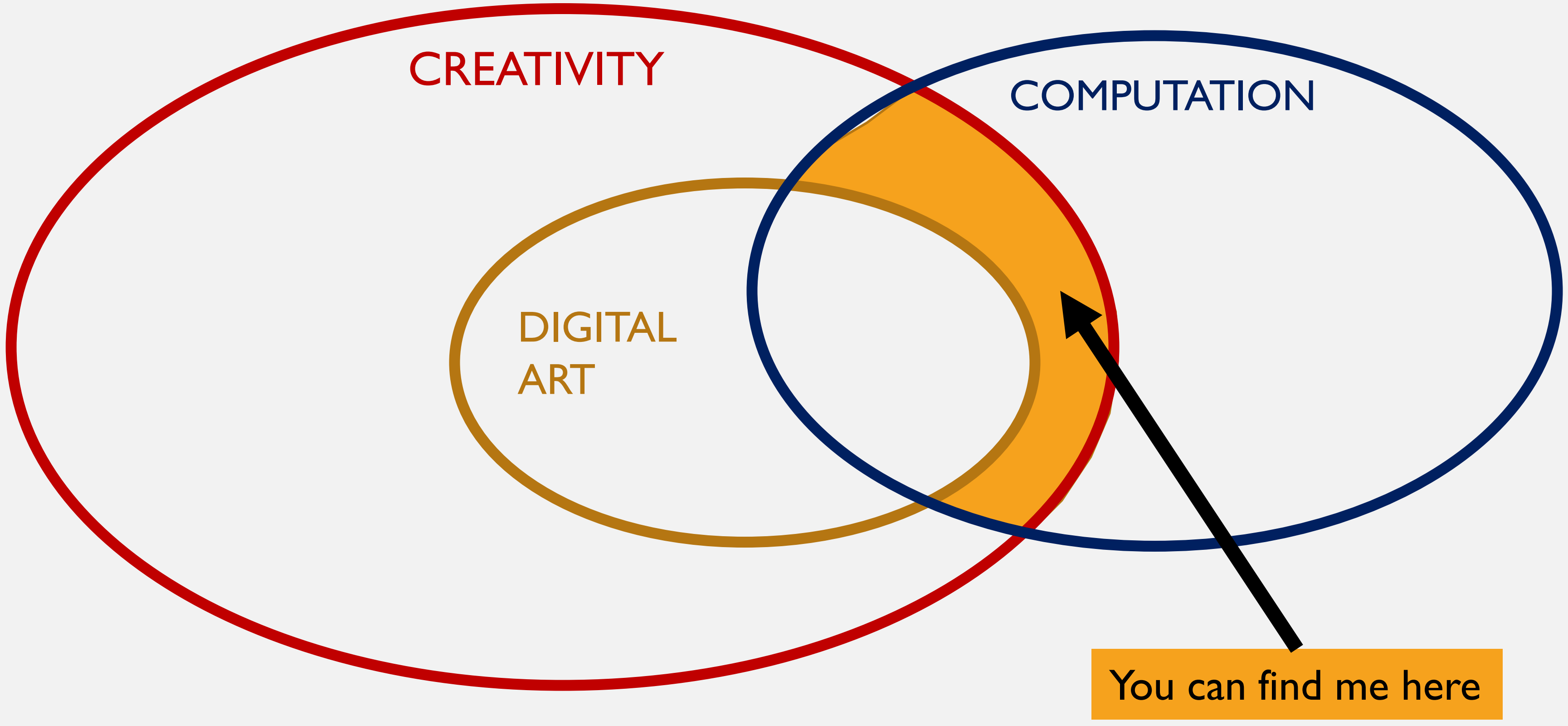
2016 – 2018 **Post-doc researcher**

- Presenting new work with demos every six months (at least), also using VR technology

2016 – present **hackathon enthusiast**

- 2016 @ Spotify NYC
- 2017 @ Waves Vienna (with Comanducci, Borrelli, etc.)
- 2018 @ Abbey Road Studios (with Borrelli)
- 2020 Crispy Bacon #hackathome (during lockdown)

ART, CREATIVITY AND COMPUTING



SLACK

- Prof. Zanoni invited you to a slack cpac2020
- Slack is used by many companies to communicate
  and manage their projects
- We encourage you to enroll and actively participate to the channels, posting about seminars, activities and cool demos you find around
- And there is a #helpme channel for debugging

GITHUB

- Code is released on the Github repository
- Learning to use git is **crucial** to develop projects with other people and keep track of your code
- Start creating your Github portfolio of projects! Recruiters love github profiles with cool (and well described) projects
- Your project will be on Github too

https://github.com/mae-creative-pc/cpac_course_2020

HOW LABS WORK

- I will briefly introduce tools and code simple ideas to leverage this tool
- I will give you the skeleton, because starting from a blank screen takes TIME
- After some bricks are introduced, I will ask you to try to connect them and make the idea more complex and interesting following your creativity and inclination
  - I'm not a good sound designer, but maybe you are, and can make interesting sounds out of these examples

- There is not enough time to explain to you in detail what every function does, so try to catch the idea and RTFM («read the *friendly* manual»),
  - i.e., refer to the documentation
- When coding your idea, it is very common to learn new tools from tutorial and examples

TODAY

- Music features
  - Low-level computation with Processing
  - Mid-level extraction with Sonic Annotator (and their use with Python)
  - High-level extraction with Python (and Spotify!)

