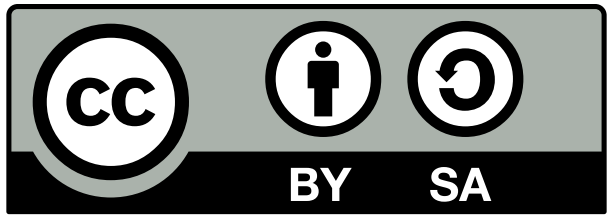
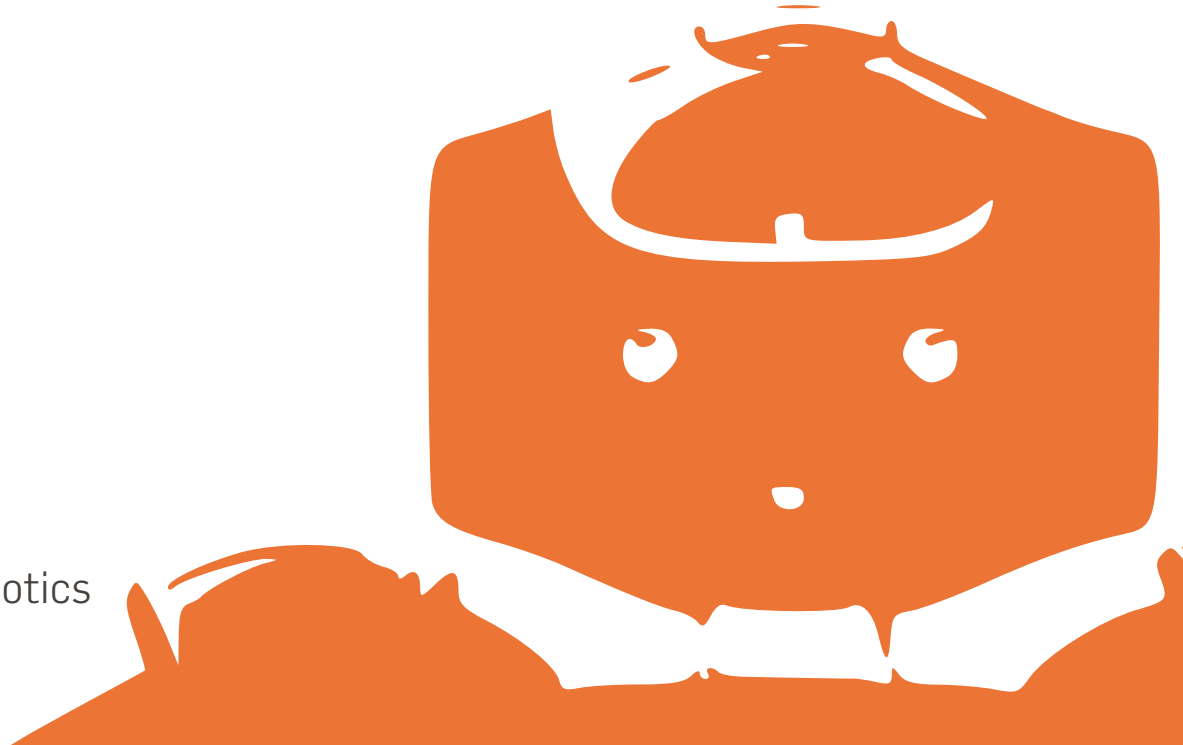


ROBOTICS WITH PLYMOUTH UNIVERSITY

The face: detection, recognition, attention

AINT512



Séverin Lemaignan

Centre for Neural Systems and Robotics
Plymouth University

THIS WEEK

This week, we are looking at the face:

- face detection
- face recognition
- head pose estimation
- attention tracking

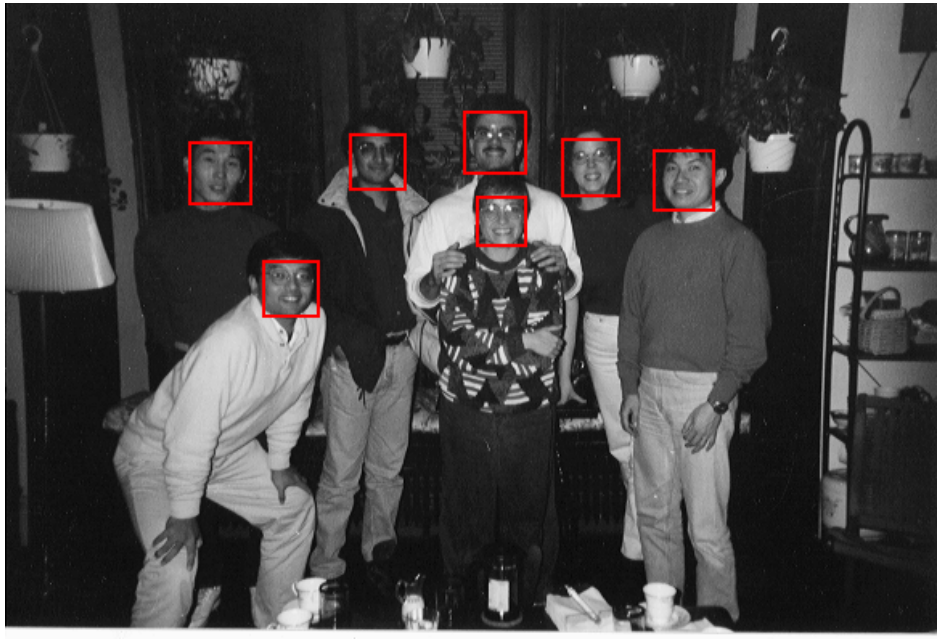
FACE DETECTION

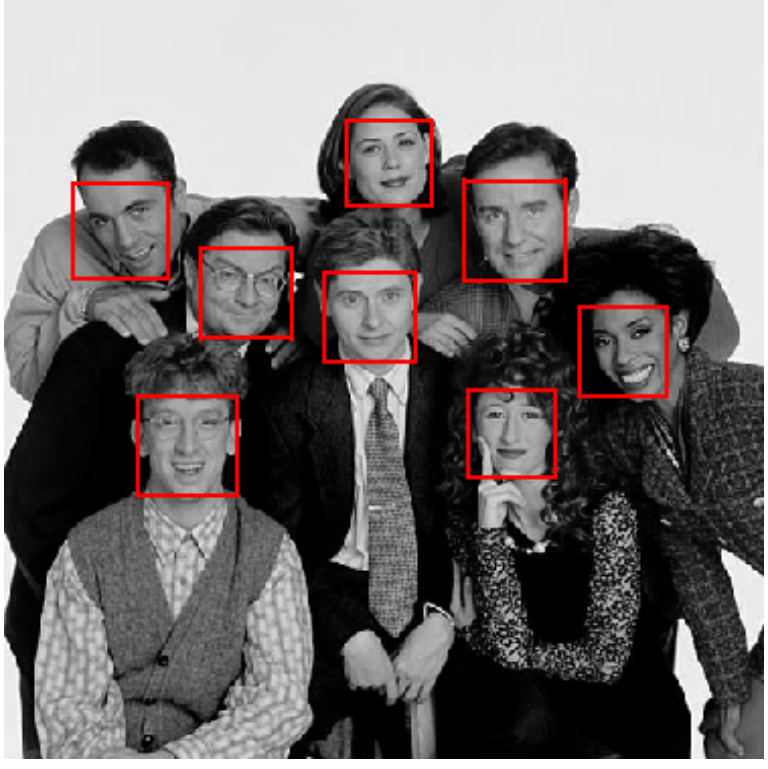
Slides in this section are based on Svetlana Lazebnik's material.

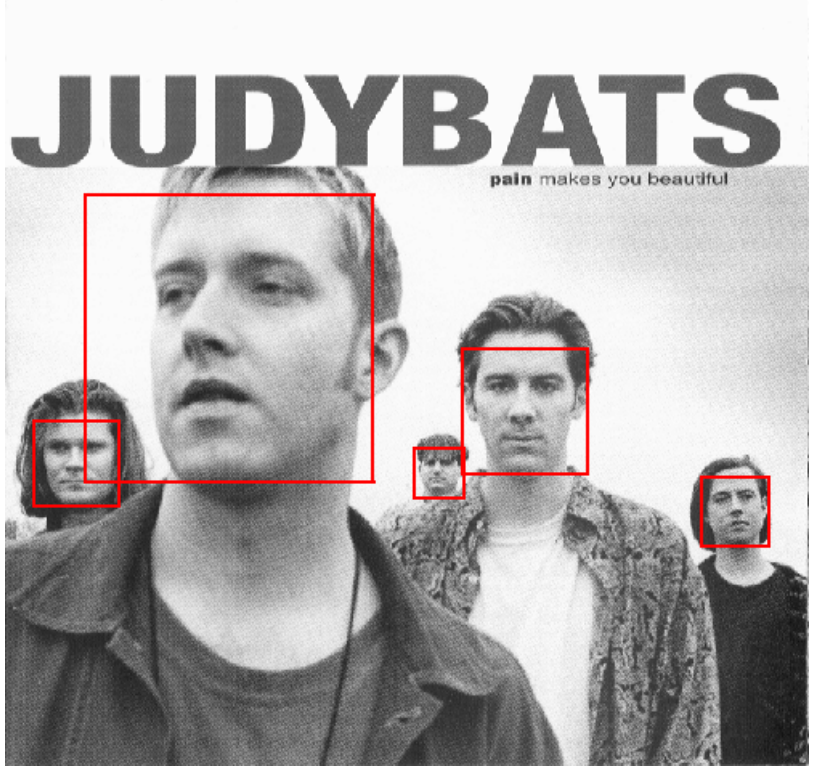
She has many other really good lectures on computer vision.

Go and check it!

See also the OpenCV tutorials: they provides a lot of explanations on how face detection works.






JUDYBATS
pain makes you beautiful

FACE DETECTION WITH THE VIOLA-JONES ALGORITHM

- de-facto standard for face detection
- available in OpenCV
- (relatively) slow training, but fast detection

[P. Viola and M. Jones. ***Robust real-time face detection.*** IJCV 57(2), 2004.]

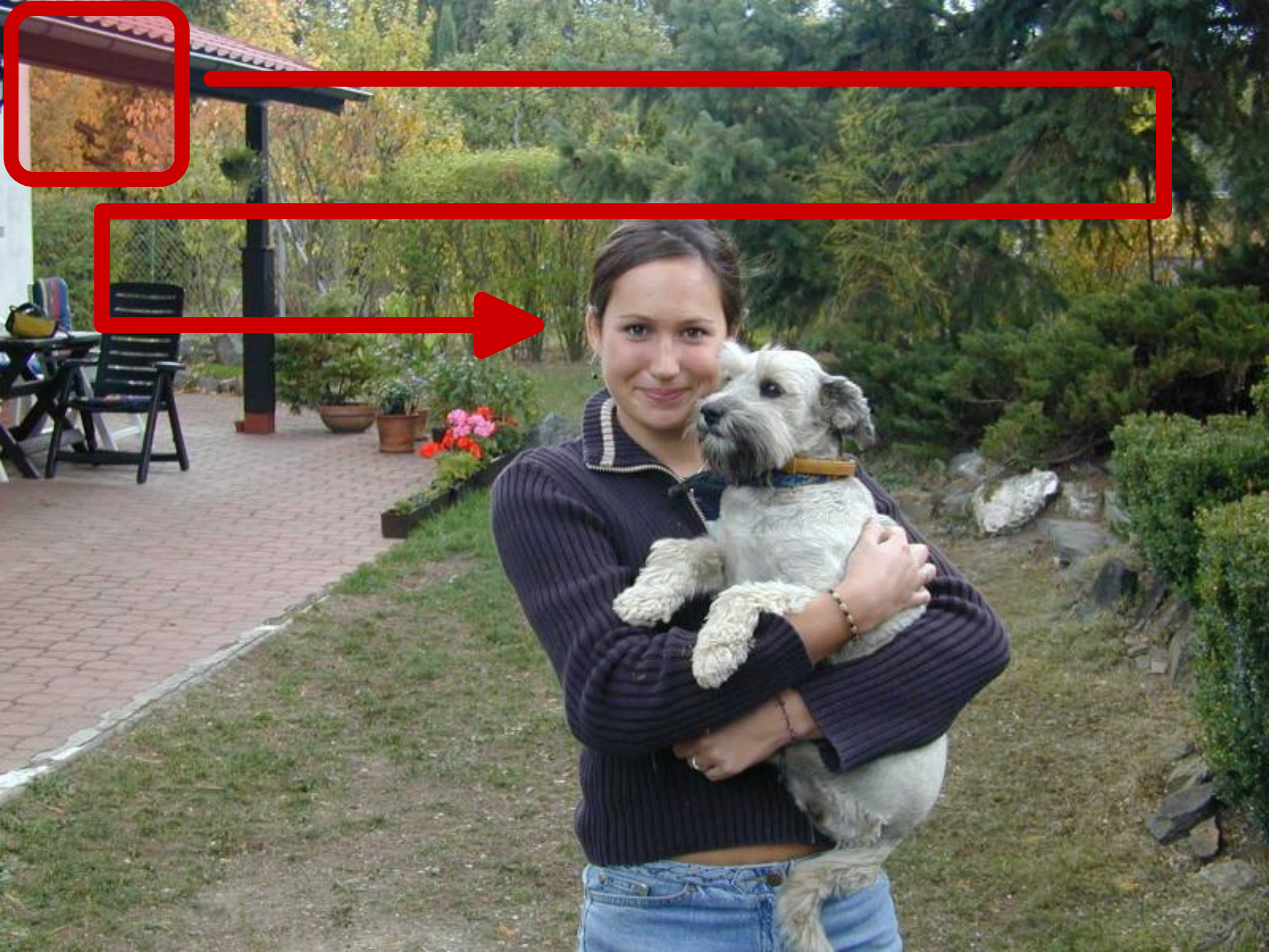
FACE DETECTION WITH THE VIOLA-JONES ALGORITHM

- de-facto standard for face detection
- available in OpenCV
- (relatively) slow training, but fast detection

Key ideas:

- *Sliding window*
- *Rectangle filters* as image features
- *Integral images* for fast feature evaluation
- *Boosting* for feature selection
- *Attentional cascade* for fast rejection of non-face windows

[P. Viola and M. Jones. ***Robust real-time face detection.*** IJCV 57(2), 2004.]



