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**

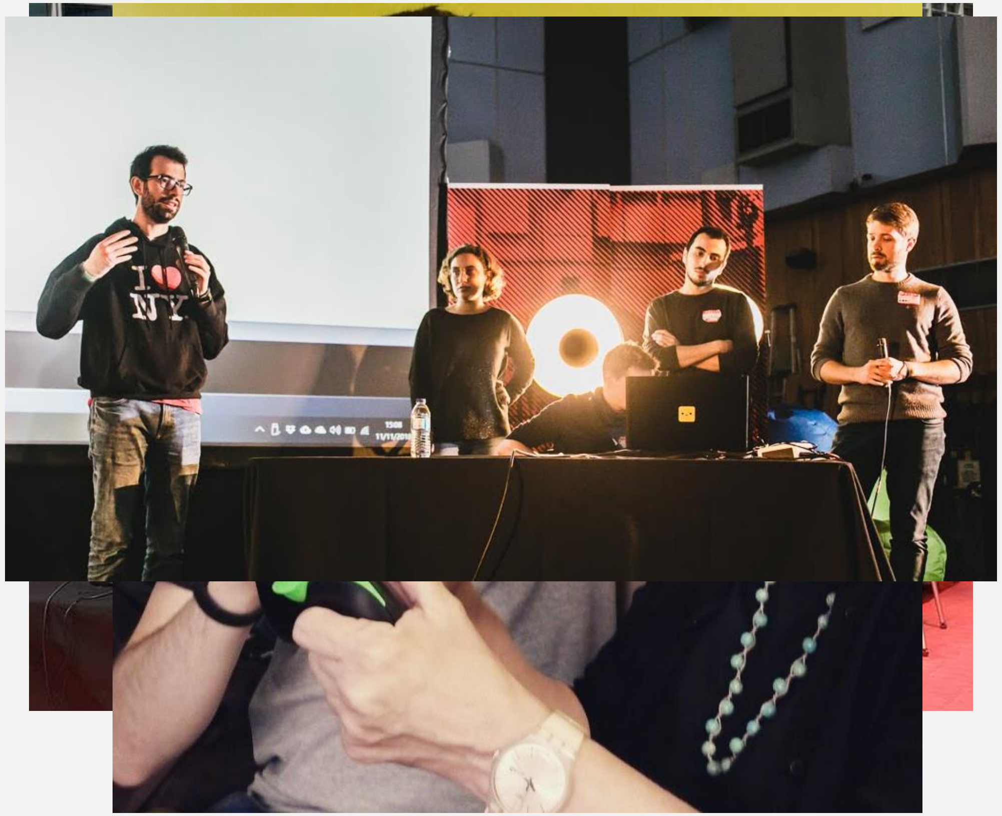
- Researcher for the WholoDance European project