- Video features with Processing

AUDIO, MUSIC AND FEATURES

Hand-crafted audio features

Mid-level features

Human-readable descriptors

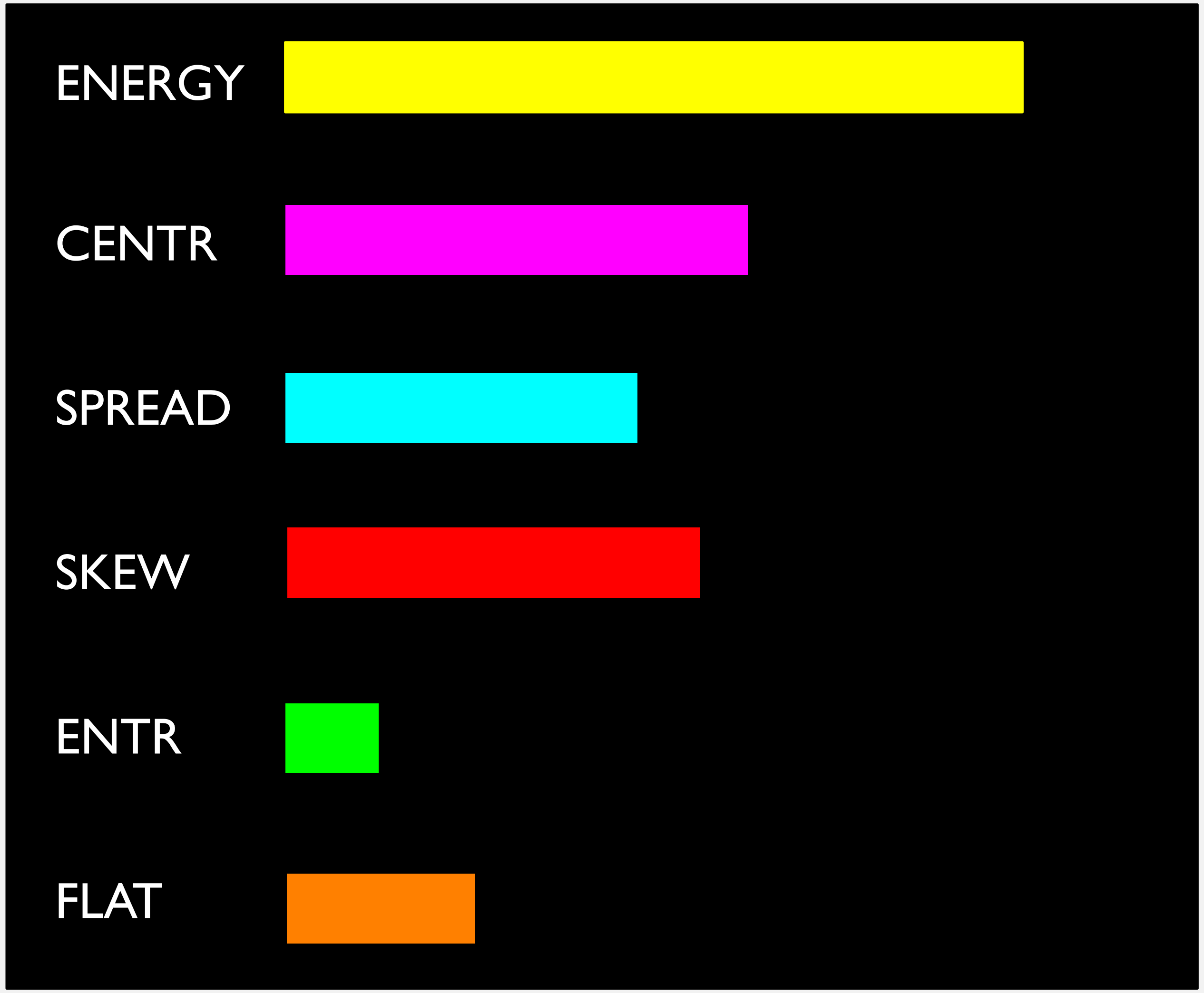
HAND-CRAFTED AUDIO FEATURES

- Information from audio are called *features* and divided into three levels:
  - Low-level features describe the *signal*: computed with math formulas and named therefore *hand-crafted*
  - Mid-level features describe the *music*, so we need to include information on how music works
  - High-level features describe the *semantics*: how normal people talk about music.

- Let's start from the former and design a *feature* visualiser

- First: install Minim and Sound libraries (audio analysis and synthesis)
  - go to Processing → Tools → Add Tool → Libraries
  - Search and install Minim and Sound

HAND-CRAFTED AUDIO FEATURES



- Feature visualizer:
  - Takes a song and extract the Fourier Transform
  - Uses such information to compute six low-level features, namely: Energy, Spectral Centroid, Spectral Spreadness, Spectral Skewness, Spectral Entropy and Spectral Flatness
  - Visualizes the values of such features as width of a set of bars

HAND-CRAFTED AUDIO FEATURES

- See the code in the repository, you will find:
  - A processing_ft_visualiser: the main script for processing
  - An Agent file which implements two classes: AgentFeature extracts features, AgentDrawer draws them

- The main script has the following
  - In the setup function it starts the screen and initialize two objects: one AgentFeature and one AgentDrawer (the AgentDrawer can see the AgentFeature
  - For each frame, the *draw* function calls the method *reasoning* of AgentFeature to compute the features and the action *draw* of the AgentDrawer to compute them

HAND-CRAFTED AUDIO FEATURES

- I prepared the AgentDrawer for you, no need to worry too much
  - If you read the code you can understand some of what it does, but you can also ask me if something is not clear
- The AgentFeature takes the fft.forward to compute the spectrum as an object this.fft
  - this.fft.specSize() → size of the spectrum; this.fft.getBand(i) → value of the i-th band
  - this.freqs[i]→ frequency in Hz of the i-th band; this.sum_of_bands → sum of all the bands
- Features are computed as
  - float centroid = compute_centroid(this.fft,sum_of_bands,this.freqs);
  - float flatness = compute_flatness(this.fft, sum_of_bands);
  - float spread = compute_spread(this.fft, centroid, sum_of_bands, this.freqs);
  - float skewness= compute_skewness(this.fft, centroid, spread, this.freqs);
  - float entropy = compute_entropy(this.fft);
  - float energy = compute_energy(this.fft);