CHALLENGES OF SLIDING WINDOW DETECTION

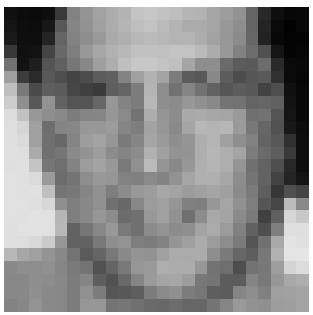
Detector must evaluate tens of thousands of location/scale combinations

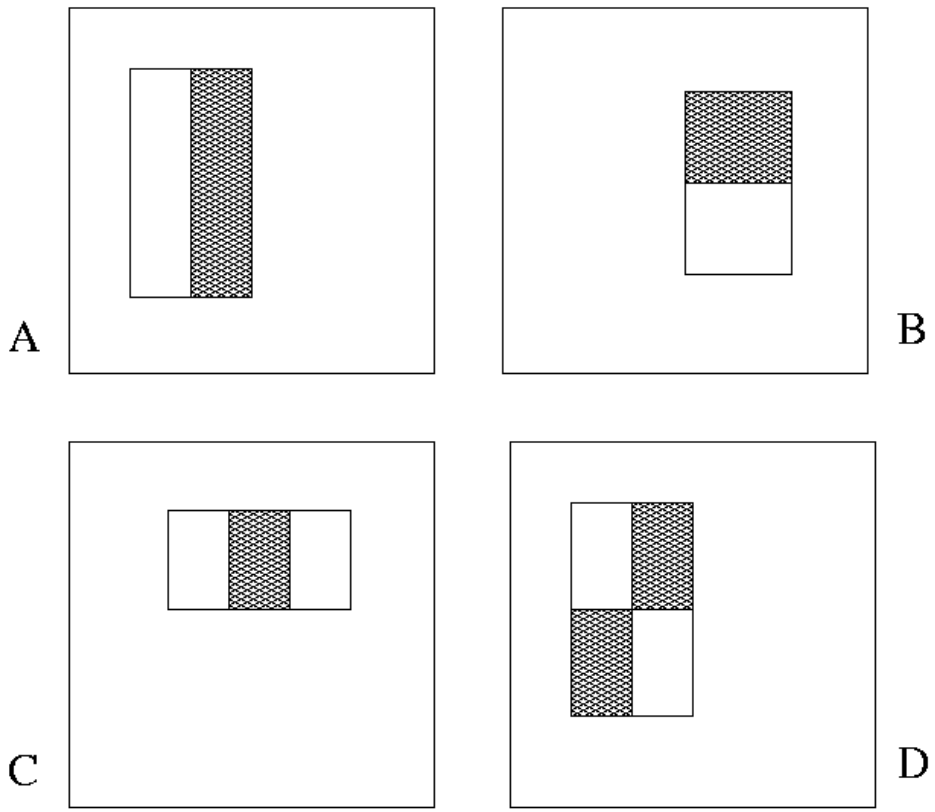
Positive instances are rare: $0 - 10$ per image

- A megapixel image has $\approx 10^6$ pixels and a comparable number of candidate object locations
- For computational efficiency, we should try to spend as little time as possible on the negative windows
- To avoid having a false positive in every image, our false positive rate has to be less than 10^{-6}.

IMAGE FEATURES

Rectangle filters

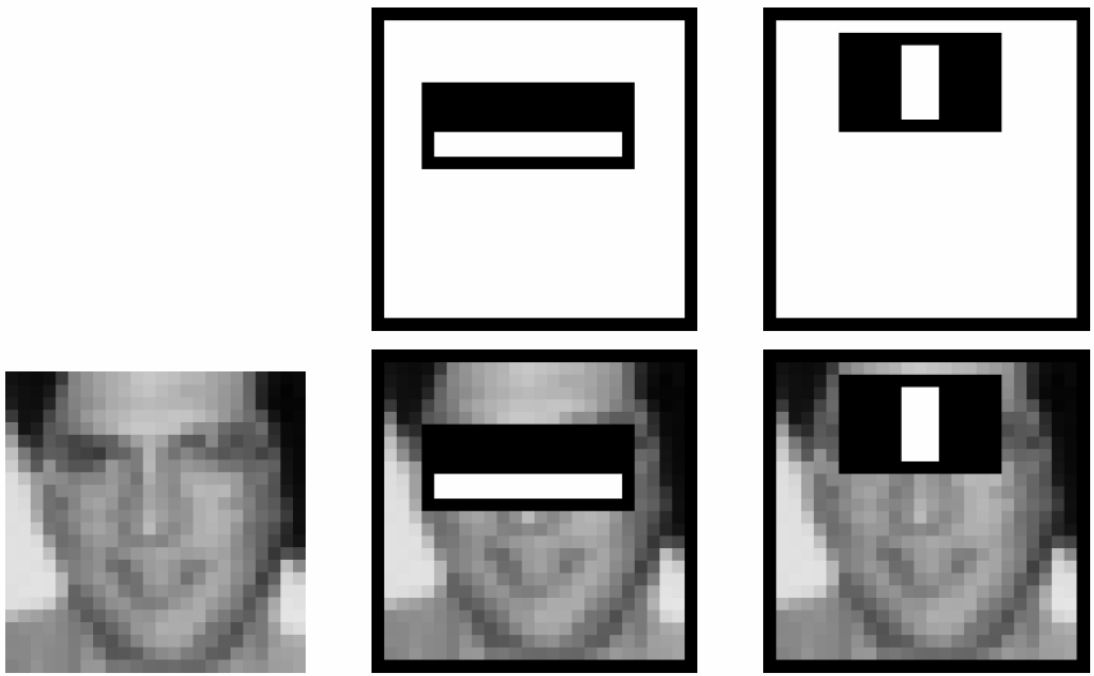




$$value = \sum(\text{pixels in white area}) - \sum(\text{pixels in black area})$$

IMAGE FEATURES

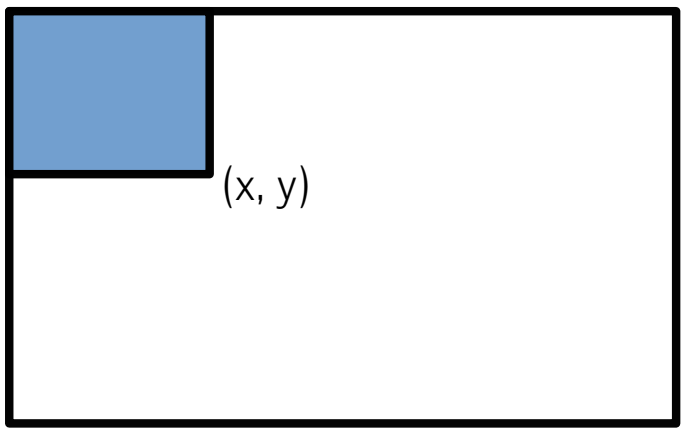
Rectangle filters



$$value = \sum(\text{pixels in white area}) - \sum(\text{pixels in black area})$$

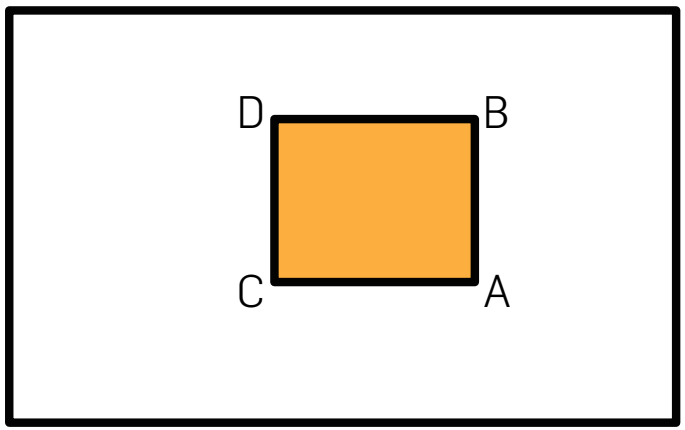
FAST COMPUTATION WITH INTEGRAL IMAGES

- The *integral image* computes a value at each pixel (x,y) that is the sum of the pixel values above and to the left of (x,y), inclusive
- This can quickly be computed in one pass through the image



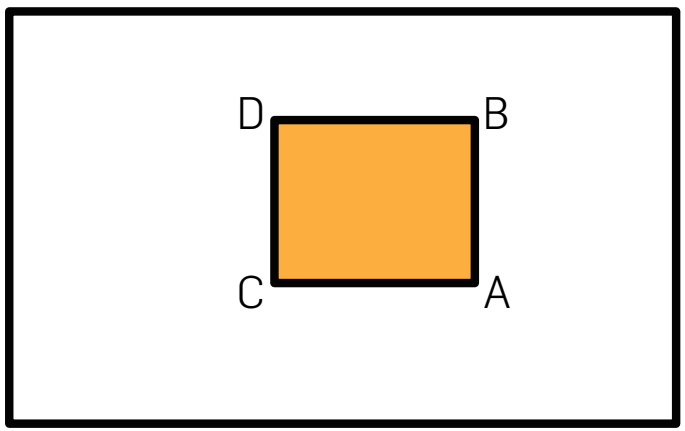
COMPUTING SUM WITHIN A RECTANGLE

- Let A,B,C,D be the values of the integral image at the corners of a rectangle
- What is the sum of pixel values within the rectangle?

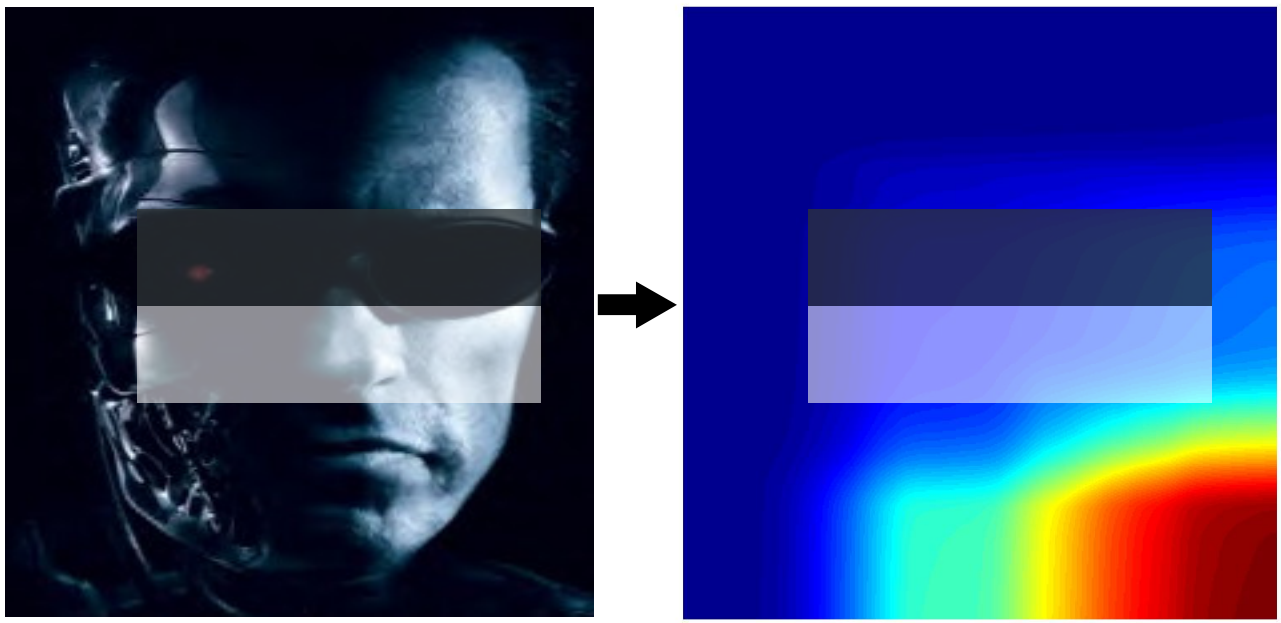


COMPUTING SUM WITHIN A RECTANGLE

- Let A,B,C,D be the values of the integral image at the corners of a rectangle
- What is the sum of pixel values within the rectangle?
- Only 3 additions are required for any size of rectangle!
  sum $= A - B - C + D$

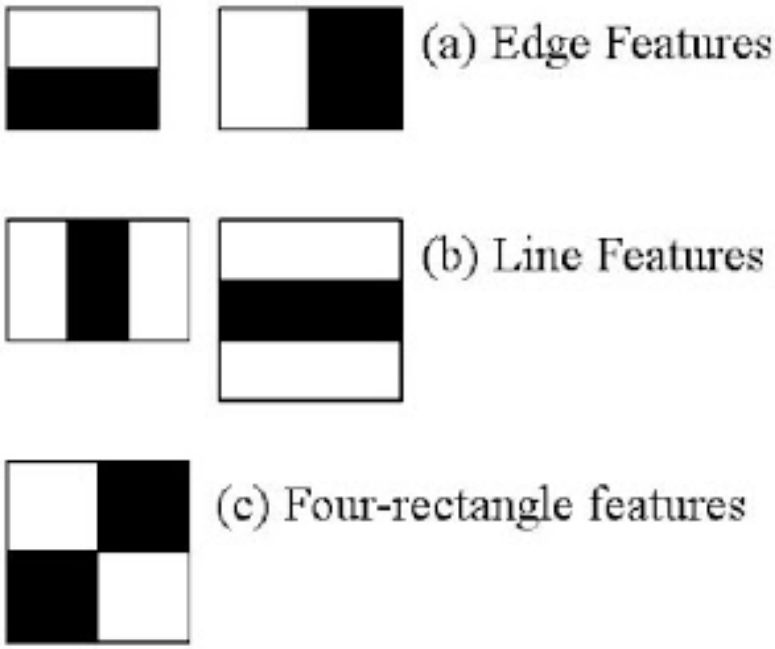


COMPUTING A RECTANGLE FEATURE



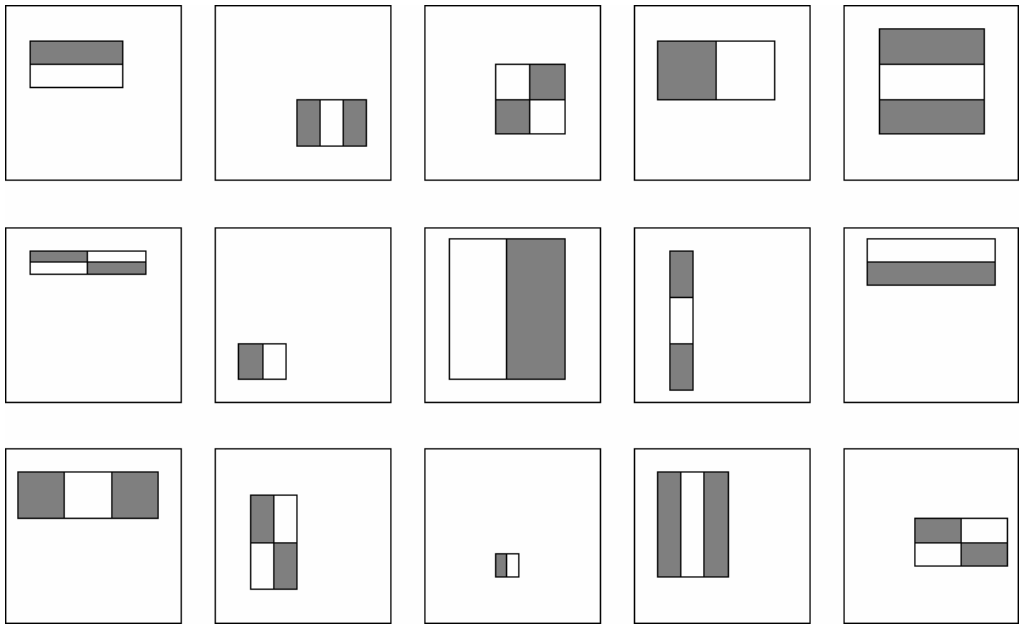
$$value = \sum(pixels\ in\ white\ area) - \sum(pixels\ in\ black\ area)$$

FEATURE SELECTION



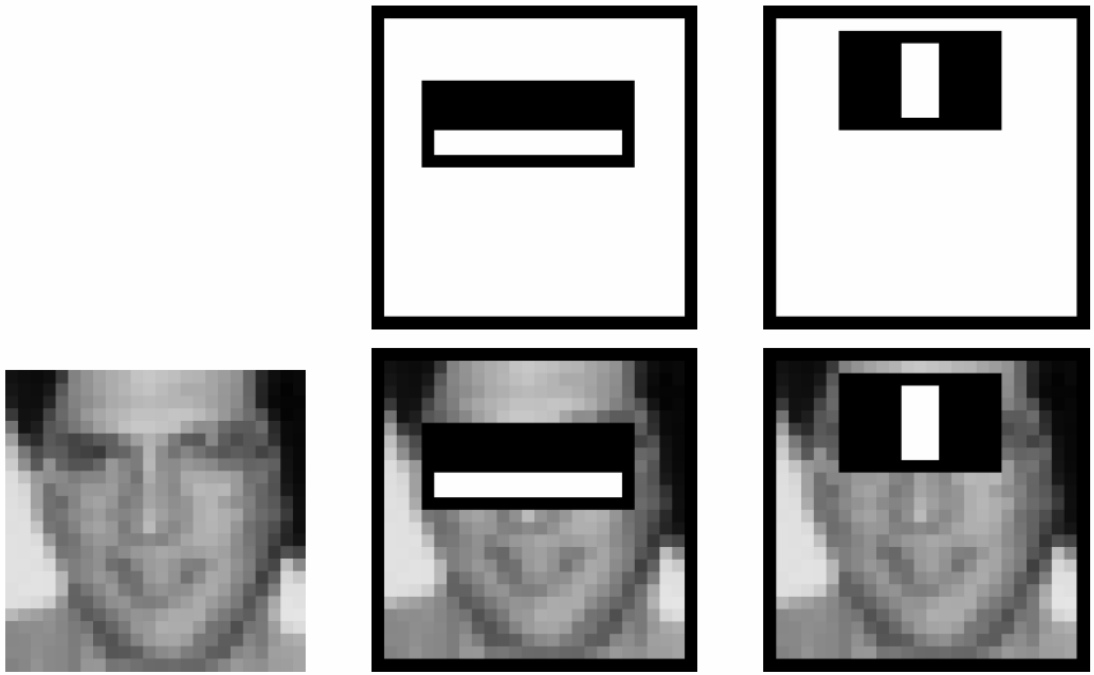
Haar features (used e.g. by the OpenCV implemetation). All sizes & positions are tested in the detection window.