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

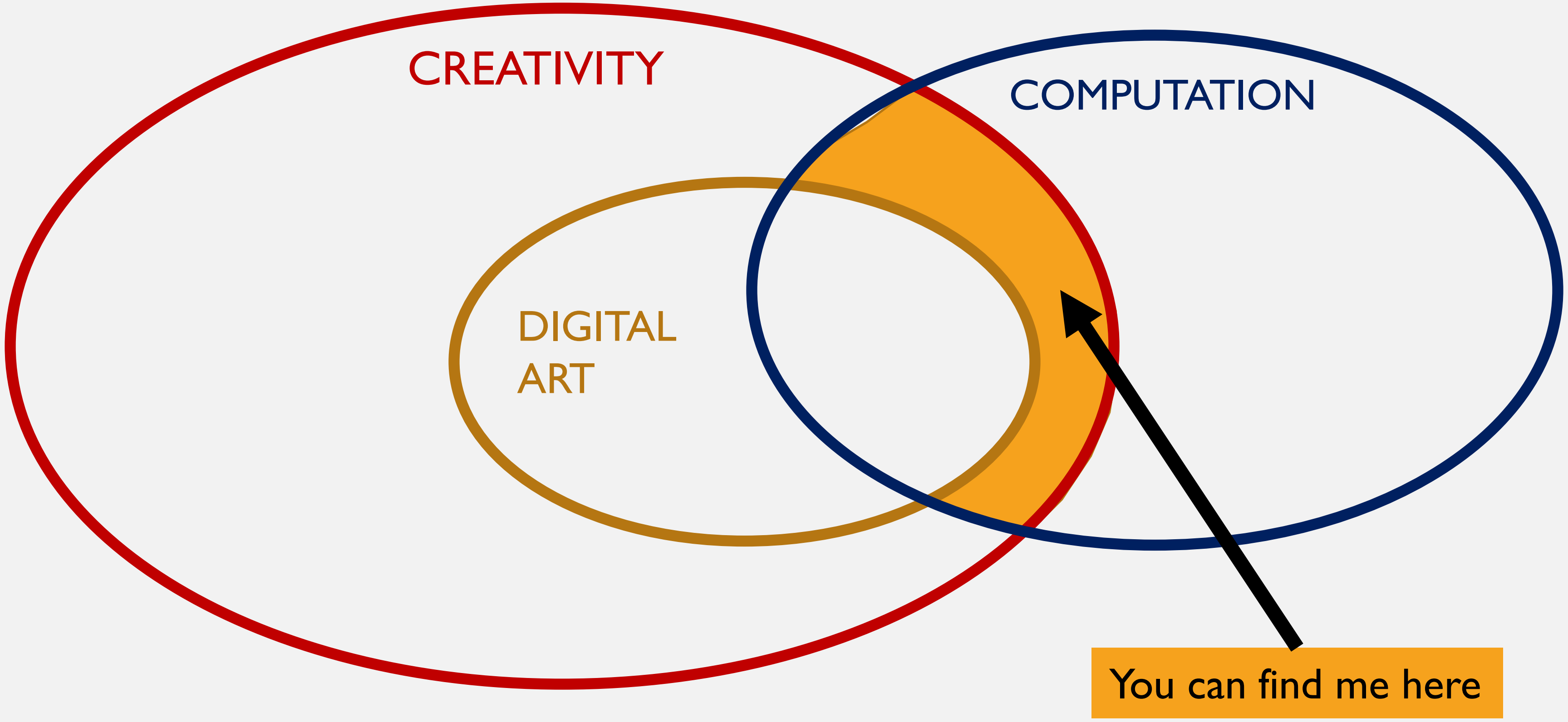
2016 – 2018 **Post-doc researcher**

- 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 cpac2021
- 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_2021

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**

- I will not explain everything in details to get you used to refer to the documentation
  - But feel free to ask if something is not clear
- 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

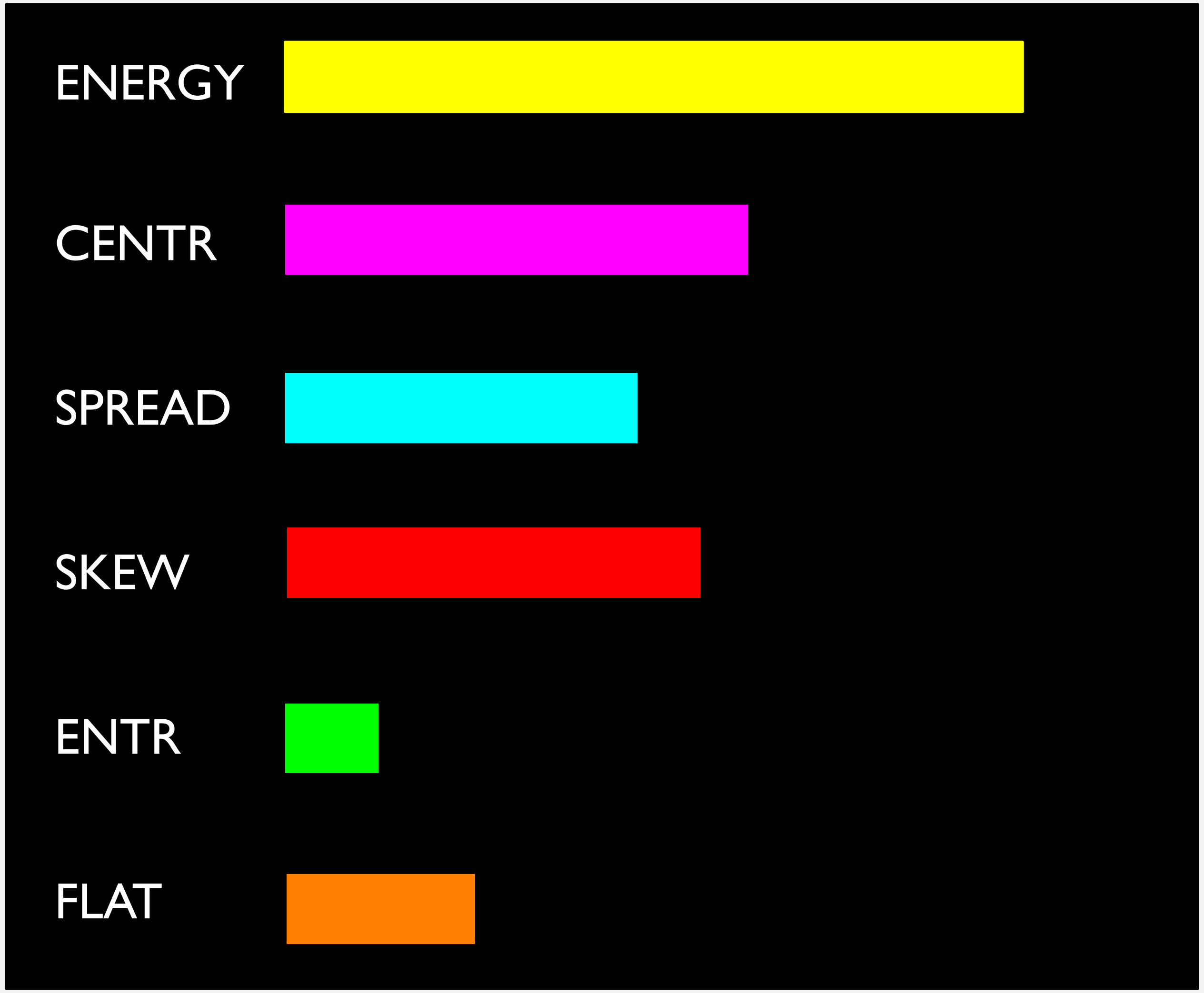
Mid-level features

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 using **Processing**
- 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 get the Fourier Transform using Processing
  - 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:
  - Ex1_feature_visualiser.pde: the main script for processing
  - drawer.pde, with a class AgentDrawer that draws the features
  - feature.pde, which a class AgentFeature that extracts the features
    - **you will edit this**