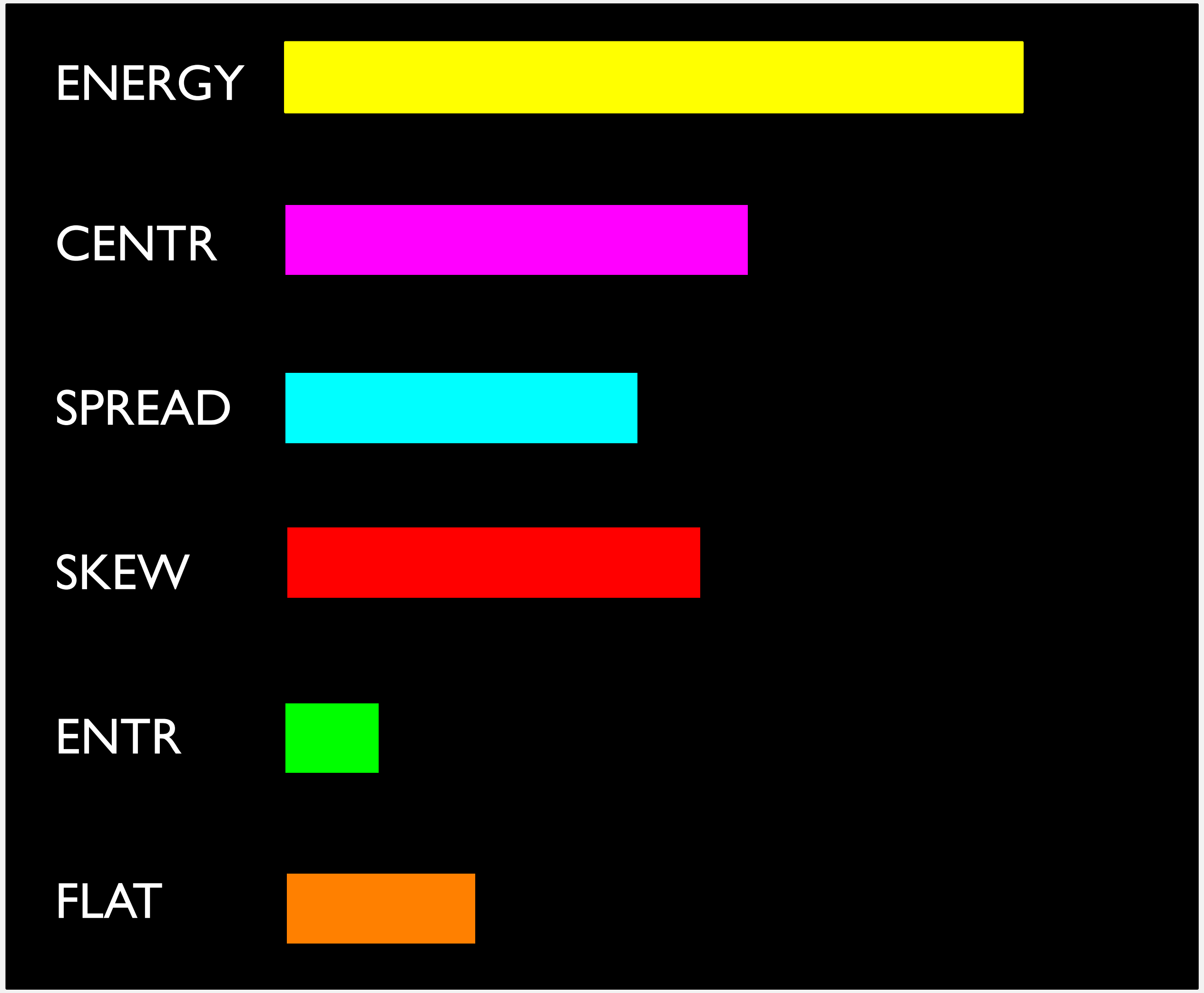
HAND-CRAFTED AUDIO FEATURES

Formulas $|X(k)|$= this.fft.get_band(k) ; $f(k)$=this.freqs[i]

- Spectral centroid: $S_c = \frac{\sum_{k=0}^{K-1} f(k)|X(k)|}{\sum_{k=0}^{K-1}|X(k)|}$ is the "center of gravity" of the spectrum
- Spectral spread: $S_{sp} = \frac{\sum_{k=0}^{K-1}(f(k)-S_c)^2|X(k)|}{\sum_{k=0}^{K-1}|X(k)|}$ measures "standard deviation" of the spectrum
- Spectral skewness $S_{sk} = \frac{\sum_{k=0}^{K-1}(f(k)-S_c)^3|X(k)|}{K\, S_{sp}^3}$ symmetry of the spectrum around the centroid
- Spectral Entropy $S_{sp} = -\frac{\sum_{k=0}^{K-1}|X(k)|\cdot\log(|X(k)|)}{\log K}$ amount of information in the spectrum
- Spectral Flatness $\mathrm{S_f} = K\,\frac{\sqrt[K]{\prod_{k=0}^{K-1}|X(k)|}}{\sum_{k=0}^{K-1}|X(k)|}$ degree of flatness in the spectrum

Michele Buccoli, *Linking Signal and Semantic representations of musical content for Music Information Retrieval*, PhD thesis

HAND-CRAFTED AUDIO FEATURES



Coding time

- Open the Agent.pde file with Processing
- Replace the functions *compute_centroid* … with actual code to compute such metrics using formulas from previous slide
  - Hint: every time there is a division, avoid possible divisions by 0 replacing x/y with x/(0.000001+y)
- Watch the result

HAND-CRAFTED AUDIO FEATURES

Advanced coding

- You may see the bars move too quickly due to abrupt changes
- How can we address this?

output signal

Input signal

To smooth a signal we need a low-pass filter $y(n) = \mathcal{F}\{x(n)\}$

- Finite Impulse Response (FIR): buffer the last N samples and apply a weighted average
  - $y(n) = \sum_{l=0}^{N-1} b_l x(n-l)$
  - Problem: we need to buffer the past N samples
- Infinite Impulse Response (IIR): also use the past samples:
  - $y(n) = \sum_{l=0}^{N_b-1} b_l x(n-l) - \sum_{l=1}^{N_a-1} a_l y(n-l)$
  - Advantage: you can obtain the same result of a FIR with much less weights, i.e., saving less past sampels

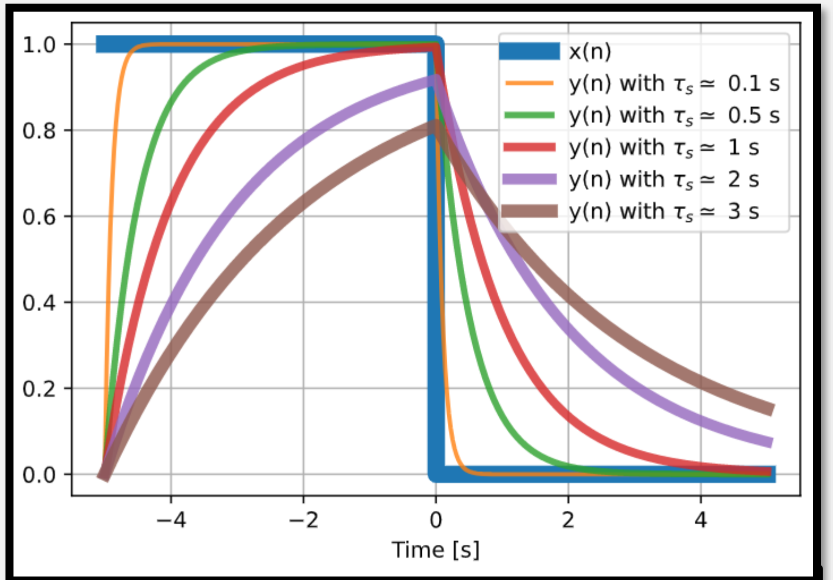
HAND-CRAFTED AUDIO FEATURES

First-order filter:

- $y(n) = \alpha x(n) + (1-\alpha)y(n-1) = y(n-1) + \alpha\big(x(n) - y(n-1)\big)$
- Using the *forget factor* $\alpha = 1 - e^{-1/\tau_s f_s}$ where τ_s is the time constant in seconds and f_s is the sampling frequency
  - After τ_s
- Or, as a function of the desired cutoff frequency f_c → $\alpha = 1 - e^{-2\pi f_c/f_s}$

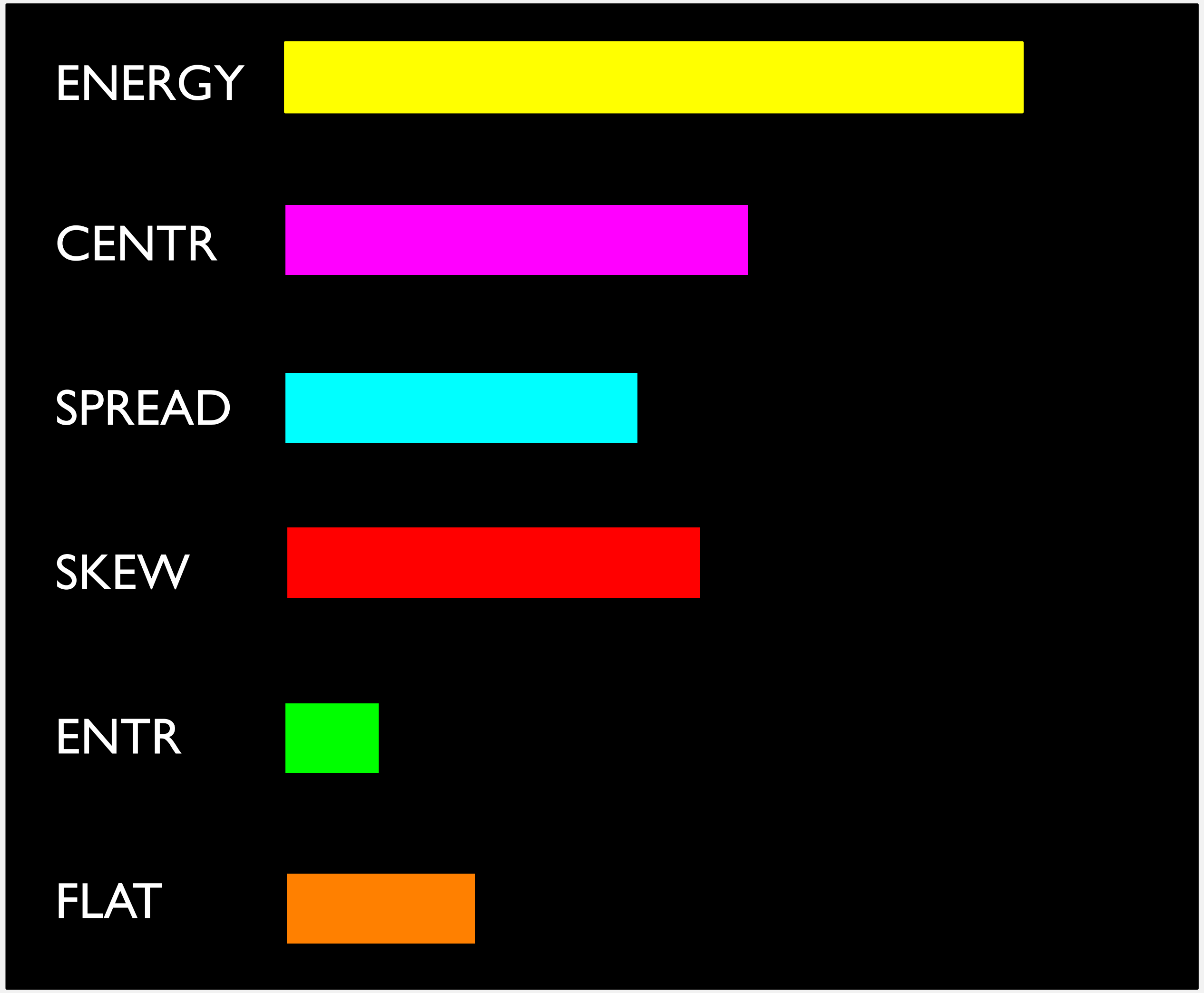
- Example
  - Suppose we want to smooth the signal so it takes 50 ms to change status
  - We know Processing works at 60 Hz
  - Our alpha will be $\alpha = 1 - e^{-1/0.05\cdot 60}$

- Homework (if you want)
  - Initialize this.lambda_smooth in the AgentFeature function
  - Implement the function smooth_filter in AgentFeature
  - See the result



Note however that after τ_s you reach 0.66% of the actual result, so you'd better choose is slightly lower than you need

HAND-CRAFTED AUDIO FEATURES



It is clear this visualizer has no artistic purposes

However, at some point you may want to make an animation that reacts according to music.

Most people use just energy or spectrum bands.

Now you know how to extract other properties in real time and can use them for your animations or other scopes.

MID-LEVEL FEATURES

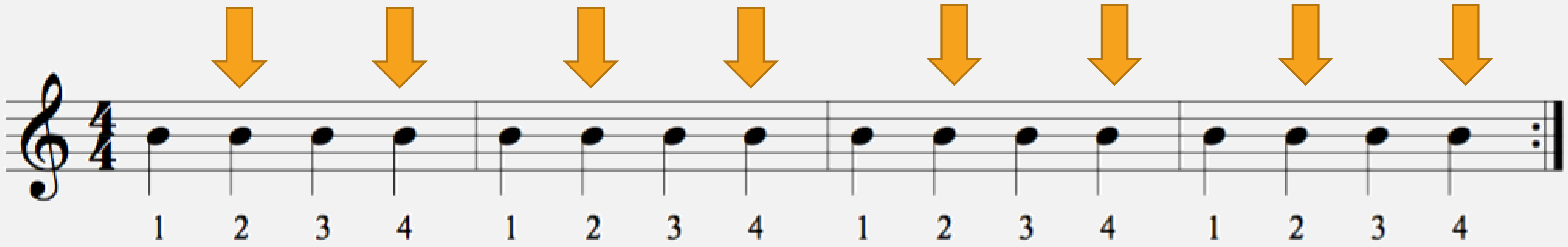
- Mid-level features are related with musical properties of audio
  - Tempo, beats, onset
  - Harmony, chords, key
  - Melody, pitch, etc.
- They are hard to compute them by yourself, but luckily there are models to compute them around
  - Some systems are even better than non-trained humans
- Let’s see how to do «artistic» things with it

EXTRACTING FEATURES WITH PYTHON

- Have you ever watched this video?

https://www.youtube.com/watch?v=0wZgEYPKKfs&feature=youtu.be&t=170

- JB tells the audience to clap *on the right notes*, which, in a typical 4/4 tempo are the second and the fourth beats



EXTRACTING FEATURES WITH PYTHON

- However, it is not easy for non-musician to follow them.
- Let’s make a tool to help them

What do we need to do?