FEATURE SELECTION



- For a 24x24px detection window, the number of possible rectangle features is $\approx 160,000$!
- At test time, it is impractical to evaluate the entire feature set
- Can we create a good classifier using just a small subset of all possible features?

BOOSTING



The key idea of **boosting** is to (1) select simple, yet fast classifiers (***weak learners***) $\Rightarrow$ select the best rectangle features.

(2) combine them linearly.

BOOSTING

A *learner* $h_t(x)$ looks like that:

rectangle feature

window

$$h_t(x) = \begin{cases} 1, & \text{if } f_t(x) > \theta_t \\ 0, & \text{otherwise} \end{cases}$$

threshold

Then, at round t:

- find the weak learner with lowest training error
- increase the weights of training examples misclassified by current weak learner
- repeat.

[Freund, Y., Schapire, R., Abe, N. **A short introduction to boosting**, 1999]

BOOSTING

A *learner* $h_t(x)$ looks like that:

$$h_t(x) = \begin{cases} 1, & \text{if } f_t(x) > \theta_t \\ 0, & \text{otherwise} \end{cases}$$

The final classifier is a weighted linear combination of all weak learners

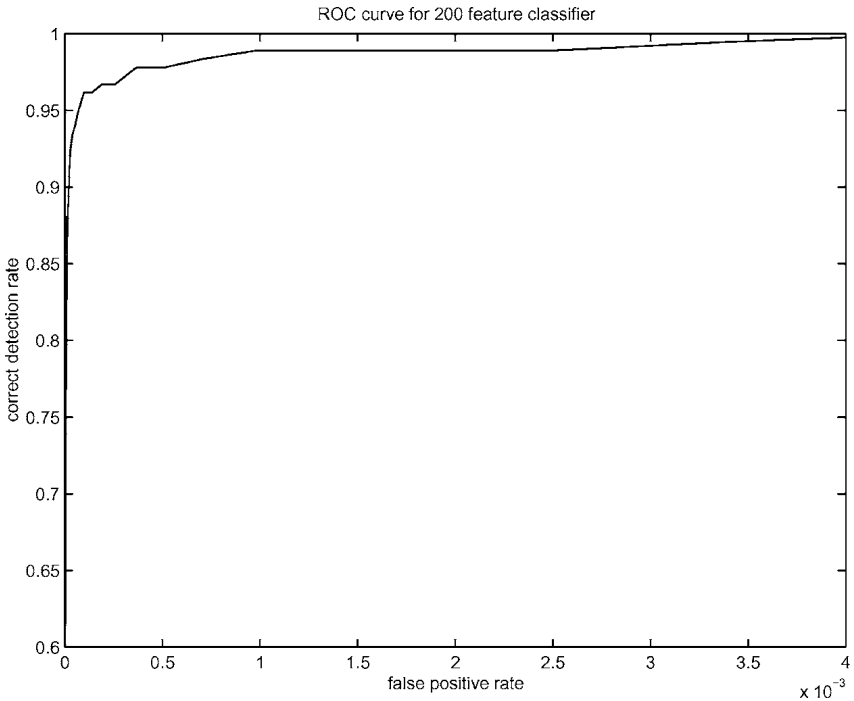
$$C(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) > \frac{1}{2} \sum_{t=1} \alpha_t \\ 0, & \text{otherwise} \end{cases}$$

learned weights

[Freund, Y., Schapire, R., Abe, N. **A short introduction to boosting**, 1999]

BOOSTING FOR FACE DETECTION

A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084

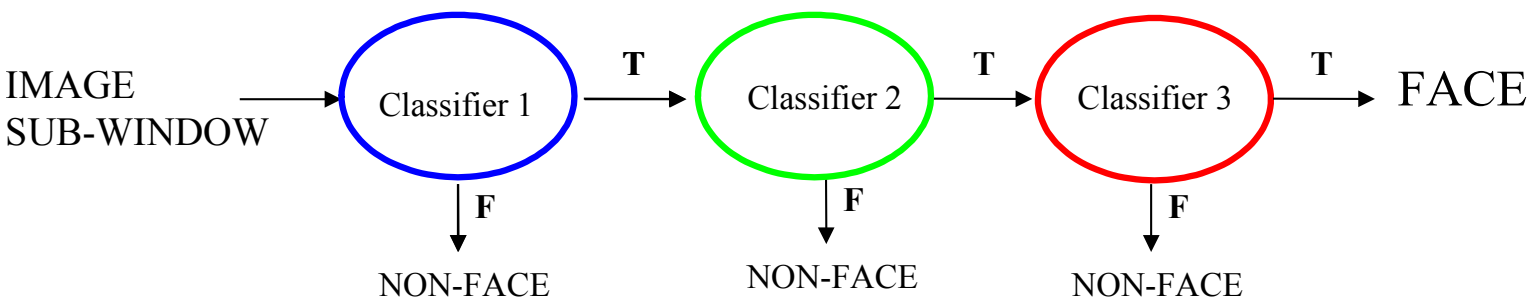


Receiver operating characteristic (ROC) curve

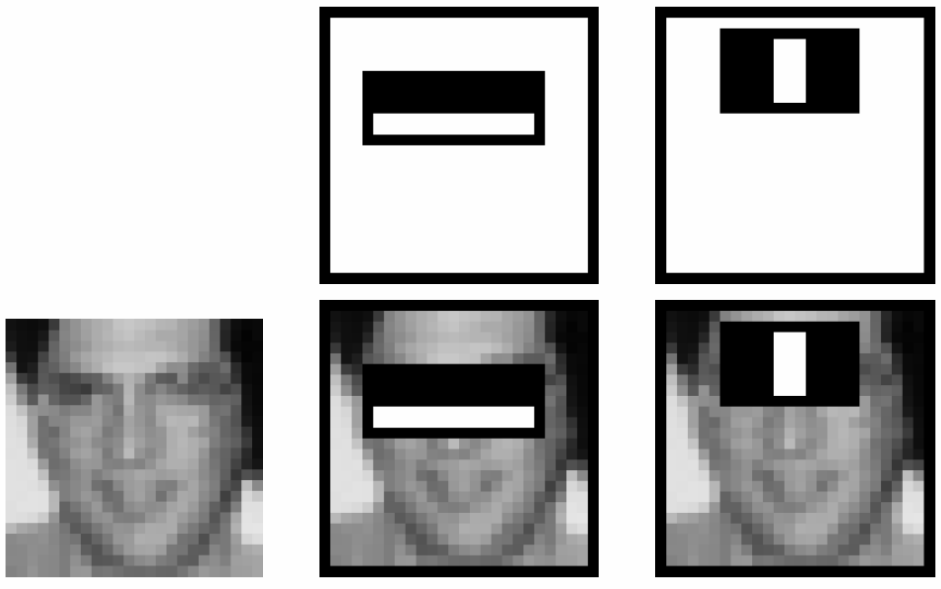
OpenCV implementation uses ≈ 6000 features.

FUTHER IMPROVEMENT: ATTENTIONAL CASCADES

- Viola-Jones algorithm implements **attentional cascades**
- simple classifiers which reject many of the negative sub-windows while detecting almost all positive sub-windows
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on
- A negative outcome at any point leads to the immediate rejection of the sub-window



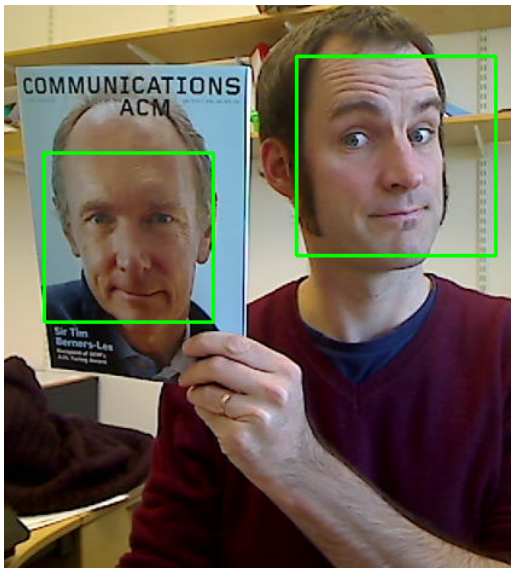
FUTHER IMPROVEMENT: ATTENTIONAL CASCADES



- the 6000 features are distributed over 38 stages (cascades)
- 1, 10, 25, 25 and 50 features in first five
- the 2 features above are actually obtained as the best two features from boosting
- on an average, 10 features out of 6000 are evaluated per sub-window

PYTHON CODE

```
import cv2

cascPath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)

while True:
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
    )

    for x, y, w, h in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imshow('Faces', frame)
    cv2.waitKey(10)
```



STATE-OF-THE-ART FACE DETECTION DEMO

Source: Boris Babenko

...from face detection to...


Microsoft
Making sure it's you...
12:39
Friday, December 4
1
9
10

FACE RECOGNITION: PRINCIPAL COMPONENT ANALYSIS

?

PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a technique to find the sources of variance in a dataset.

[*Source: Mark Richardson, Principal Component Analysis*]

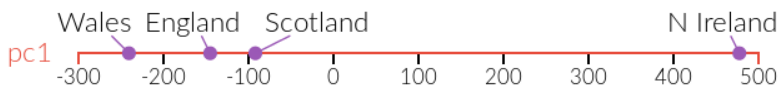
PRINCIPAL COMPONENT ANALYSIS

	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175

[*Source: Mark Richardson, Principal Component Analysis*]

PRINCIPAL COMPONENT ANALYSIS

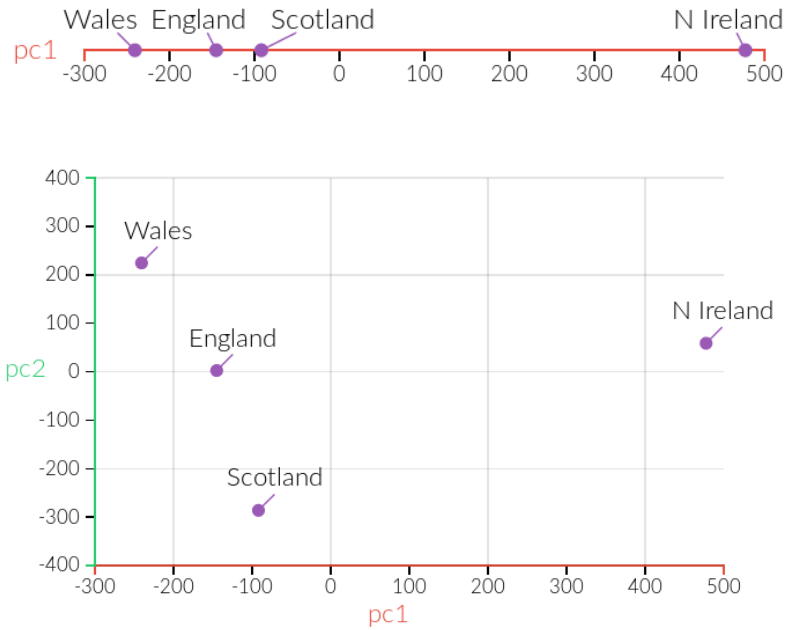
	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175



[*Source: Mark Richardson, Principal Component Analysis*]

PRINCIPAL COMPONENT ANALYSIS

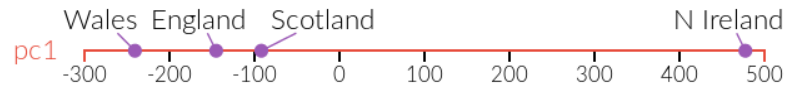
	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175

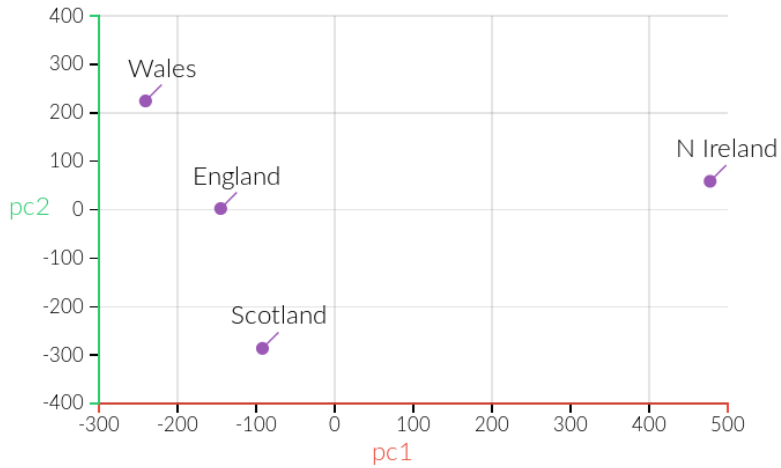


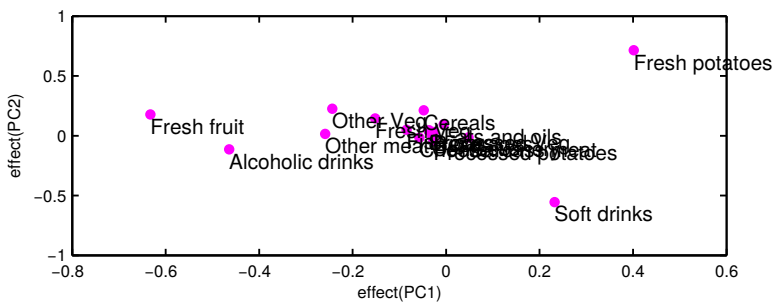
[*Source: Mark Richardson, Principal Component Analysis*]