HAND-CRAFTED AUDIO FEATURES

- processing_ft_visualiser:
  - void setup(): contains the initializations
    - Starts the screen, loads the song
    - Initialize AgentFeature with the song
    - Initialize AgentDrawer with the AgentFeature object and the number of features
  - void draw(): the function is called before each frame (about 60 Hz framerate)
    - Call AgentFeature's method *reasoning* to compute the features
      - **You will edit this**
    - Call AgentDrawer's method *action* to show them on the screen

HAND-CRAFTED AUDIO FEATURES

- processing_ft_visualiser:
  - void setup(): contains the initializations
    - Starts the screen, loads the song
    - Initialize AgentFeature with the song
    - Initialize AgentDrawer with the AgentFeature object a
  - void draw(): the function is called before each frame (ab
    - Call AgentFeature's method *reasoning* to compute the
      - **You will edit this**
    - Call AgentDrawer's method *action* to show them on t

```
# ex1_feature_visualiser.pde
// import and init, see the code

void setup(){
  size(1280, 720); background(0);
  minim = new Minim(this);
  song = minim.loadFile(// ...
  feat = new AgentFeature(song.bufferSize(),
                          song.sampleRate());
  song.play();
  drawer=new AgentDrawer(feat, 6);}

void draw(){
  fill(0);
  rect(0,0,width, height);
  feat.reasoning(song.mix);
  drawer.action();
}
```

HAND-CRAFTED AUDIO FEATURES

We need to extract the following features

- 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

```
# features.pde
float compute_centroid(){
    # your code here
}

class AgentFeature {
  // attributes definition
  AgentFeature(int bufferSize, float sampleRate){
    this.fft = new FFT(bufferSize, sampleRate); this.fft.window(FFT.HAMMING);
    this.K = this.fft.specSize();
    this.freqs = new float[this.K];
    for(int k=0; k<this.K; i++){this.freqs[k]= (0.5*k/this.K)*sampleRate;}
    this.centroid=0; // example
    // your code here
  }
  void reasoning(AudioBuffer mix){
     this.fft.forward(mix);
     float centroid = compute_centroid(/*what do you need?*/);
     this.centroid = centroid
     //  your code here
  }
}
```