1. To know when the beats are
2. To know which beats are the second and the fourth
3. Put a sound on them (e.g., *a clap*)
4. Write the final song with the correct claps

EXTRACTING FEATURES WITH PYTHON

Before starting, install the environments if you haven't yet

- At https://docs.google.com/document/d/1GxJ8pgdAnah1K2ExN9J4gN-SvlEsH0420-Y75BbFTi4/edit you can find the instructions to set up an environment like we will use in the labs
- Download the example file from https://drive.google.com/file/d/1XmXdODdsJUOC2C6znokr8wQnXaN3V9hY/view?usp=sharing

You are free to use your own, but this is the official documentation that I will keep updated

EXTRACTING FEATURES WITH PYTHON

1) To know when the beats are

We use *Librosa* to extract the information

- `bpm, beats=your_code.compute_beats(y, sr=SR)`
  - Find the function in librosa which actually extract the beats
  - Use it in your_code.compute_beats

EXTRACTING FEATURES WITH PYTHON

2) To know which beats are the second and the fourth

```
beat0=your_code.first_beat(y, sample_beats)
```

- Let’s assume we have a 4/4 and that the first beat will have more energy
- We first put the beats in a N x 4 matrix and see which is the first beat of a bar and the corresponding energy
  - $\boldsymbol{B} = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & \ddots & \ddots & b_7 \\ \vdots & \ddots & \ddots & \vdots \\ b_{N-3} & b_{N-2} & b_{N-1} & b_N \end{bmatrix}; \boldsymbol{Y} = y(\boldsymbol{B})^2 = \begin{bmatrix} y(b_0)^2 & y(b_1)^2 & y(b_2)^2 & y(b_3)^2 \\ y(b_4)^2 & \ddots & \ddots & yb_7{}^2 \\ \vdots & \ddots & \ddots & \vdots \\ y(b_{N-3})^2 & y(b_{N-2})^2 & y(b_{N-1})^2 & y(b_N)^2 \end{bmatrix}$
- Find the average for each of the four columns: $\bar{y}(\boldsymbol{B})^2$
- The index with the highest mean is the first beat i_b

EXTRACTING FEATURES WITH PYTHON

3) Put a sound on them (e.g., *a clap*)

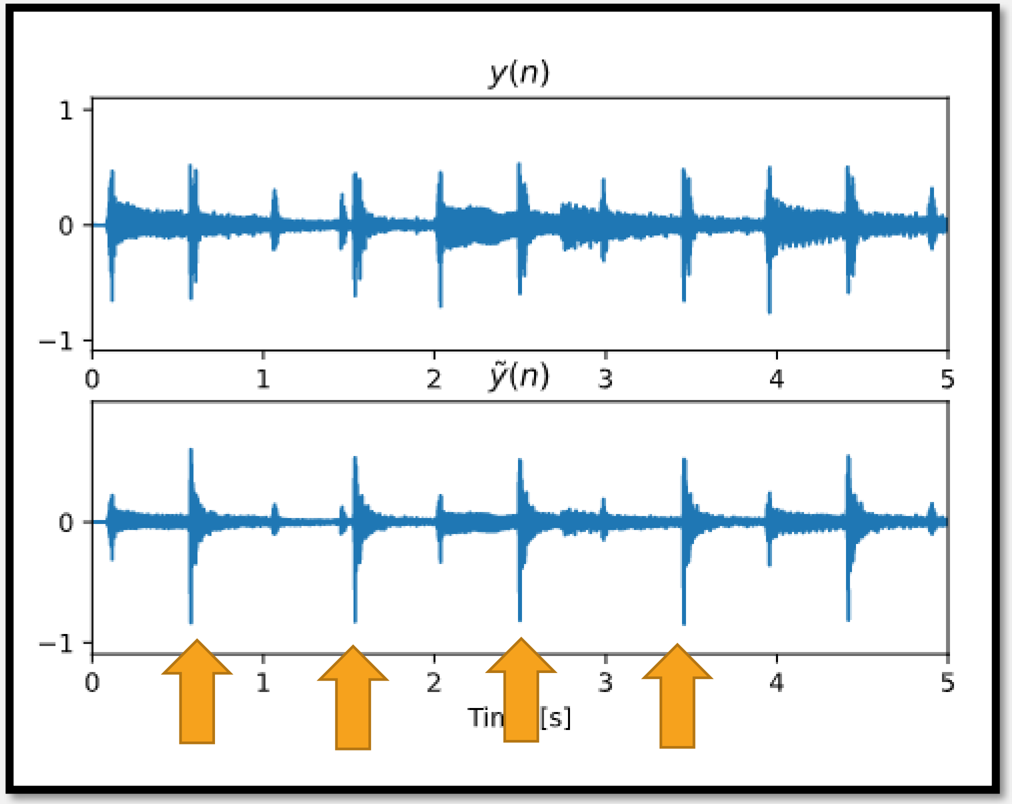
Load the clap $c(n)$ and the index of maximum energy (the onset) i_c and size N_c

```
y_out = your_code.add_claps(y, sample_beats, i_b, clap, i_c, N_c)
```

- Create a copy of the input signal $\tilde{y}(n) = y(n)$
- Beats for the clap are $\boldsymbol{b}_c = \left[b_{i_b+1}, b_{i_b+3}, \ldots\right]$
  - Starting from $i_b + 1$, i.e., the second beat, and then go on every 2 beats
  - `b_c=sample_beats[i_b+1::2]`
- For each new beat $\tilde{b} \in [b_{i_b+1}, b_{i_b+3}, \ldots]$
  - $\tilde{y}(\tilde{b} - i_c + n) = y(\tilde{b} - i_c + n) + c(n)\ \forall n = 1, \ldots, N_c$
- Normalize the signal $\tilde{y}(n)$ by the maximum absolute value

4) Write the final song with the correct claps

```
Import soundfile as sf
sf.write(filename_out, y_out, SR)
```