PRINCIPAL COMPONENT ANALYSIS

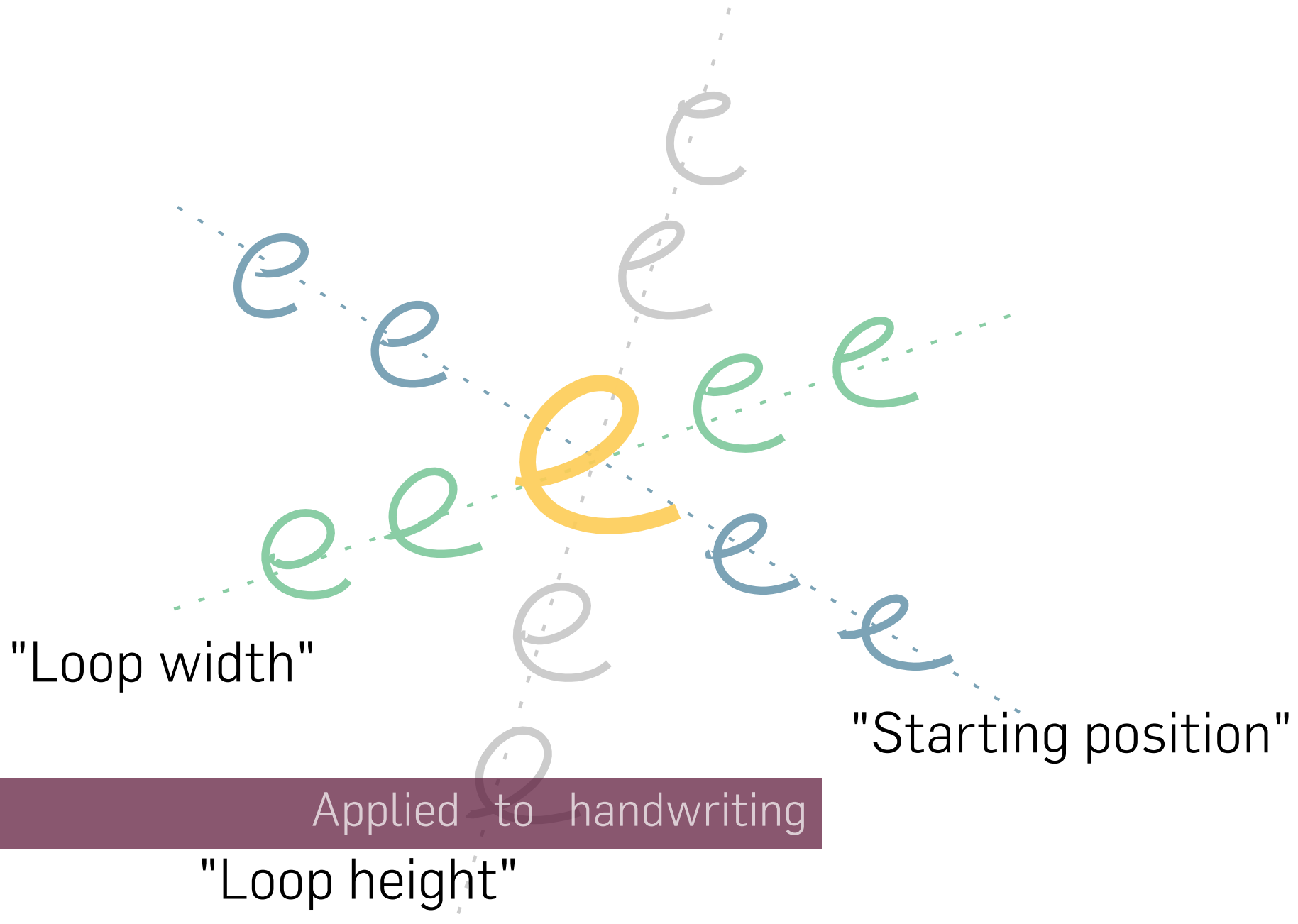
	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175







[*Source: Mark Richardson, Principal Component Analysis*]


"Loop width"
"Starting position"
Applied to handwriting
"Loop height"

Getting the NAO to write

Writing letters badly

Learning to write well

AT&T Face dataset

Applied to faces

PCA ALGORITHM

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a vector with observations $\mathbf{x}_i \in \mathbb{R}^d$.

1. Compute the mean μ

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

2. Compute the the Covariance Matrix S

$$S = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

3. Compute the eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ of S

$$S \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \text{with } i = 1, 2, \ldots, n$$

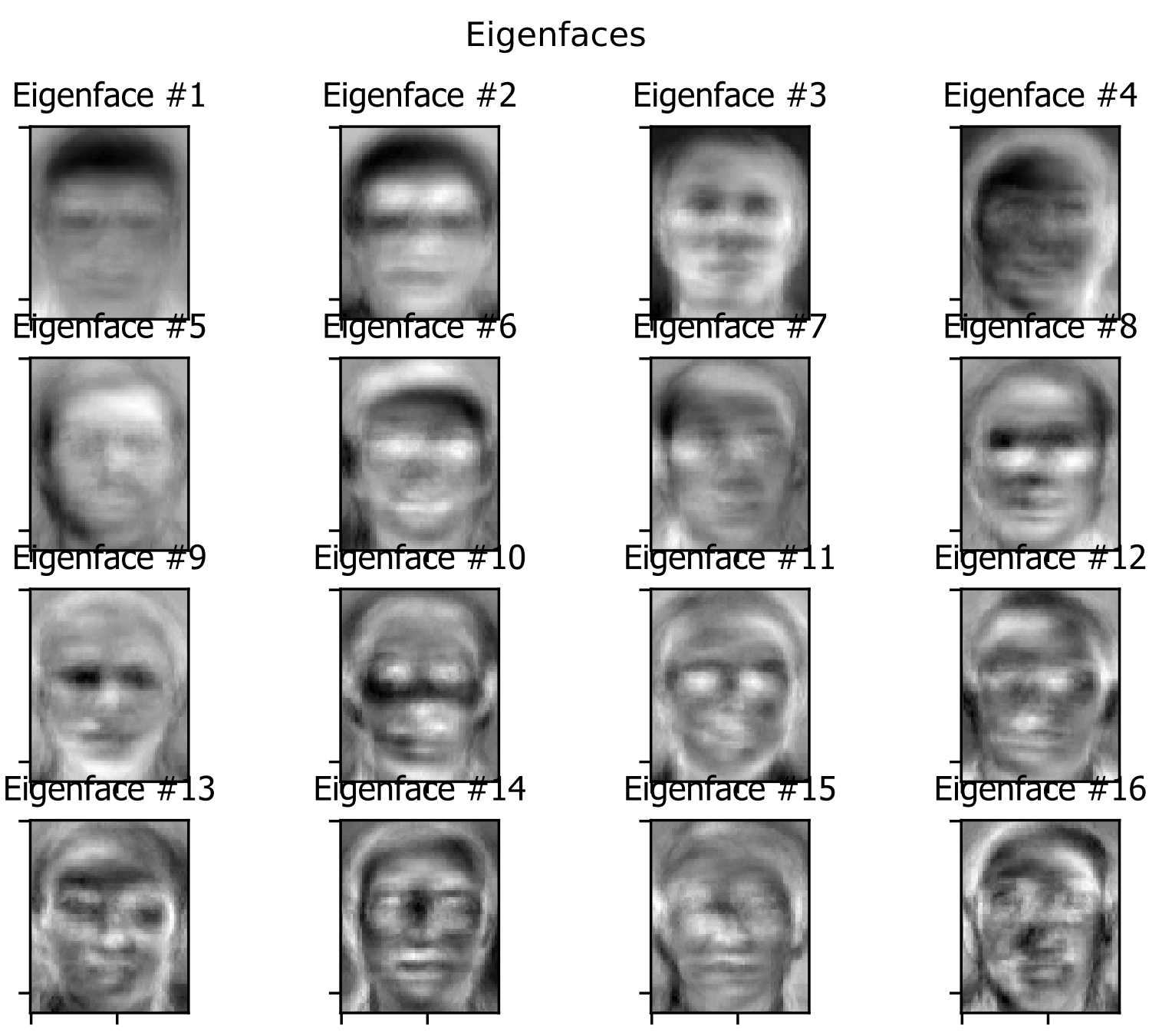
4. Order the eigenvectors descending by their eigenvalue. The k principal components are the eigenvectors corresponding to the k largest eigenvalues.

[*Source: Philipp Wagner, Face Recognition with Python* (code source available as well)]

PYTHON CODE

```
def pca(X):

    mu = X.mean(axis=0)
    X = X - mu
    C = np.dot(X.T,X)
    eigenvalues, eigenvectors = np.linalg.eigh(C)

    # sort eigenvectors descending by their eigenvalue
    idx = np.argsort(-eigenvalues)
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:,idx]
    return eigenvalues, eigenvectors, mu

# D: eigenvalues, W: eigenvectors, mu: mean, X: 40 X 10304 image array
D, W, mu = pca(X)

# plot the first 16 'eigenfaces'
images = []
for i in range(16):
    image = W[:,i].reshape(X[0].shape)
    images.append(normalize(image,0,255))

subplot(title="Eigenfaces", images=images, rows=4, cols=4)
```

AT&T Face dataset


Eigenfaces
Eigenface #1
Eigenface #2
Eigenface #3
Eigenface #4
Eigenface #5
Eigenface #6
Eigenface #7
Eigenface #8
Eigenface #9
Eigenface #10
Eigenface #11
Eigenface #12
Eigenface #13
Eigenface #14
Eigenface #15
Eigenface #16

PCA PROJECTION AND RECONSTRUCTION

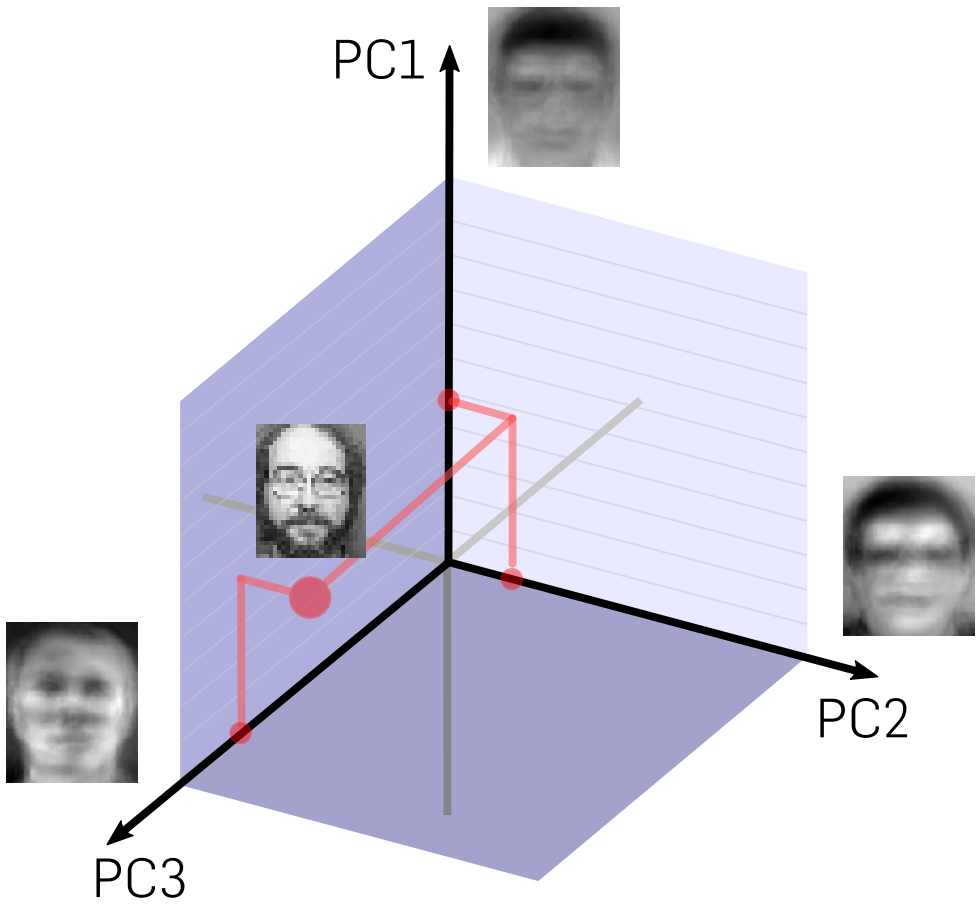
The k principal components of an observed vector $\mathbf{x}$ are then given by:

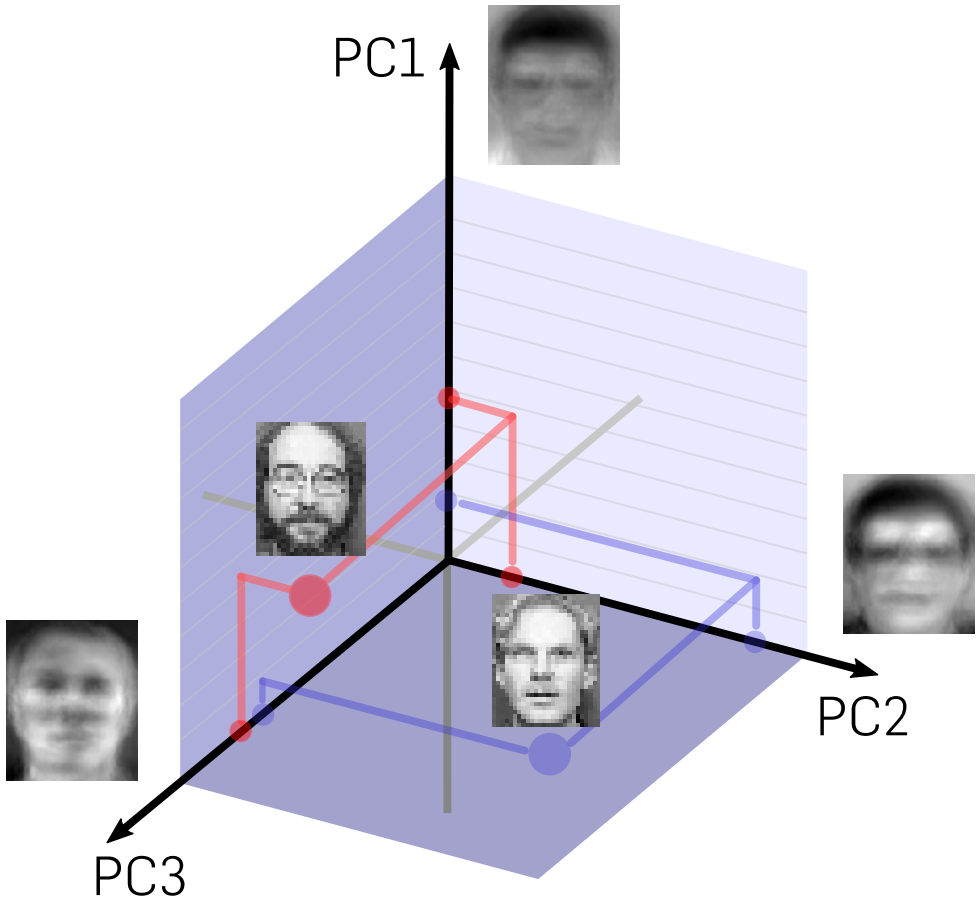
The image of a face!