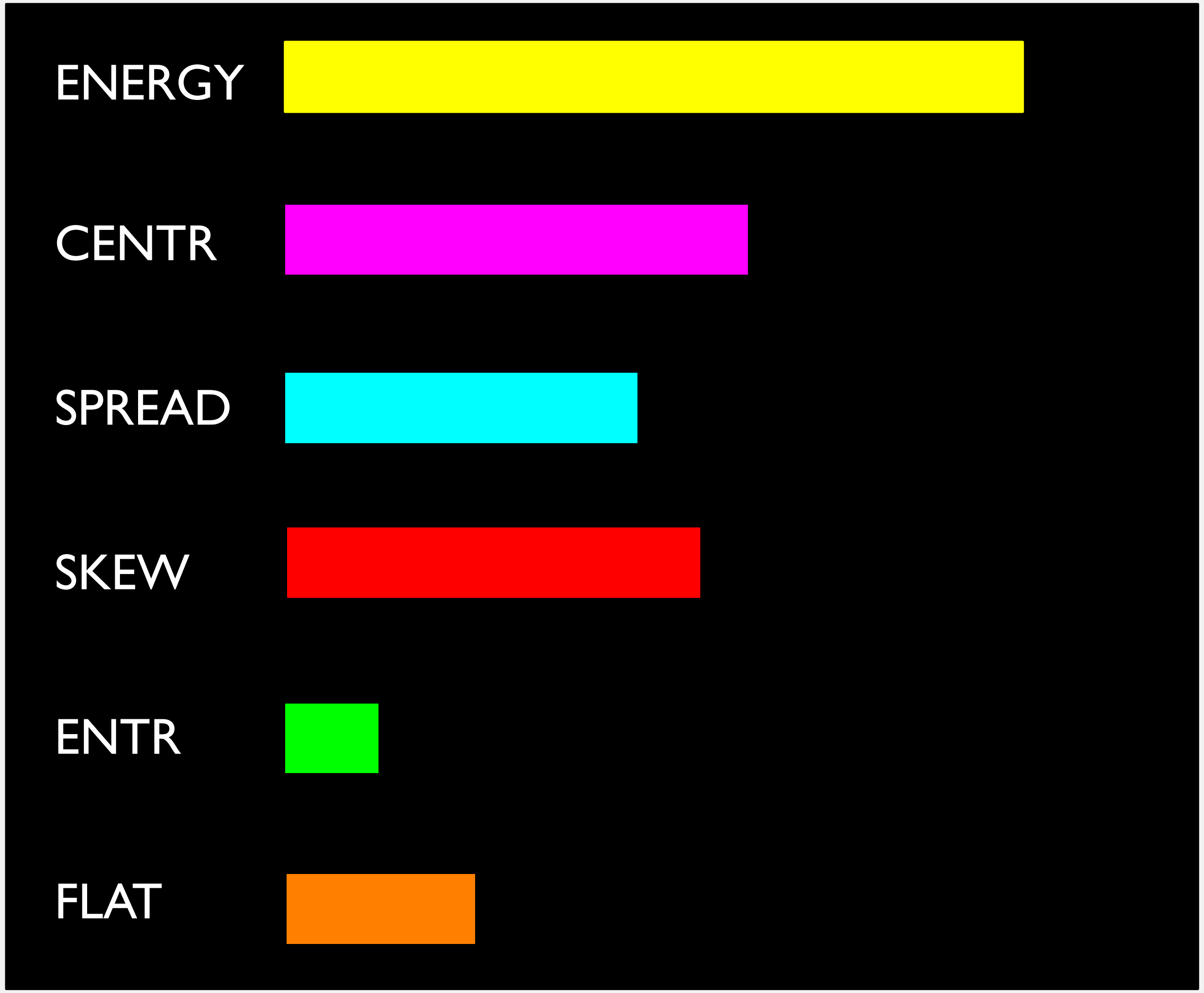
HAND-CRAFTED AUDIO FEATURES

We need to extract the following features

- 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 $S_f = K\frac{\sqrt[K]{\prod_{k=0}^{K-1}|X(k)|}}{\sum_{k=0}^{K-1}|X(k)|}$ degree of flatness in the spectrum

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

HAND-CRAFTED AUDIO FEATURES



Coding time

- Open the feature.pde file with Processing
- Replace the functions *compute_centroid* … with actual code to compute such metrics using formulas from previous slide
- Watch the result
- Hints:
  - Use functions to avoid computing the same quantities more than once
  - Avoid divisions by 0 adding a small constant to denominators

HAND-CRAFTED AUDIO FEATURES

Advanced coding

- Take a look on AgentDrawer, and see some of the code
- For example: how can we represent something we don't know the range of?
  - In AgentDrawer I keep track of minima and maxima values of each features
- How can we map a quantity into a width?
  - In AgentDrawer I use the **map** function
  - value_out= map(value_in, min_in, max_in, min_out, max_out)
    - Min_in, max_in: range of the value
    - Min_out, max_out: 0 to the maximum width of the bar
  - https://processing.org/reference/map_.html

- Don't just *copy* the solution, try to *learn* from it

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 samples

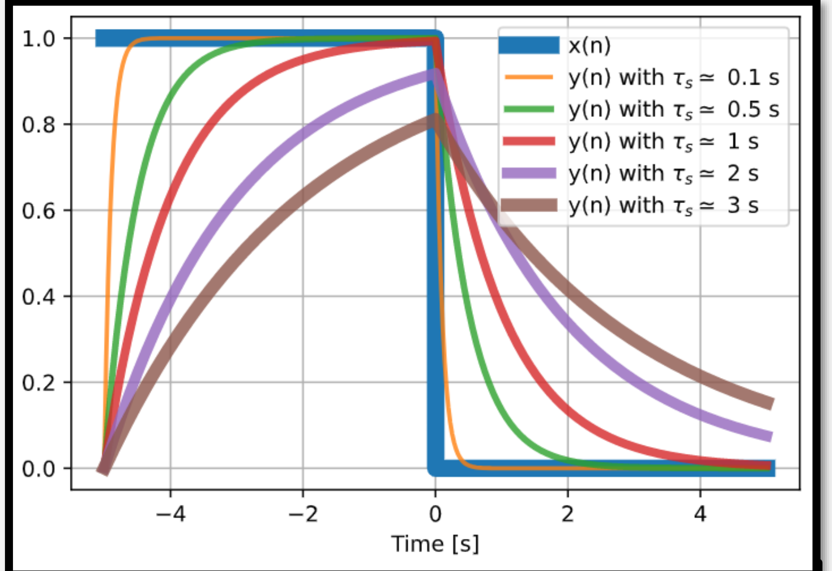
0.28346868942621073

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 → $\alpha = 1 - e^{-2\pi f_c / f_s}$