HUMAN READABLE DESCRIPTORS

- As we see low-level features describe the *signal* and mid-level features describe the *musical properties*.
- People use different ways to describe music, in a semantic way, e.g., *catchy*, *depressing*, *great do dance*, *acoustic*, etc.
- It’s not easy to extract such information and we would need machine learning techniques which are expensive
- But there are easier way to do it

HUMAN READABLE DESCRIPTORS

- In 2014 Spotify acquired *EchoNest*, a company devoted to extract information from music and it still uses their technology to extract information from their whole database
- Spotify also offers developers like you and me the possibility to access such information through APIs, i.e., Application Program Interface
  - Basically, a set of websites you can use from your own Python to download information
- Reference is at https://developer.spotify.com/documentation/web-api/reference/
  - But we will need a «token», i.e., a string that tells Spotify who is asking for data
  - Everything you do is tracked to you, so be careful and don't ask for one million song at once
- Let's see preliminary_operations.py

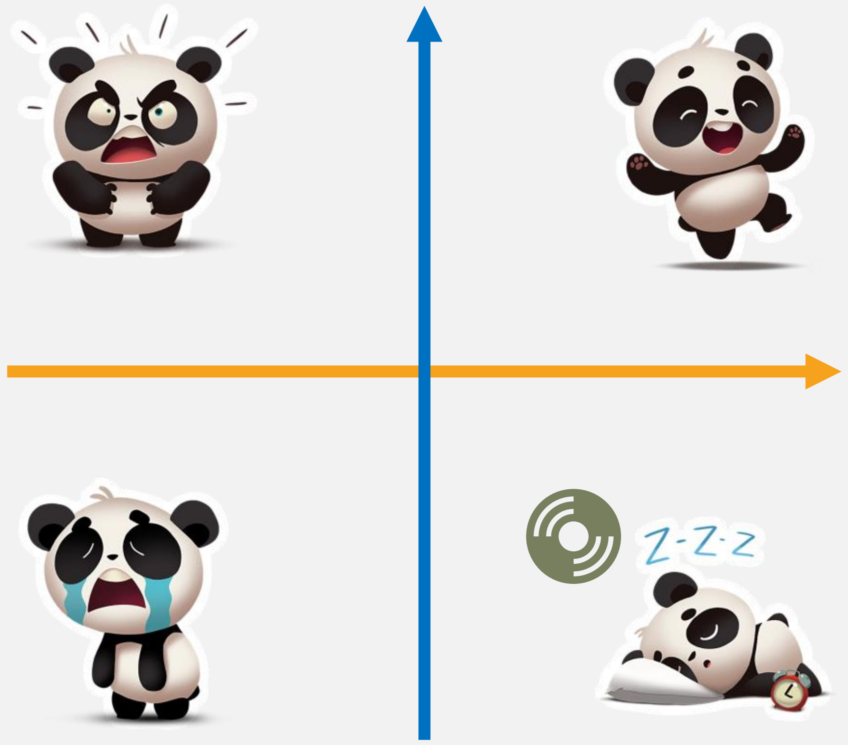
HUMAN READABLE DESCRIPTORS

Audio Features computed from Stand by Me by Ben E. King

Meaning: https://developer.spotify.com/documentation/web-api/reference/object-model/#audio-features-object

- Duration: 180.056 seconds
- BPM: 118
- Key: A-major
- The danceability of the song is 65 %
- The energy of the song is 31 %
- The speechiness of the song is 4 %
- The acousticness of the song is 57 %
- The liveness of the song is 7 %
- The instrumentalness of the song is 0 %
- The valence of the song is 60 %

Valence and Energy (also named *Arousal*) can be used to get the *mood* of the song



HUMAN READABLE DESCRIPTORS

- Now let's create a *personalized* playlist
  - This will REALLY create a playlist on your personal Spotify account. Bear with me
  - If you don't want to, set the flag `CREATE_SPOTIFY_PLAYLIST = False`
- Scenario: collaborative playlist
  - I invite friends home for a party
  - I ask them to suggest two/three songs each
  - I want to create a good path among them
    - Suggested readings: *High Fidelity* by Nick Hornby
- Finally, we create the playlist (for real)

We will exploit Spotify's API

HUMAN READABLE DESCRIPTORS

See the code on playlist.py

Structure of the code

1. Get the token from https://developer.spotify.com/console/get-audio-analysis-track/
2. Open the file list_of_songs.json where I saved the songs you suggested
3. Download the audio features for each song
4. Ok, now you have a list `audio_features`, where each element contains the features I just asked to Spotify. **Now it's your turn!**
5. Implement the function `shuffled_songs= your_code.sort_songs(audio_features)` following any criterion of your choice. For example (feel free to ignore me)
   - Danceability grows constantly, but a few of the least-danceable songs are at the end (when you want people to go home)
   - You compute mood (as the angle in the VA space) and sort songs by mood
6. Create a playlist with the songs sorted by your choice

HUMAN READABLE DESCRIPTORS

Now what?

- Using Spotify you can extract reliable human-readable descriptors for most of the songs that are around
- There are plenty of APIs around that you can use for your artistic projects
  - E.g., detect the mood from your face https://azure.microsoft.com/en-us/services/cognitive-services/face/
- you can use integrate external tools and benefit from big companies
  instead of having to deploy your own solutions