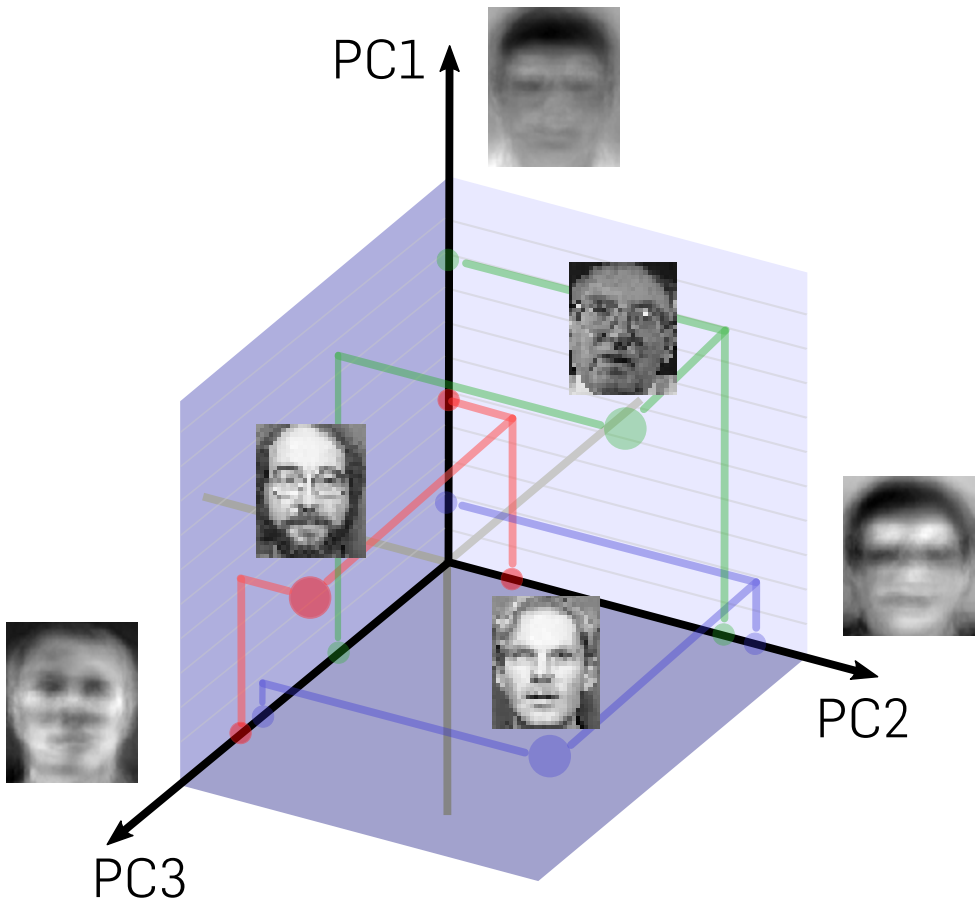
$$\mathbf{y} = W^T(\mathbf{x} - \mu)$$

where $W = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$.

The PCA basis


PC1
PC2
PC3


PC1
PC2
PC3


PC1
PC2
PC3

PCA PROJECTION AND RECONSTRUCTION

The k principal components of an observed vector $\mathbf{x}$ are then given by:

The image of a face!

$$\mathbf{y} = W^T(\mathbf{x} - \mu)$$

where $W = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$.

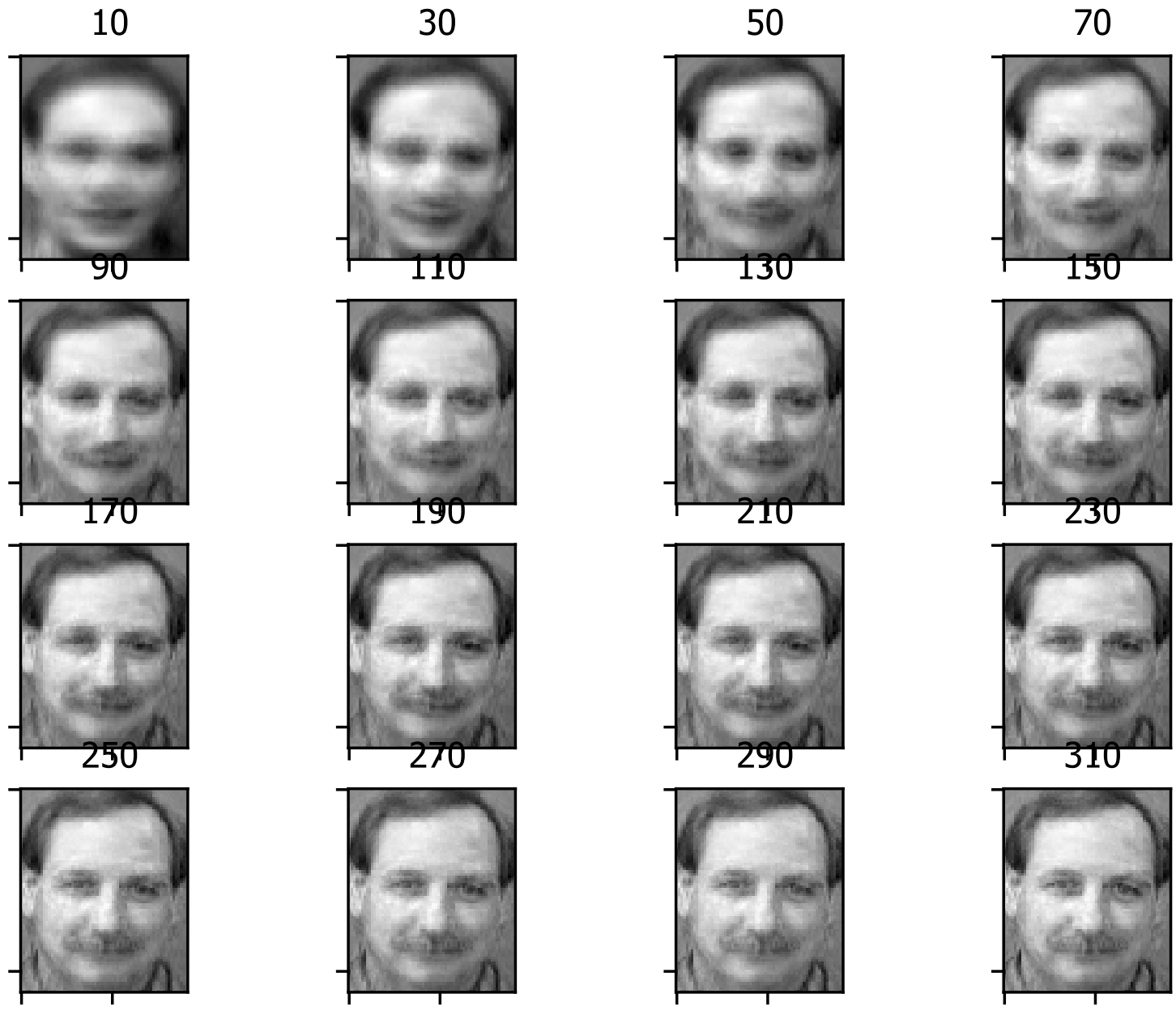
The PCA basis

The reconstruction from the PCA basis is given by:

$$\mathbf{x} = W \cdot \mathbf{y} + \mu$$

PYTHON CODE

```
def project(W, X, mu=None):
    if mu is None:
        return np.dot(X,W)
    return np.dot(X - mu, W)

def reconstruct(W, Y, mu=None):
    if mu is None:
        return np.dot(Y,W.T)
    return np.dot(Y, W.T) + mu

images = []
for nb_evs in range(10, 310, 20):
    P = project(W[:,0:nb_evs], X[0].reshape(1,-1), mu)
    R = reconstruct(W[:,0:nb_evs], P, mu)

    R = R.reshape(X[0].shape)
    images.append(normalize(R,0,255))

subplot(title="Reconstruction of one face", images=images, rows=4, cols=4)
```


Reconstruction of one face
10
30
50
70
90
110
130
150
170
190
210
230
250
270
290
310

WHY IS IT USEFUL?

Original images: $\textit{dim}(\mathbf{x}) = 92 \times 112 = 10304$ pixels: large number of dimensions!

$\Rightarrow$ difficult to tell whether 2 images represent the same person (i.e. ***classify*** them).

WHY IS IT USEFUL?

Original images: $\textit{dim}(\mathbf{x}) = 92 \times 112 = 10304$ pixels: large number of dimensions!

$\Rightarrow$ difficult to tell whether 2 images represent the same person (i.e. ***classify*** them).

With the PCA, we project our test image onto a PCA basis of k principal components: $\mathbf{y} = W^T(\mathbf{x} - \mu)$ with $W = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$.

$\textit{dim}(\mathbf{y}) = k$ is much smaller than $\textit{dim}(\mathbf{x})$

WHY IS IT USEFUL?

Original images: $dim(\mathbf{x}) = 92 \times 112 = 10304$ pixels: large number of dimensions!

$\Rightarrow$ difficult to tell whether 2 images represent the same person (i.e. *classify* them).

With the PCA, we project our test image onto a PCA basis of k principal components: $\mathbf{y} = W^T(\mathbf{x} - \mu)$ with $W = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$.

$dim(\mathbf{y}) = k$ is much smaller than $dim(\mathbf{x})$

We effectively "summarize" our image into a few key values, along the principal axes of variation of our dataset.

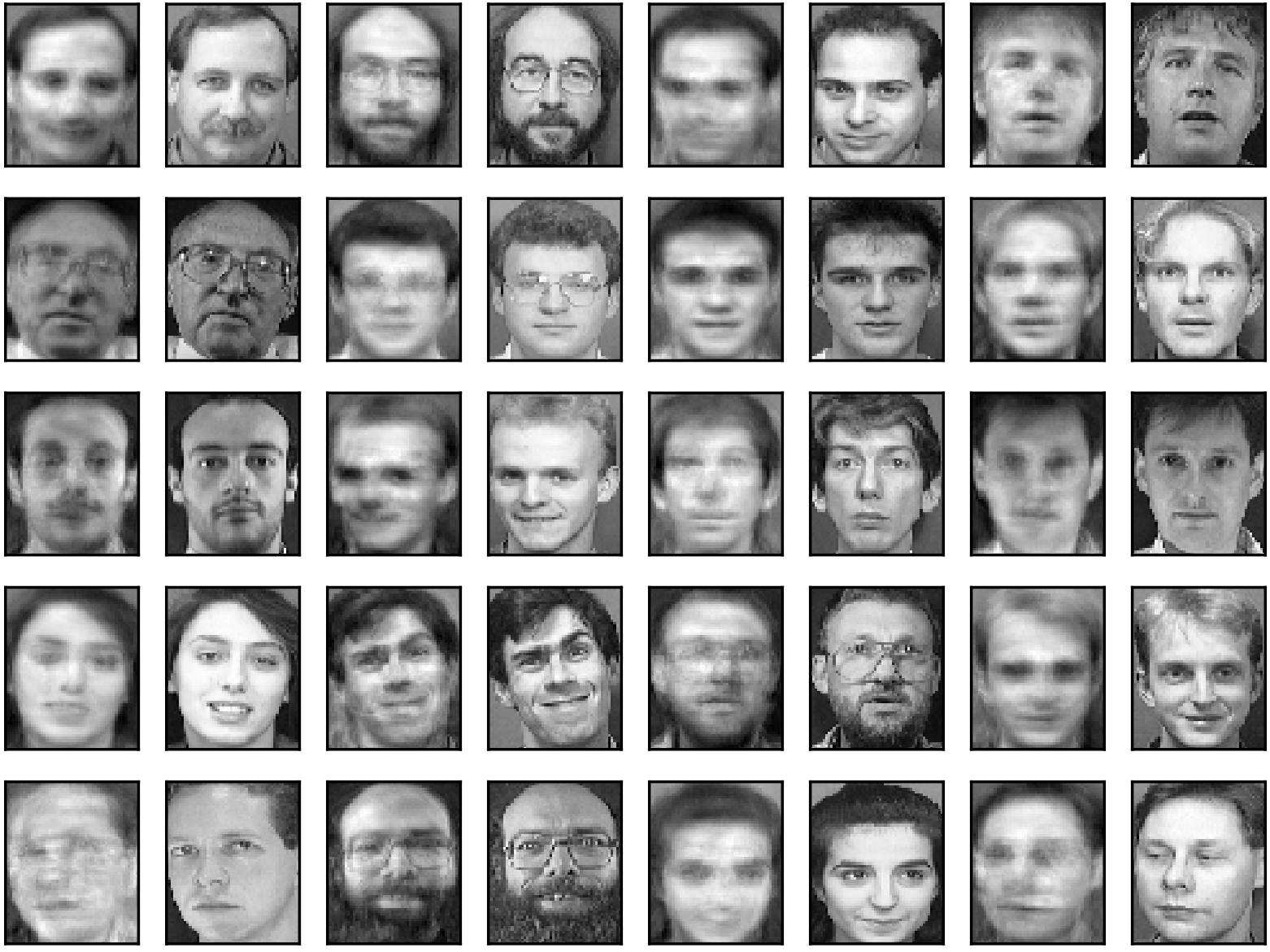
$\Rightarrow$ these values discriminate effectively amongst our images

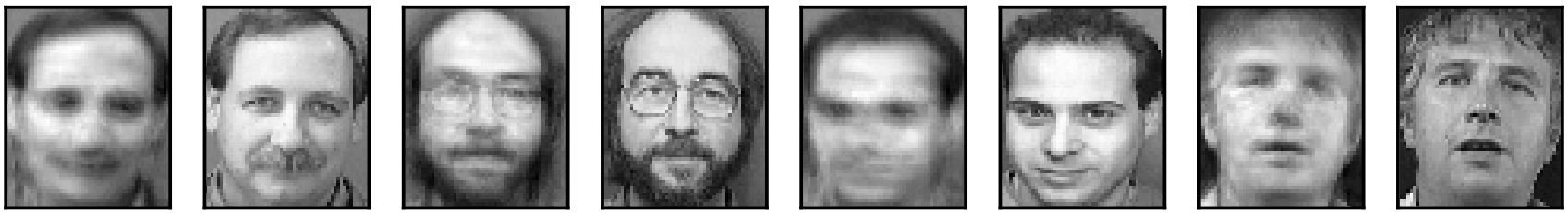
$\Rightarrow$ **Well suited for classification!**


Reconstruction with 1 Eigenvectors

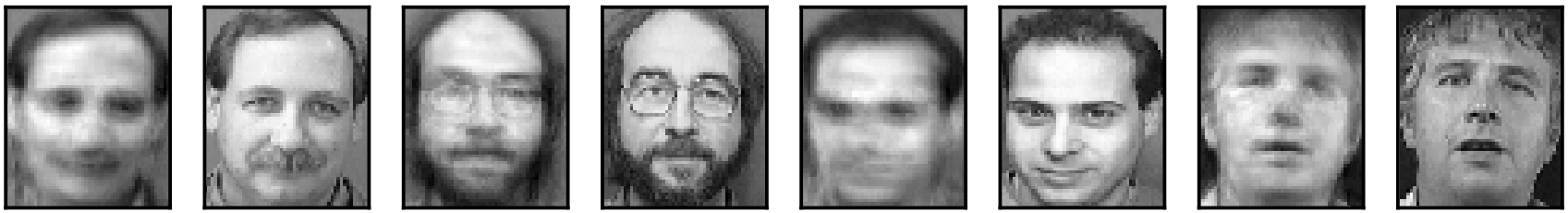
Reconstruction with 10 Eigenvectors

Reconstruction with 50 Eigenvectors





Remember: these faces are reconstructed from 50 values (to be compared to the 10304 values required for the original photos).