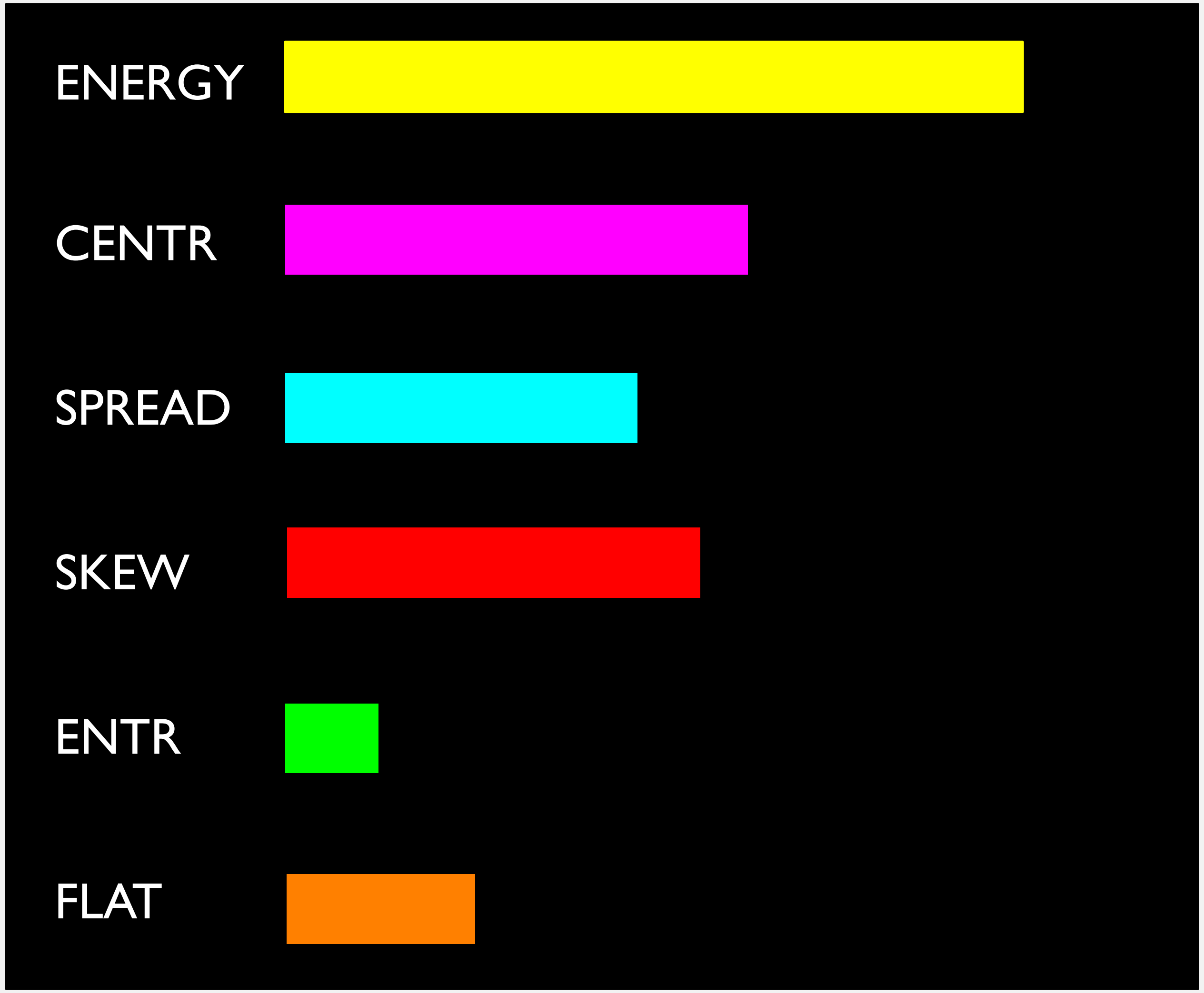
- Example
  - Suppose we want to smooth the signal so it takes 50 ms to change status
  - We know Processing works at 60 Hz
  - Our alpha will be $\alpha = 1 - e^{-1/0.05 \cdot 60}$



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

- Try to implement a smooth filter in the AgentFeature
  - How does it change the visualization?

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: style transfer

MID-LEVEL FEATURES

- *Style transfer* refers to the action of taking a content made with a certain *style* and change it to another
- For example: change the genre of a song
- Proper style transfer take a lot of effort, but some effect can be achieved easily
- Dummy style transfer: put an heavy kick on the beats of any song

MID-LEVEL FEATURES

What do we need to do?

1. Load a song
2. Find the beats
3. Load a heavy kick
4. Put the kick on the song
5. Write the final song

MID-LEVEL FEATURES

What do we need to do?

1. Load a song
2. Find the beats
3. Load a heavy kick
4. Put the kick on the song
5. Write the final song

```python
# main.py
# %% Preliminary operations: check/install libraries
import os
import numpy as np
import librosa
from librosa import load, frames_to_samples
import soundfile as sf
os.chdir(os.path.abspath(os.path.dirname(__file__))) # what ?!
import your_code
DATA_DIR="../../../../data"
assert os.path.exists(DATA_DIR), "wrong data dir"
# %% Define filenames
filename_in=os.path.join(DATA_DIR, "tire_swings.wav")
filename_kick=os.path.join(DATA_DIR, "kick.wav")
filename_out=os.path.join(DATA_DIR, "tire_bum.wav")
```