VIDEO FEATURES

Accessing the camera

Change colors

Difference of frames

Optical flow

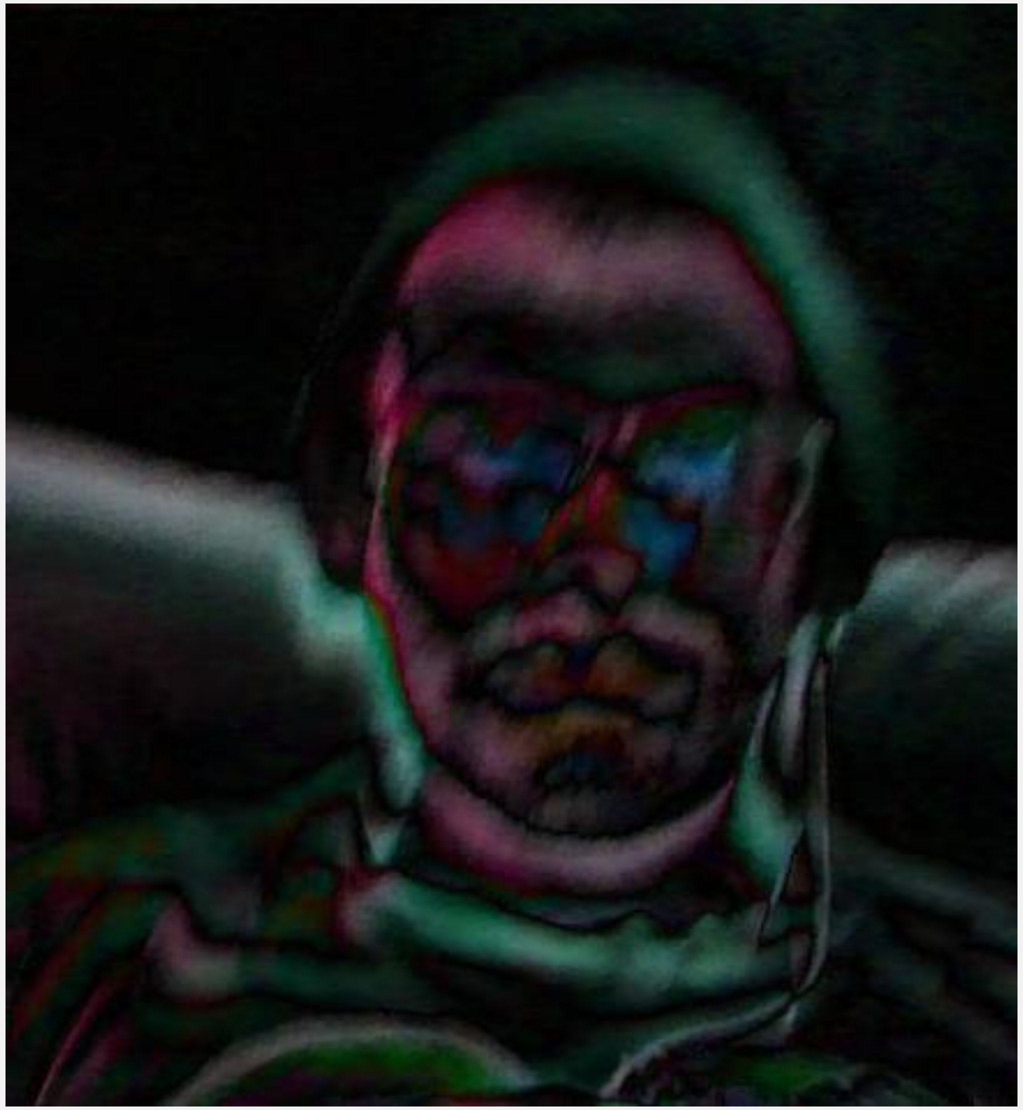


IMAGE PROCESSING

- Information extracted from images follow a taxonomy similar to audio features
  - Low-level features describe the *signal*: computed with math formulas
  - Mid-level features describe something happening in the screen with models
  - High-level features describe what is in an image or what it represents

- The nice thing about image feature is that it is much easier to view them
- More specifically: we can view them *on top* of the image from which we are extracting them
- And hence we can create interesting image effects

IMAGE PROCESSING

We will use Processing for this part

- First install the opencv and Video Libraries
  - Install libraries via Tools → AddTools →Library
  - look for *OpenCV for Processing* and *Video*

Very first example: how to get the webcam

- Let's run this piece of script first
- Keep changing C (starting from 0) until you get something
- change W and H (width and height) of the screen accordingly

```
# processing_video_fts.pde

import processing.video.*;
Capture cam;
void setup(){
  size(W, H); // size of the webcam you choose
  String[] cams = Capture.list();
  if (cams.length == 0) {
    println("No cameras =( ");
    exit();}

  println("Available cameras:");
  for (int i = 0; i < cams.length; i++) {
    println(cams[i]);}
  cam = new Capture(this, cameras[C]);
  cam.start();
}
void draw() {
  if (!cam.available()) {return;}
  cam.read();
  if(cam.width>0){ image(cam, 0, 0);}
}
```

IMAGE PROCESSING

Issue with macOS

Unfortunately, macOS is having troubles accessing the camera. A workaround is available but it is not straightforward. https://github.com/processing/processing-video/issues/134#issuecomment-617301980

Otherwise, I suggest you to use the processing_video_fts_mov.pde file:

- Use a sample file or use https://drive.google.com/file/d/1F2plrz0jfmjIT TVhP66BHdb7Fw961S1E/view?usp=sharing
- Place it in the data folder inside processing_video_fts_mov folder

```
# processing_video_fts_mov.pde

import processing.video.*;

Movie cam; // instead of a camera, we use a
movie

void setup() {
  size(640, 480);
  cam = new Movie(this, "./sample.mp4");
  cam.loop();
}
void draw() {
  if (!cam.available()) {return;}
  cam.read();
  if(cam.width>0){ image(cam, 0, 0);}
}
```

IMAGE PROCESSING

To get the pixels from the cam:

- cam.pixels is an array of integers, representing pixels;
  - cam.pixels[0] is the top-left corner, pixels[W*H-1] is the bottom-right corner
- You can navigate pixels as index in array or via row/columns of the image:

```
for (int x = 0; x < cam.width; x++) {
  for (int y = 0; y < cam.height; y++) {
    int loc = x + y * cam.width;
    ...
  }
}
```

```
for (int i=0; x<cam.width*cam.height; i++) {
  x= i% cam.width;
  y= i/cam.width;
  ...
}
```

IMAGE PROCESSING

Each pixel can be decomposed as

- R, G and B components,
- hue, brightness, saturation
- Alpha is the transparency

```
for (int i=0; x<cam.width*cam.height; i++) {
  int r= red(cam.pixels[i]);
  int g= green(cam.pixels[i]);
  int b= blue(cam.pixels[i]);
  int alpha= alpha(cam.pixels[i]);
  int h= hue(cam.pixels[i]);
  int br= brightness(cam.pixels[i]);
  int sat= saturation(cam.pixels[i]);
}
```

A new color can be created:

- color(gray): 0 is black, 1 is white
- color(v1, v2, v3)
  - Depends on colorMode (see right)
- color(gray, alpha) and color(v1, v2, v3, alpha)
  - alpha controls the transparancy

```
colorMode(HSB, 255);
color hsb_color=color(h, sat, br);
colorMode(RGB, 255);
color rgb_color=color(r, g, b);
```

IMAGE PROCESSING

- However you cannot modify the pixels of a camera, so instead we can just copy the camera in an image and modify it
  - Create an image with proper size and imagemode
  - Use loadPixels() to access img pixel
  - Use updatePixels() to store the modified pixels
  - We wrap it in a copy2img function
- See also the documentation of Pimage
  - https://processing.org/reference/PImage.html

```
void copy2img(Capture cam, Pimage img){
 img.loadPixels();
 for(int i=0; i<w*h; i++){
    img.pixels[i]=cam.pixels[i];}
 img.updatePixels();
}

int w= cam.width; int h=cam.height;
PImage img=createImage(w,h,RGB);
 copy2img(cam, img);
```

IMAGE PROCESSING

- However you cannot modify the pixels of a camera, so instead we can just copy the camera in an image and modify it
  - Create an image with proper size and imagemode
  - Use loadPixels() to access img pixel
  - Use updatePixels() to store the modified pixels
  - We wrap it in a copy2img function
- See also the documentation of Pimage
  - https://processing.org/reference/PImage.html

```
void copy2img(Movie cam, Pimage img){
 img.loadPixels();
 for(int i=0; i<w*h; i++){
    img.pixels[i]=cam.pixels[i];}
 img.updatePixels();
}

int w= cam.width; int h=cam.height;
PImage img=createImage(w,h,RGB);
 copy2img(cam, img);
```

Use Movie cam if you are using a video file

IMAGE PROCESSING

First exercize: switch colors!