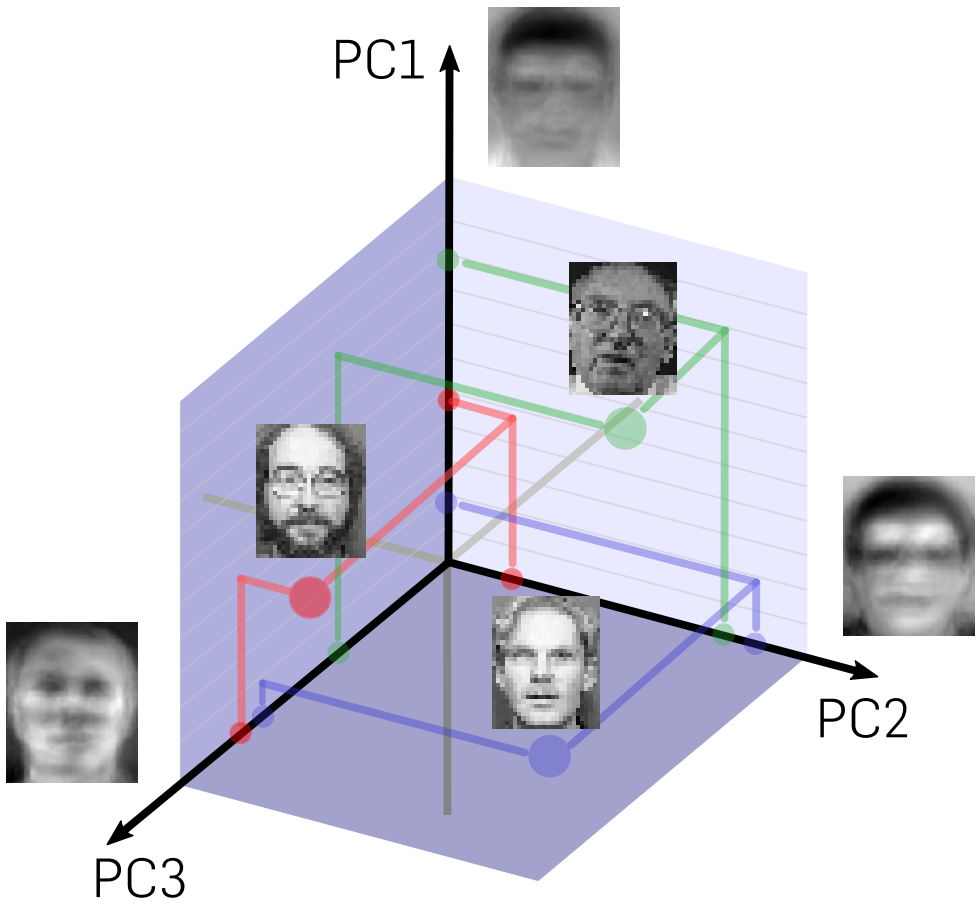


Remember: these faces are reconstructed from 50 values (to be compared to the 10304 values required for the original photos).

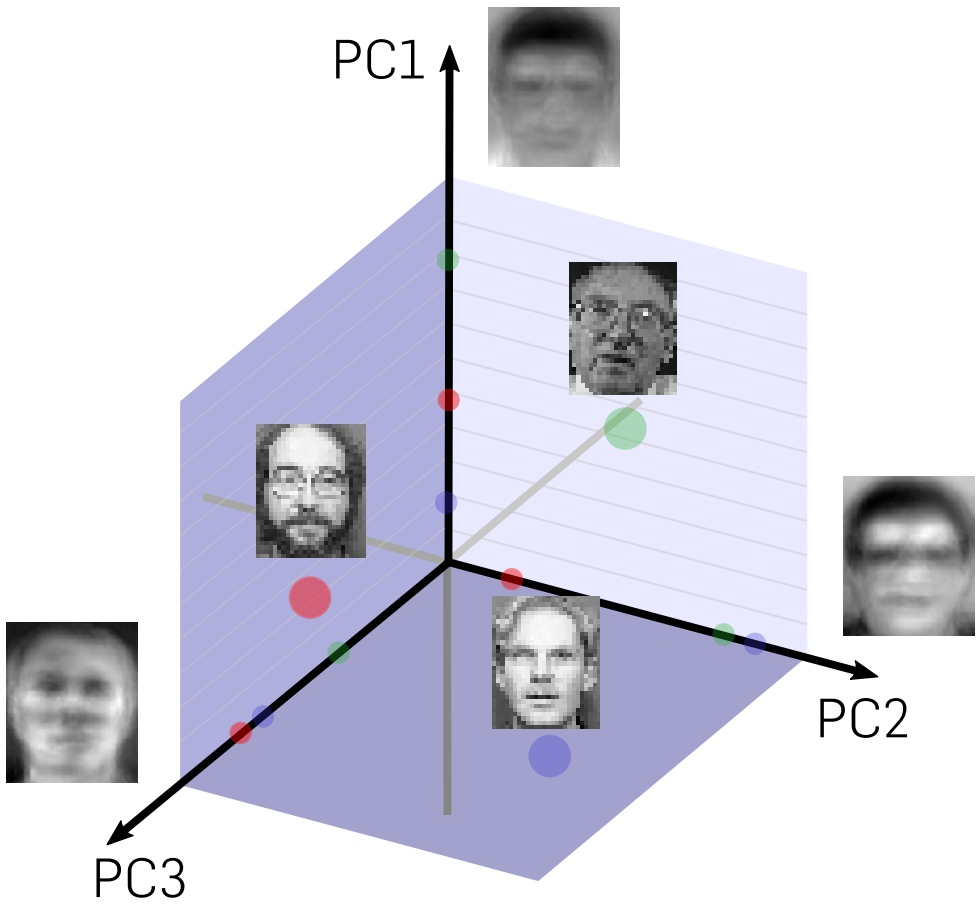
PCA is often used as a **dimensionality reduction** technique (i.e. a kind of data lossy data compression).

FACE RECOGNITION

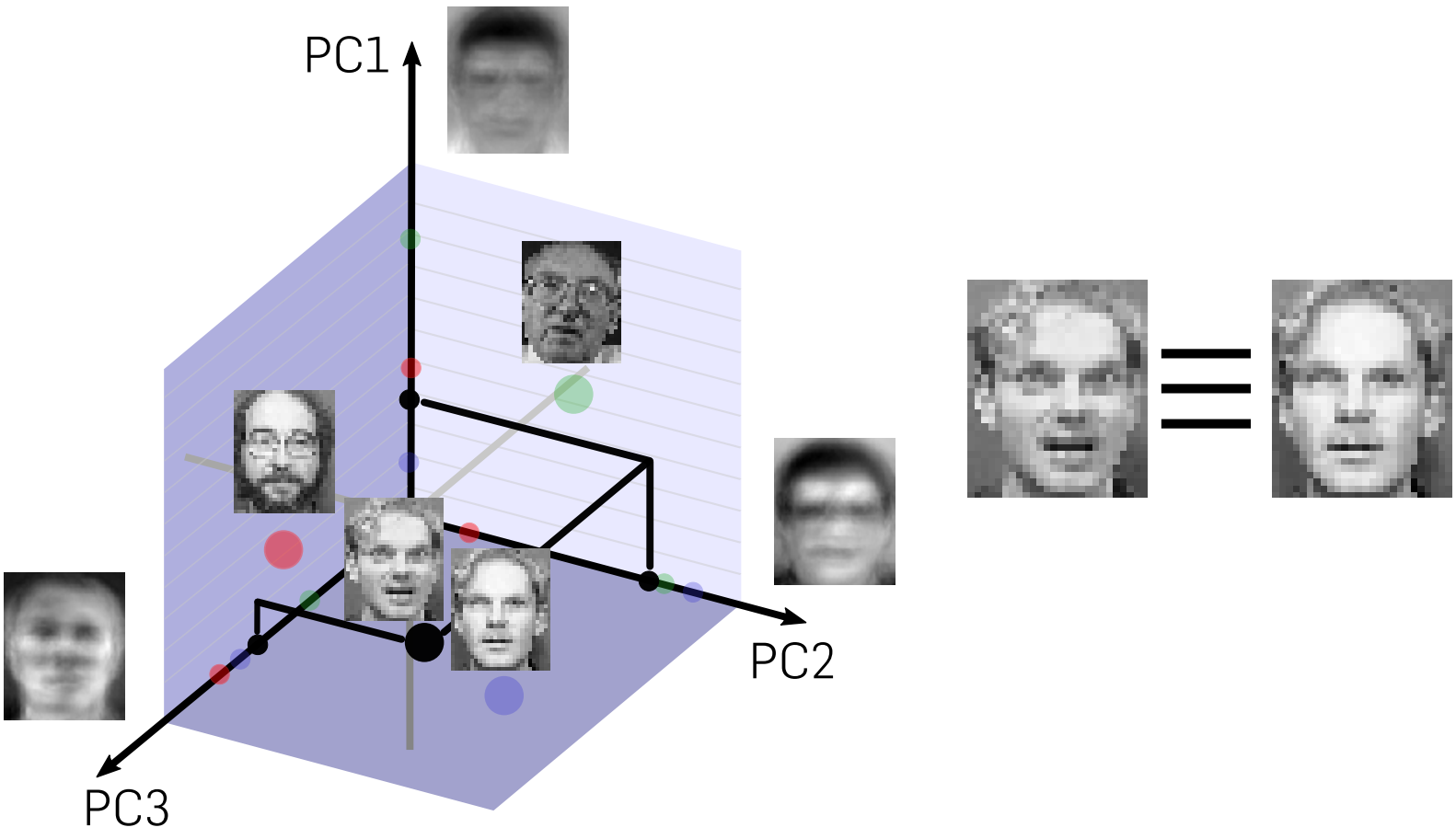
?


PC1
PC2
PC3

?


PC1
PC2
PC3

?


PC1
PC2
PC3

RECOGNITION

1. **learn a model** by projecting the training set onto the PCA basis
2. **project the test image** as well
3. **find the 1-nearest neighbour**

PYTHON CODE

```
def dist(p, q):
    p = np.asarray(p).flatten()
    q = np.asarray(q).flatten()
    return np.sqrt(np.sum(
                    np.power((p-q),2)
                    ))


def learn_model(X):
    D, W, mu = pca(X, nb_evs=10)
    # compute projections
    projections = []
    for xi in X:
        yi = project(W,
                     xi.reshape(1,-1),
                     mu)
        projections.append(yi)

    return W, projections
```

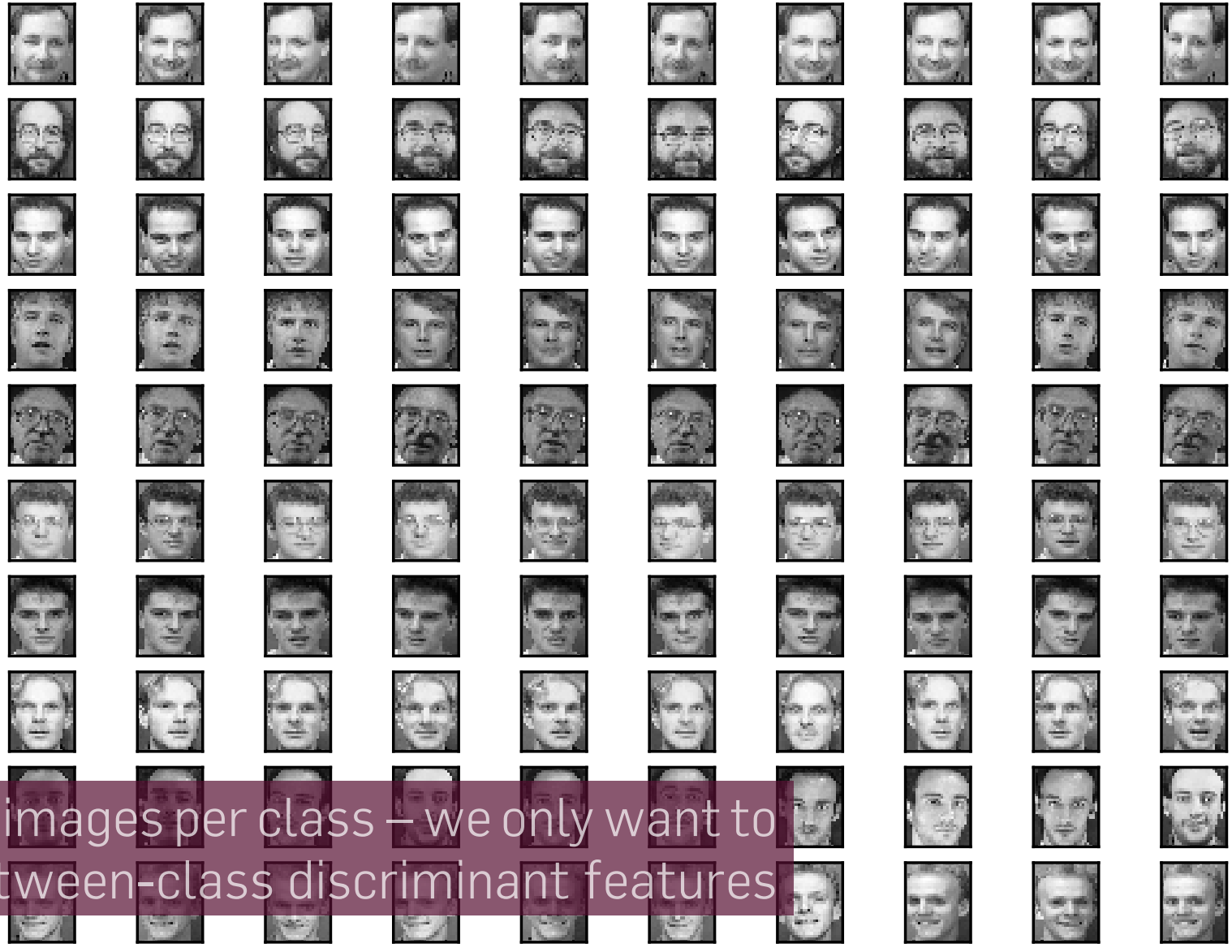
```
def predict(X, W, projections):
    minDist = np.finfo('float').max
    minClass = -1
    Q = project(W, X.reshape(1,-1), mu)

    for i in range(len(projections)):
        dist = dist(projections[i], Q)
        if dist < minDist:
            minDist = dist
            faceClass = faceClasses[i]
    return faceClass


X, faceClasses = read_images()
W, projections = learn_model(X)
predict(test_image, W, projections)
```

LIMITS OF THE PCA APPROACH (EIGENFACES)

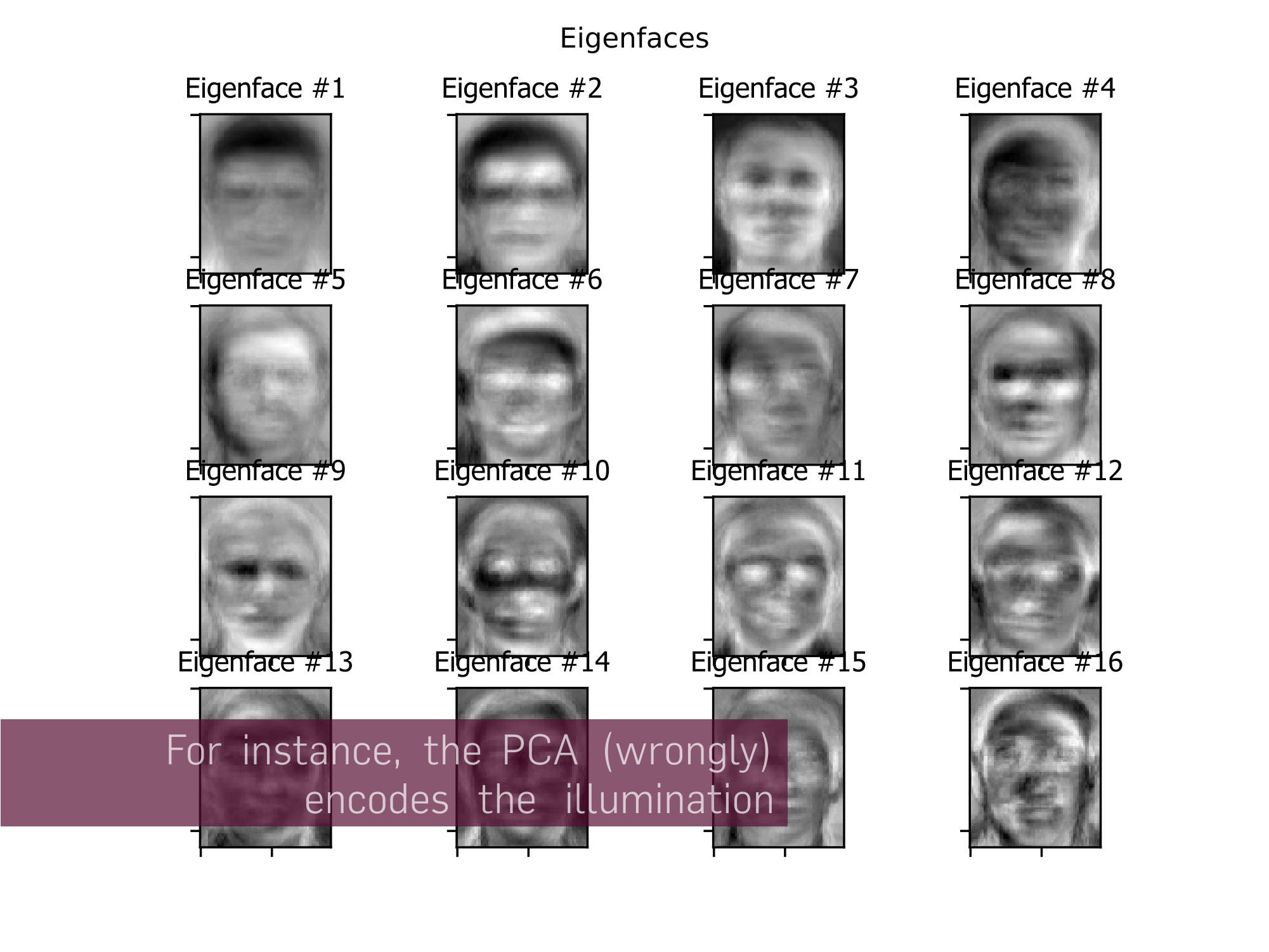
PCA tries to find a combination of linear features that maximizes the total variance (i.e. the “axes of maximum variation”).