MID-LEVEL FEATURES

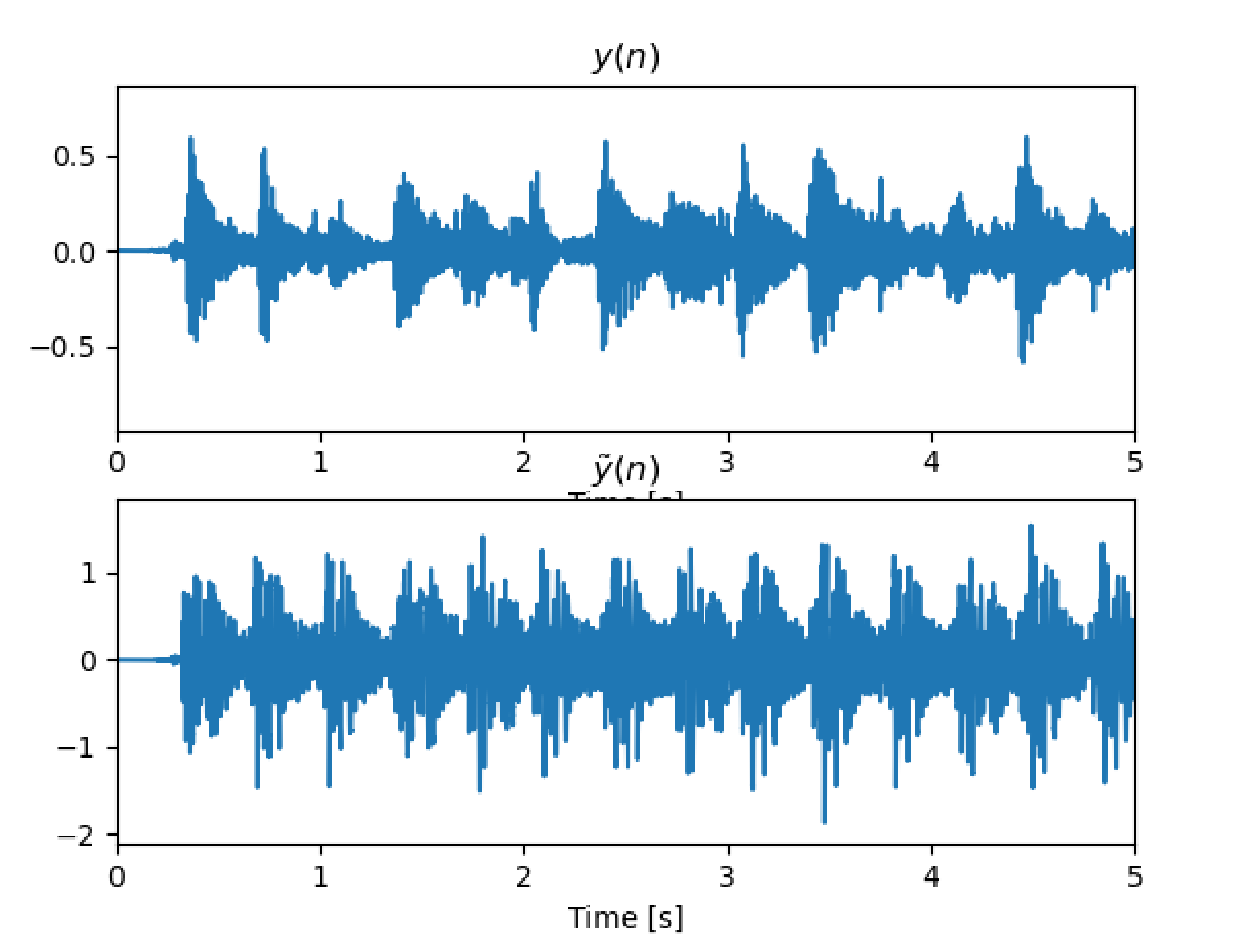
What do we need to do?

1. Load a song into a variable y
2. Find the beats
3. Load a heavy kick into a variable **kick**
4. Put the kick on the song
   - At each beat: **y** must be summed with **kick**
5. Write the final song

Using Librosa package

Using your code

Using Soundfile package



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/
  - we 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