- See the skeleton of the code (you have to modify it to make it work)
  - First start with the no effect to test your cam is working
  - Then write a function switch colors that changes the camera image so that the colors are changed somehow
  - how: it is up to you

```
import processing.video.*;
Capture cam;

/* CODES for EFFECTS (more later)*/
int NO_EFFECT=0; int EFFECT_SWITCH_COLORS=1;
int effect=EFFECT_SWITCH_COLOR;

void setup() { /* as above*/}
void copy2img(Capture camera, PImage img) {/*as above*/}
void changeColors(PImage img){/* your code */}
void draw() {
  if (! cam.available()) {cam.read();}
  cam.read();
  PImage img=createImage(cam.width,cam.height,RGB);
  copy2img(cam, img);
  if(effect==NO_EFFECT){;}
  else if(effect==EFFECT_SWITCH_COLORS){/*your code!*/}
 /* else if(effect==OTHER EFFECT){ etc...}*/
  if(img.width>0){image(img, 0, 0);}
}
```

IMAGE PROCESSING

Second exercise:
difference of frames

- We can see it as a low-level feature

$$\boldsymbol{d}(i_{xy}, l) = |I(i_{xy}, l) - I(i_{xy}, l-1)|$$

- Where I is the frame at instant l evaluated in the pixel i_{xy} with $x \in (1, W); y \in (1, H)$
- You may want to store the old frame in a global variable

- You can use the utility copy_img to copy a source image into a destination one.

```
import processing.video.*;
Capture cam;
int EFFECT_DIFF_FRAMES =2;
int effect= EFFECT_DIFF_FRAMES;


void copy_img(PImage src, PImage dst) {
  dst.set(0,0,src);}

void effectDiffFrames(Pimage img){
   /*your code */}
void draw() {
  if (! cam.available()) {cam.read();}
  cam.read();
  /*  ... as above*/
  else if(effect==EFFECT_DIFF_FRAMES){
     effectDiffFrames(img);}
}
```

IMAGE PROCESSING

In Processing you can also use the OpenCV library

- It exposes a set of video features thay you may use for your applications
  - Including several features that are hard to compute by yourself
  - Similar to the audio mid-level features
- Unfortunately it is not well documented in Processing
  - http://atduskgreg.github.io/opencv-processing/reference/
- Look at the examples to better understand what they do
  - https://github.com/atduskgreg/opencv-processing
  - https://github.com/atduskgreg/opencv-processing/tree/master/examples
- We will cover one of the most interesting one: optical flows!

IMAGE PROCESSING

The Optical Flow features «follow» the movement of the pixels

- It returns a measure of the «movement» of the pixel in the x and y direction
- It is useful to track the movement of an object –or people- in a video
- You can get the average flow in a portion of the image
- Here we will do it line by line

IMAGE PROCESSING

1. In draw: if we haven't yet, we initialize opencv
2. We plot the image
3. We divide the image in squares of *gs* pixels each, centered in *c_x* and *c_y*
4. For each square:
    1. We compute the average optical flow
    2. The result is a PVector with an x and a y component
    3. We draw a line starting from the center of the square to the direction of the optical flow!

```
import gab.opencv.*;
int EFFECT_OPTICAL_FLOW=3; int effect=EFFECT_OPTICAL_FLOW;
OpenCV opencv=null;

void opticalFlow(PImage img){
  opencv.loadImage(img); opencv.calculateOpticalFlow();
  int gs=10; /*compute every 10 pixels*/ int hg=gs/2;
  PVector aveFlow; int c_x=0; int c_y=0;
  image(img,0,0); stroke(255,0,0); strokeWeight(2);
  for (int w=0; w<img.width; w+=grid_size){
    for (int h=0; h<img.height; h+=grid_size){
      aveFlow = opencv.getAverageFlowInRegion(w, h, gs, gs);
      c_x=w+half_grid; c_y=h+half_grid;
      line(c_x, c_y, c_x+min(aveFlow.x*hg, hg),
                     c_y+min(aveFlow.y*hg, hg));}}}

void draw() {
  if(opencv ==null){opencv = new OpenCV(this, cam.width,
cam.height);}
   /*...*/
  if(effect==EFFECT_OPTICAL_FLOW){
     opticalFlow(img); return;} /*we have already plotted*/
    // ...
}
```

IMAGE PROCESSING

Third exercise:

- Change the code of the line so it will follow the intensity of the movement
- You can change the color, the weight of the stroke or whatever you prefer

```
void opticalFlow(PImage img){
  opencv.loadImage(img); opencv.calculateOpticalFlow();
  int gs=10; /*compute every 10 pixels*/ int hg=gs/2;
  PVector aveFlow; int c_x=0; int c_y=0;
  image(img,0,0);
  for (int w=0; w<img.width; w+=grid_size){
    for (int h=0; h<img.height; h+=grid_size){
       aveFlow = opencv.getAverageFlowInRegion(w, h, gs, gs);
       c_x=w+half_grid; c_y=h+half_grid;
       stroke(/*your code*/); strokeWeight(/*your code*/);
       line(c_x, c_y, c_x+min(aveFlow.x*hg, hg),
                      c_y+min(aveFlow.y*hg, hg));}}}

/*...*/
```

ONE LAST THING…

A NEW CHALLENGE

- *When coding your idea, it is very common to learn new tools from tutorial and examples*

- Be curious about new stuff, use social networks to follow artists and tech people who may always make you discover new cool tools

Chirp

- For example: Chirp is a library to transmit data over sound
- No need of wifi, bluetooth, and it's sounds like R2-D2
- Maybe it's useless for artistic performances, maybe you can use it to control an army of devices even without the internet
- Just: be curious