No concept of class!

AT&T Face dataset



Here, 10 images per class – we only want to learn between-class discriminant features


Eigenfaces
Eigenface #1
Eigenface #2
Eigenface #3
Eigenface #4
Eigenface #5
Eigenface #6
Eigenface #7
Eigenface #8
Eigenface #9
Eigenface #10
Eigenface #11
Eigenface #12
Eigenface #13
Eigenface #14
Eigenface #15
Eigenface #16
For instance, the PCA (wrongly) encodes the illumination

LIMITS OF THE PCA APPROACH (EIGENFACES)

⇒ Linear Discriminant Analysis (LDA) (and the corresponding *Fischerfaces*)

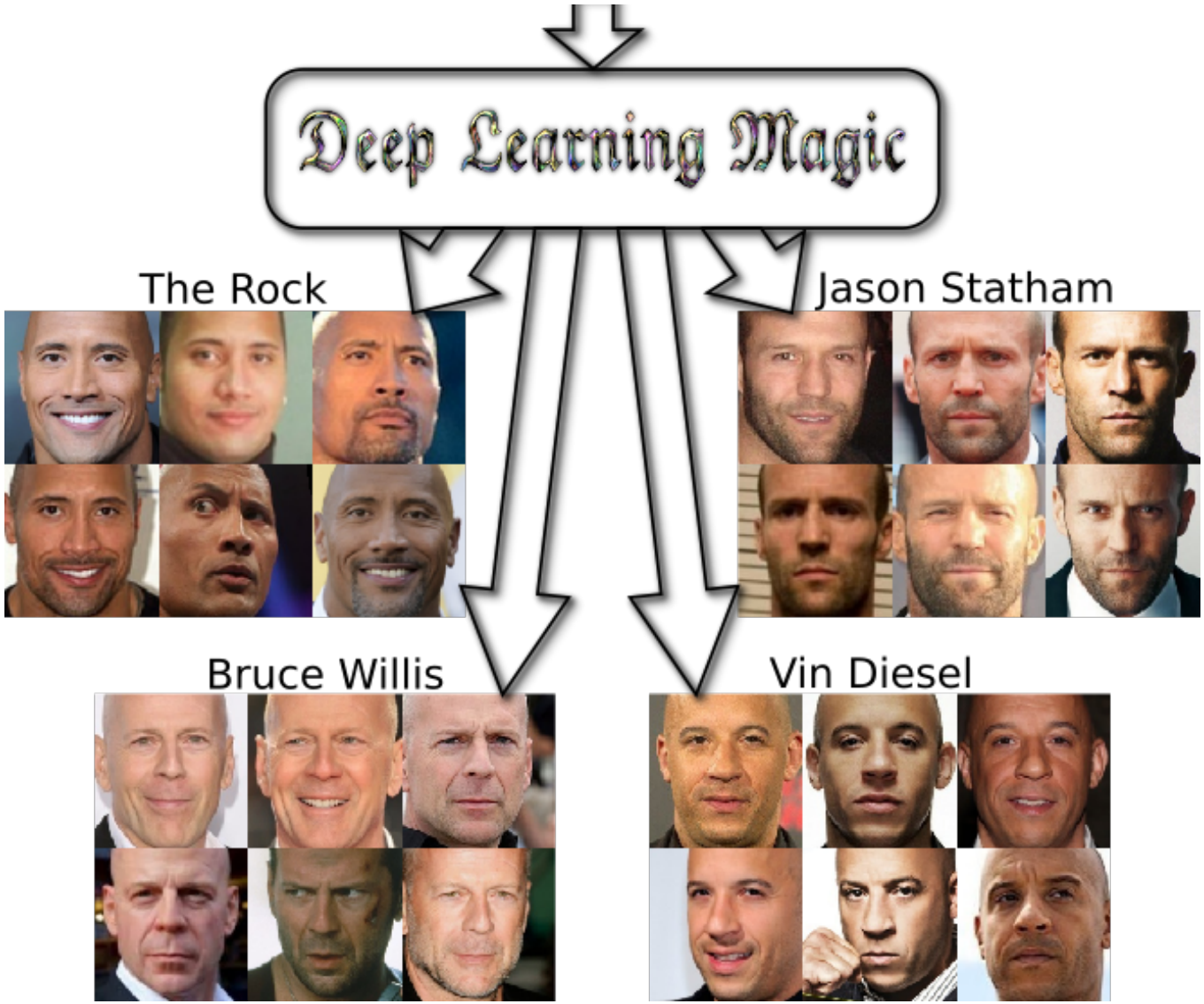
LDA tries to find a combination of linear features that maximizes the ratio of between-classes to within-classes scatter.

[*Source: Belhumeur, Hespanha and Kriegman, Eigenfaces vs. fisherfaces: Recognition using class specific linear projection*]

STATE OF THE ART FACE RECOGNITION

[*Source:* ***dlib*** face recognition]

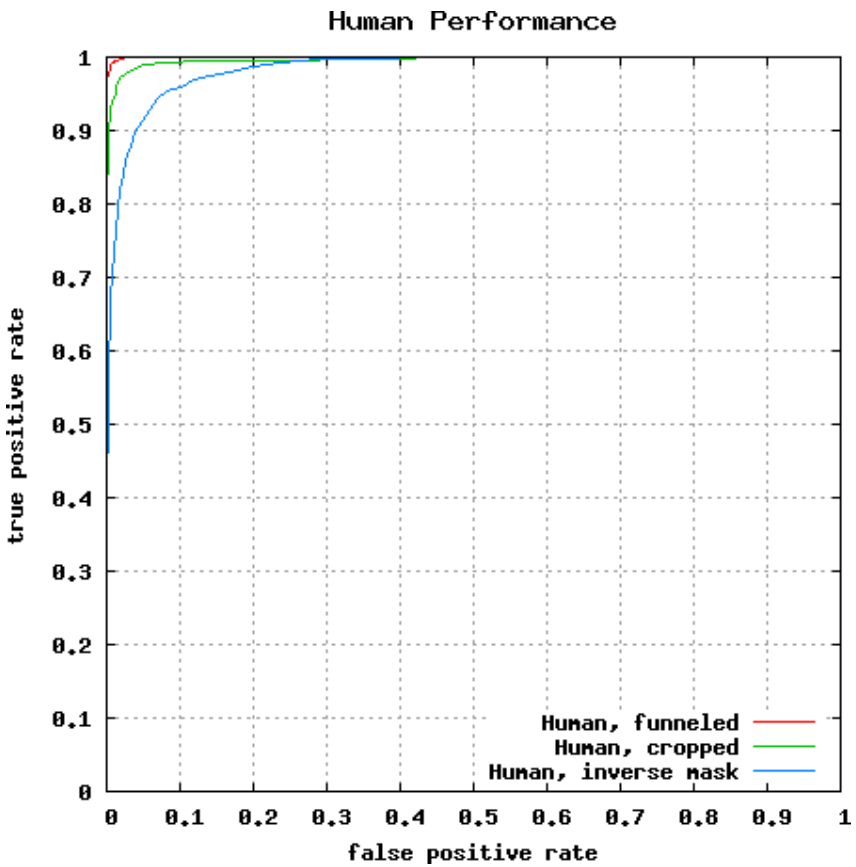
STATE OF THE ART FACE RECOGNITION

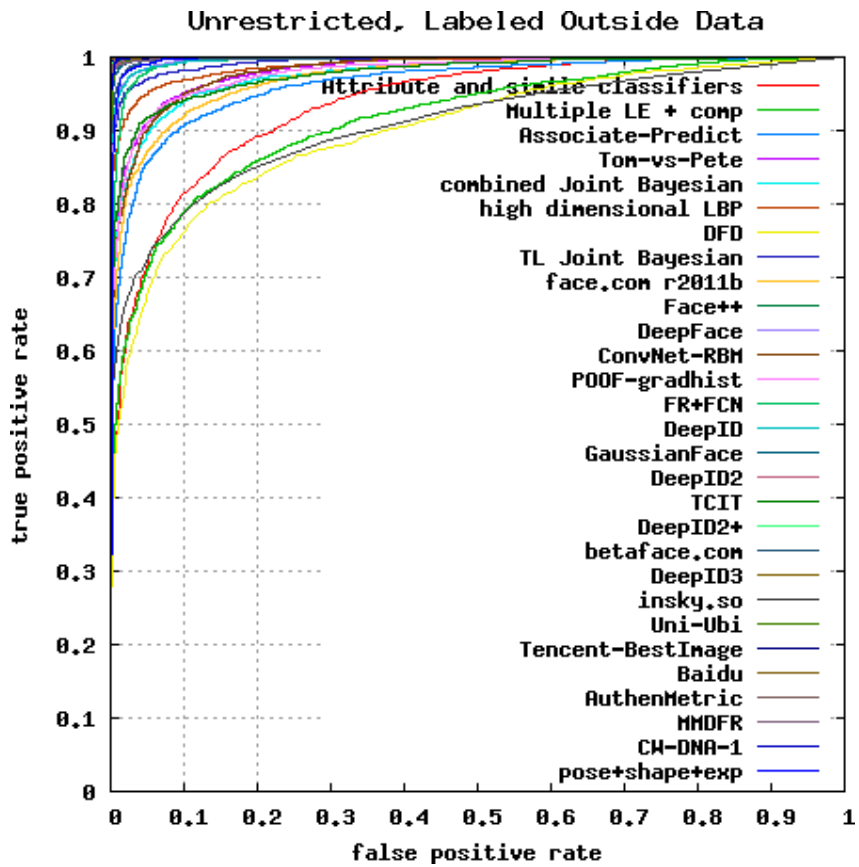


[*Source:* ***dlib*** face recognition]

FACE RECOGNITION: HUMANS VS MACHINES

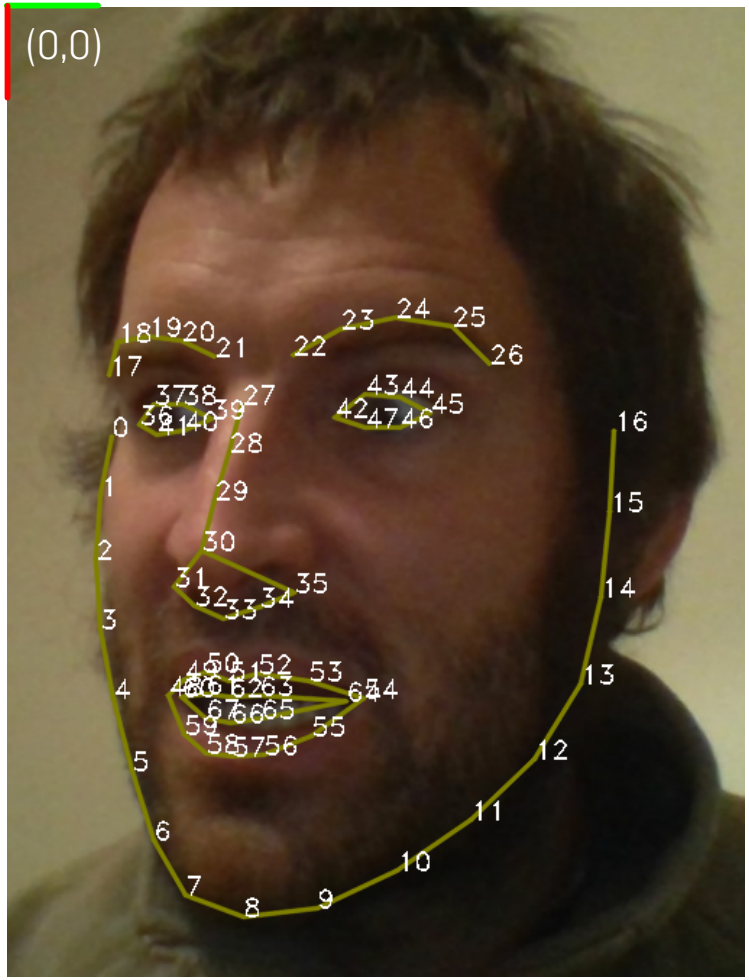
Standard dataset: Labelled Face in the Wild (LFW) (13233 images and 5749 people)





6D HEAD POSE ESTIMATION

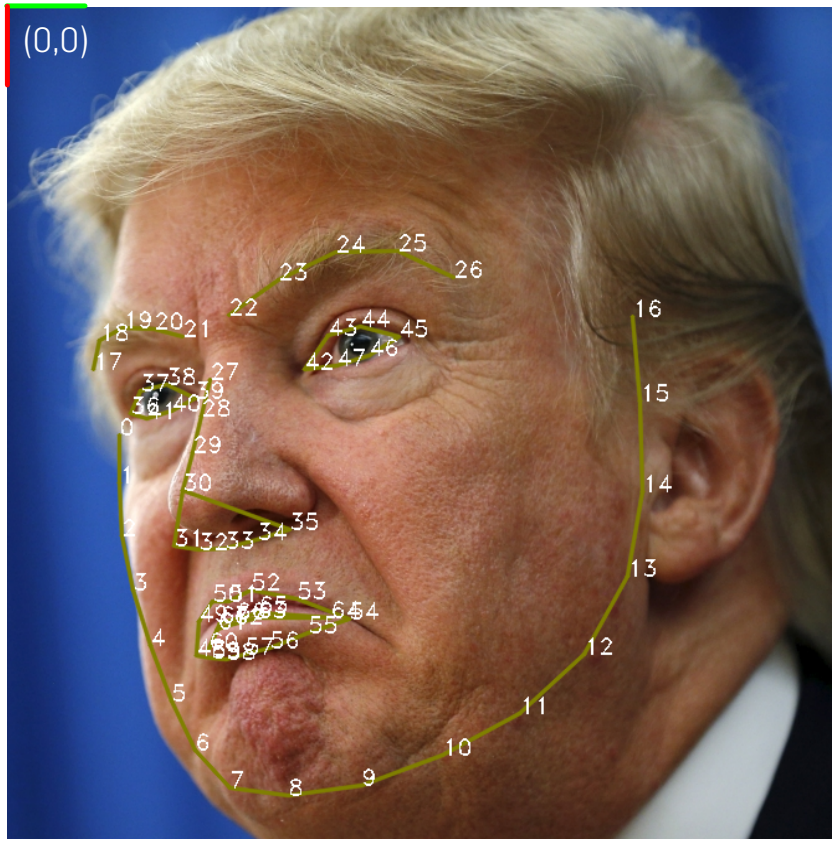
FACIAL LANDMARKS



`dlib` or OpenPose/OpenFace

[Cao X., Wei Y., Wen F., Sun J., **Face Alignment by Explicit Shape Regression**, CVPR 2012]