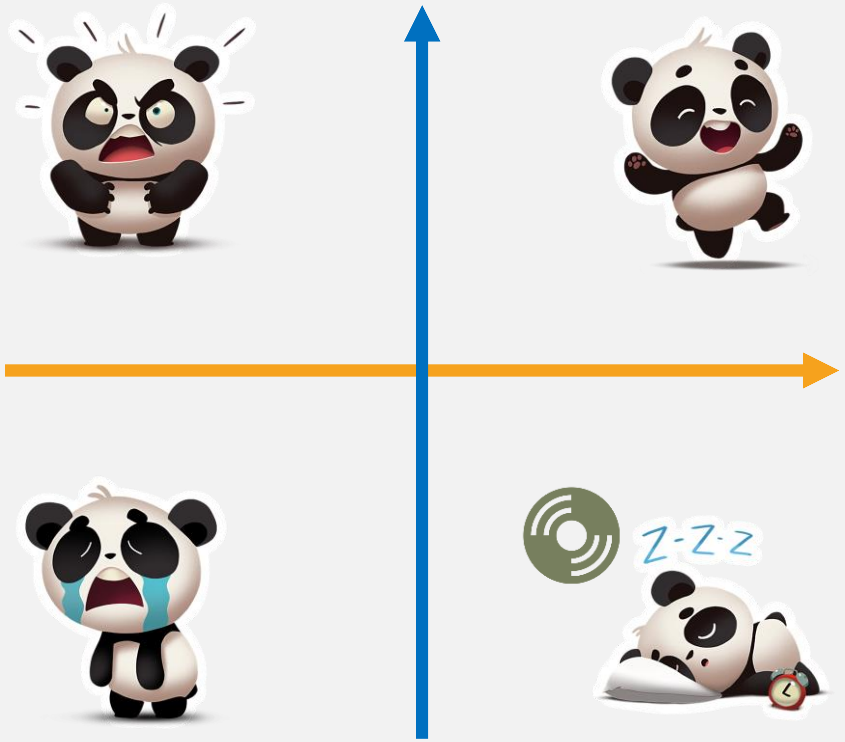
HUMAN READABLE DESCRIPTORS

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

Exercise: create an *automatic collaborative playlist*

- You invite some friends for a party
- Each friend suggests two/three friends → **collaborative playlist**
- Instead of shuffling them, we write an algorithm to sort them → **automatic playlist**

How do we want to sort the songs?

HUMAN READABLE DESCRIPTORS

Exercise: create an *automatic collaborative playlist*

- You invite some friends for a party
- Each friend suggests two/three friends → **collaborative playlist**
- Instead of *shuffling* them, we write an algorithm to *sort* them → **automatic playlist**

How do we want to sort the songs?

Let's create a script to create a REAL playlist on your personal Spotify account.

- If you don't want to, set the flag `CREATE_SPOTIFY_PLAYLIST = False`

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
   - Use the Python's json package
3. Download the audio features for each song into the list `audio_features`
4. Implement the function `shuffled_songs= your_code.sort_songs(audio_features)` following any criterion of your choice, such as
   - songs from lowest to highest danceability, to make the groove grows
   - songs with the lowest levels of danceability and energy at the end of the playlist, to let people go home
   - You compute mood (as the angle in the VA space) and sort songs by mood
5. 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

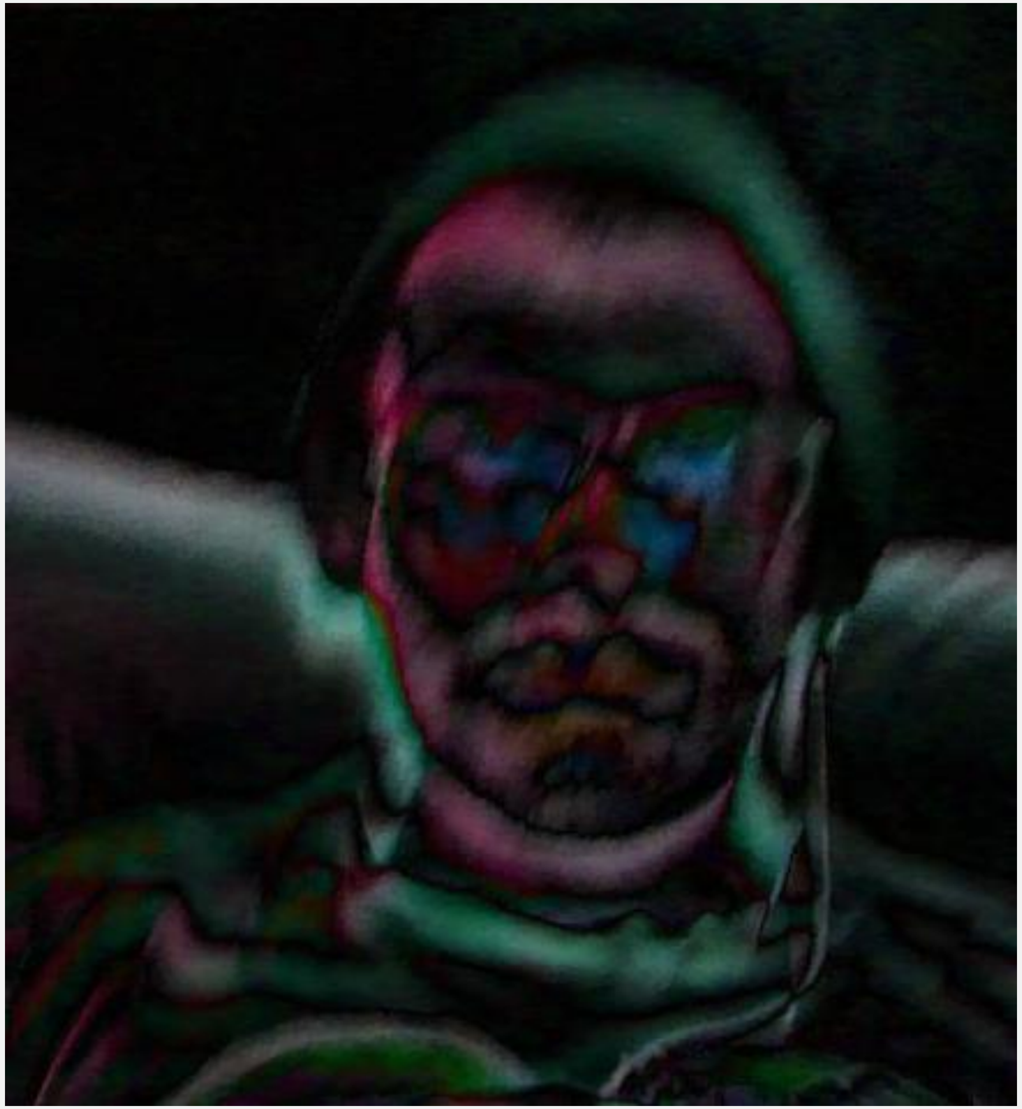


IMAGE PROCESSING

- 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 natural to visualise them
- We can show 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

- If you are using macOS, you might not be able to access the camera
- Use Processing 4 or this tutorial to solve the bug
- https://www.youtube.com/watch?v=xNa_ua_esmw

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

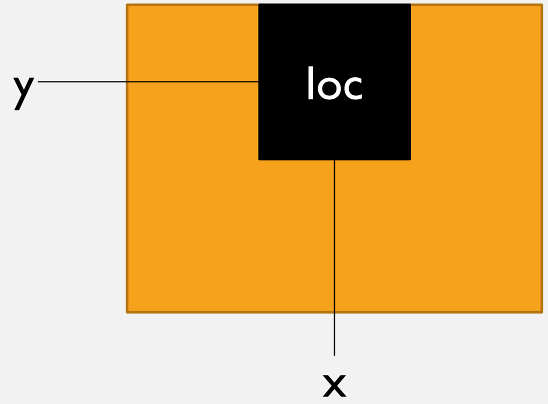
- Let's run this piece of script first
- depending on your computer, C might be 0, or 1, or 2, etc…
  - Check it until you get something
- Once you find the right C , change W and H (width and height) of the screen accordingly

```
# ex4_video_tutorial.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(i, 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



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 loc=0; loc<cam.width*cam.height;
                                    loc ++) {
  x= loc % cam.width;
  y= loc /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 with the function color():

- Passing one parameter:
  - color(gray): 0 is black, 255 is white
- Passing three parameters:
  - color(v1, v2, v3)
  - v1, v2, v3 depend on colorMode (see right)
- Adding an alpha: 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

- We can 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

First exercize: switch colors!

- write a function that changes the camera image so that the colors are somehow changed
- how: it is up to you

```
# ex4_switch_colors.pde
import processing.video.*;
Capture cam;

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);
   /*your code!*/
   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 $x \in (1, W); y \in (1, H)$
- Store the old frame in a global variable
- Use the utility copy_img to copy a source image into a destination one.

```
# ex4_difference_of_frames.pde
import processing.video.*;
Capture cam;
Pimage old_frame, cur_frame, img;
boolean first_frame=true;

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();
  PImage img = // as above
  copy2img(cam, cur_frame);
  effectDiffFrames(img);
  if(img.width>0){image(img, 0, 0);}
  }
}
```

IMAGE PROCESSING

In Processing you can also use the OpenCV library

- It is a set of video features thay you can 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

You need to install opencv from the "add Tool" menu

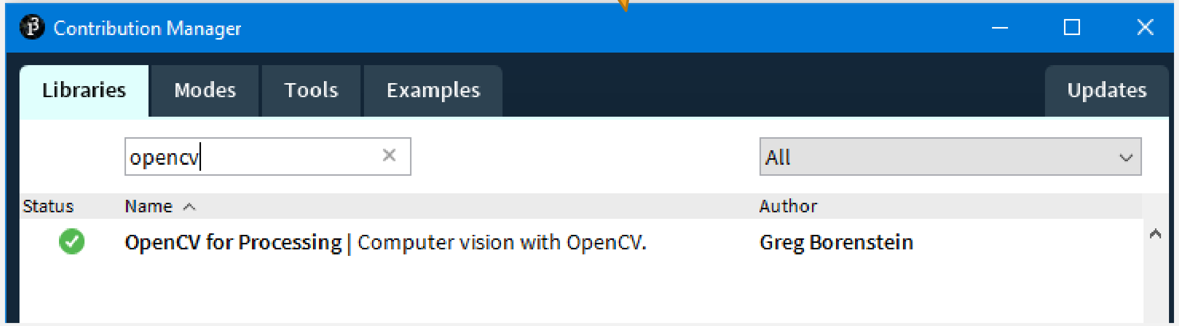


IMAGE PROCESSING

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

- Since the code is complex, we will not cover it as an exercise, but just see how it works from the solution already

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

- For example: Pedalboard is a brand new (September 2021) Python platform from Spotify
  - *Pedalboard makes it easy to use studio-quality audio effects in your code, rather than just in your digital audio workstation (DAW).*
  - *Artists, musicians, and producers with a bit of Python knowledge can use Pedalboard to produce new creative effects that would be extremely time consuming and difficult to produce in a DAW.*