FACIAL LANDMARKS

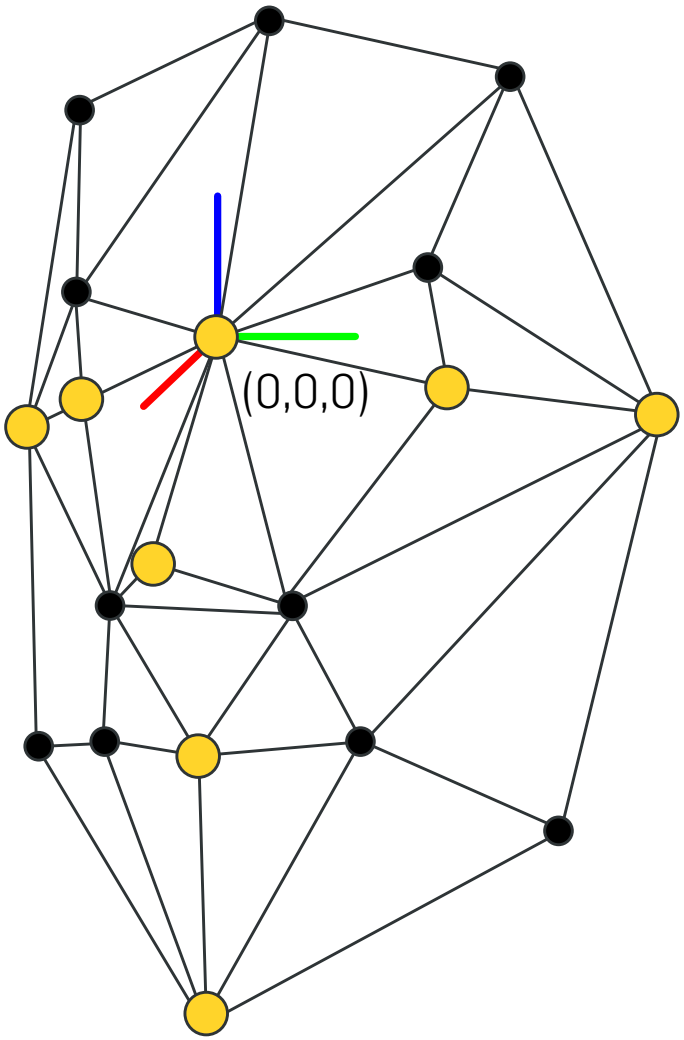


`dlib` or OpenPose/OpenFace

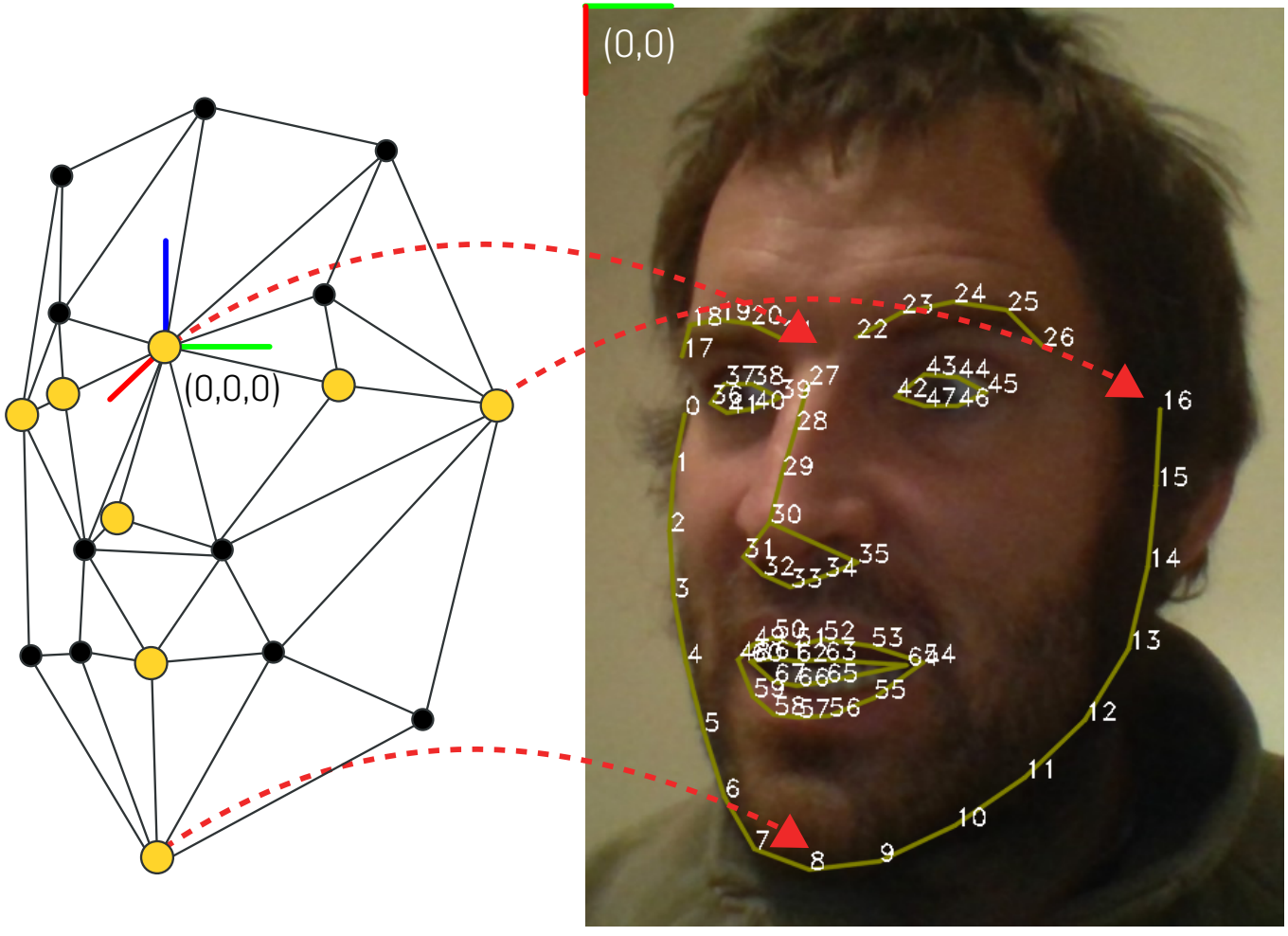
[Cao X., Wei Y., Wen F., Sun J., **Face Alignment by Explicit Shape Regression**, CVPR 2012]


Club
open me

FACE MASK




```
// anthropometry values taken from
// https://en.wikipedia.org/wiki/Human_head
// (in mm)
Point3f SELLION(0., 0.,0.);
Point3f RIGHT_EYE(-20., -65.5,-5.);
Point3f LEFT_EYE(-20., 65.5,-5.);
Point3f RIGHT_EAR(-100., -77.5,-6.);
Point3f LEFT_EAR(-100., 77.5,-6.);
Point3f NOSE(21.0, 0., -48.0);
Point3f STOMMION(10.0, 0., -75.0);
Point3f MENTON(0., 0.,-133.0);
```



`dlib` or OpenPose/OpenFace + OpenCV

HEAD POSE ESTIMATION IN C++

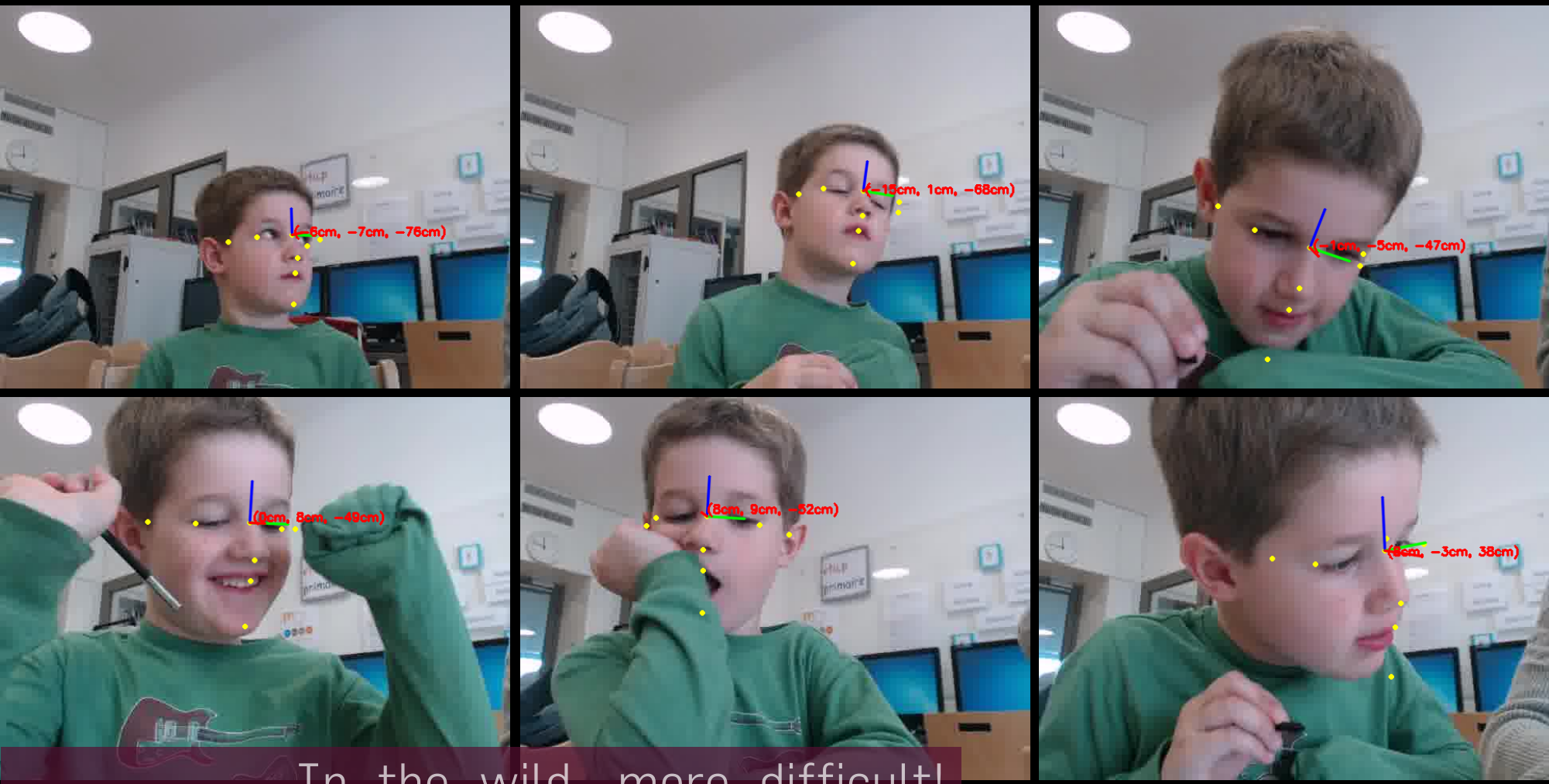
```
cv::Mat projectionMat = cv::Mat::zeros(3,3,CV_32F);
cv::Matx33f projection = projectionMat;
projection(0,0) = focalLength;
projection(1,1) = focalLength;
projection(0,2) = opticalCenterX;
projection(1,2) = opticalCenterY;
projection(2,2) = 1;

std::vector<Point3f> head_points;

head_points.push_back(SELLION);
head_points.push_back(RIGHT_EYE);
// ...

std::vector<Point2f> detected_points;

detected_points.push_back(/* x,y of sellion in img */));
detected_points.push_back(/* x,y of right eye in img*/));
// ...
```

Source: gazr

HEAD POSE ESTIMATION IN C++

```
// Initializing the head pose 1m away, roughly facing the robot
// This initialization is important as it prevents solvePnP to find the
// mirror solution (head *behind* the camera)
Mat tvec = (Mat_<double>(3,1) << 0., 0., 1000.);
Mat rvec = (Mat_<double>(3,1) << 1.2, 1.2, -1.2);

// Find the 3D pose of our head
solvePnP(head_points, detected_points,
        projection, noArray(),
        rvec, tvec, true,
        cv::SOLVEPNP_ITERATIVE);

Matx33d rotation;
Rodrigues(rvec, rotation);

Matx44d pose = {
 rotation(0,0), rotation(0,1), rotation(0,2), tvec.at<double>(0)/1000,
 rotation(1,0), rotation(1,1), rotation(1,2), tvec.at<double>(1)/1000,
 rotation(2,0), rotation(2,1), rotation(2,2), tvec.at<double>(2)/1000,
             0,             0,             0,                      1
};
```

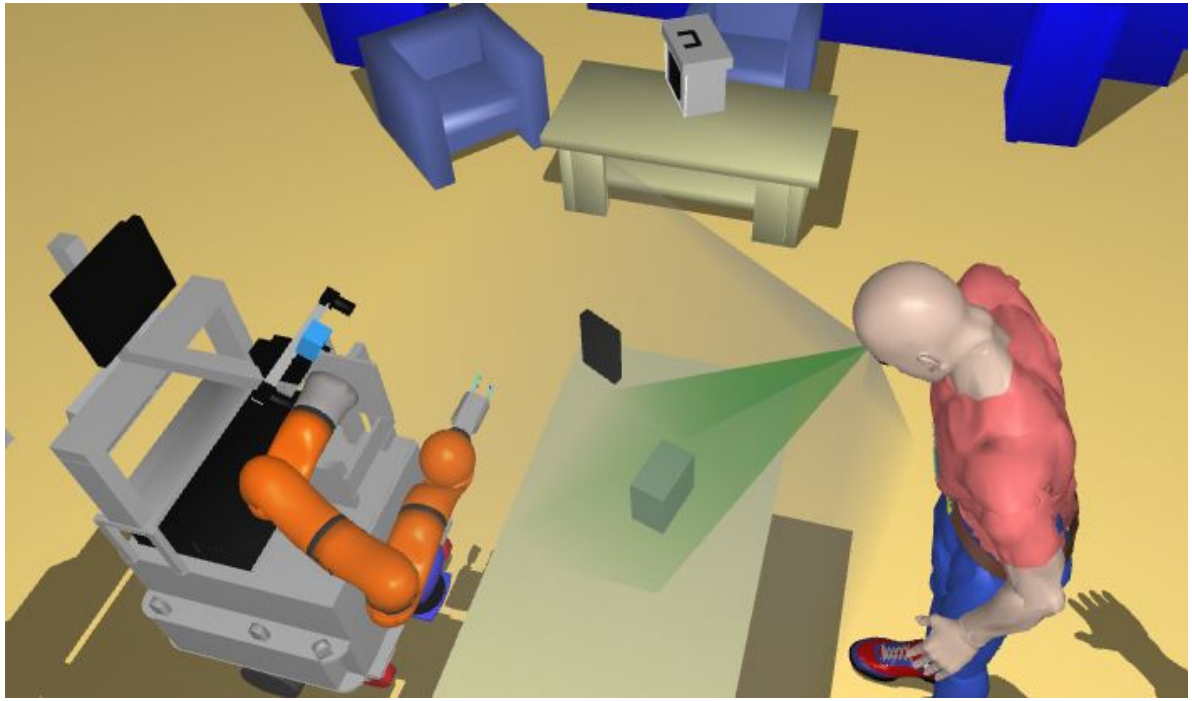
Source: gazr



In the wild, more difficult!

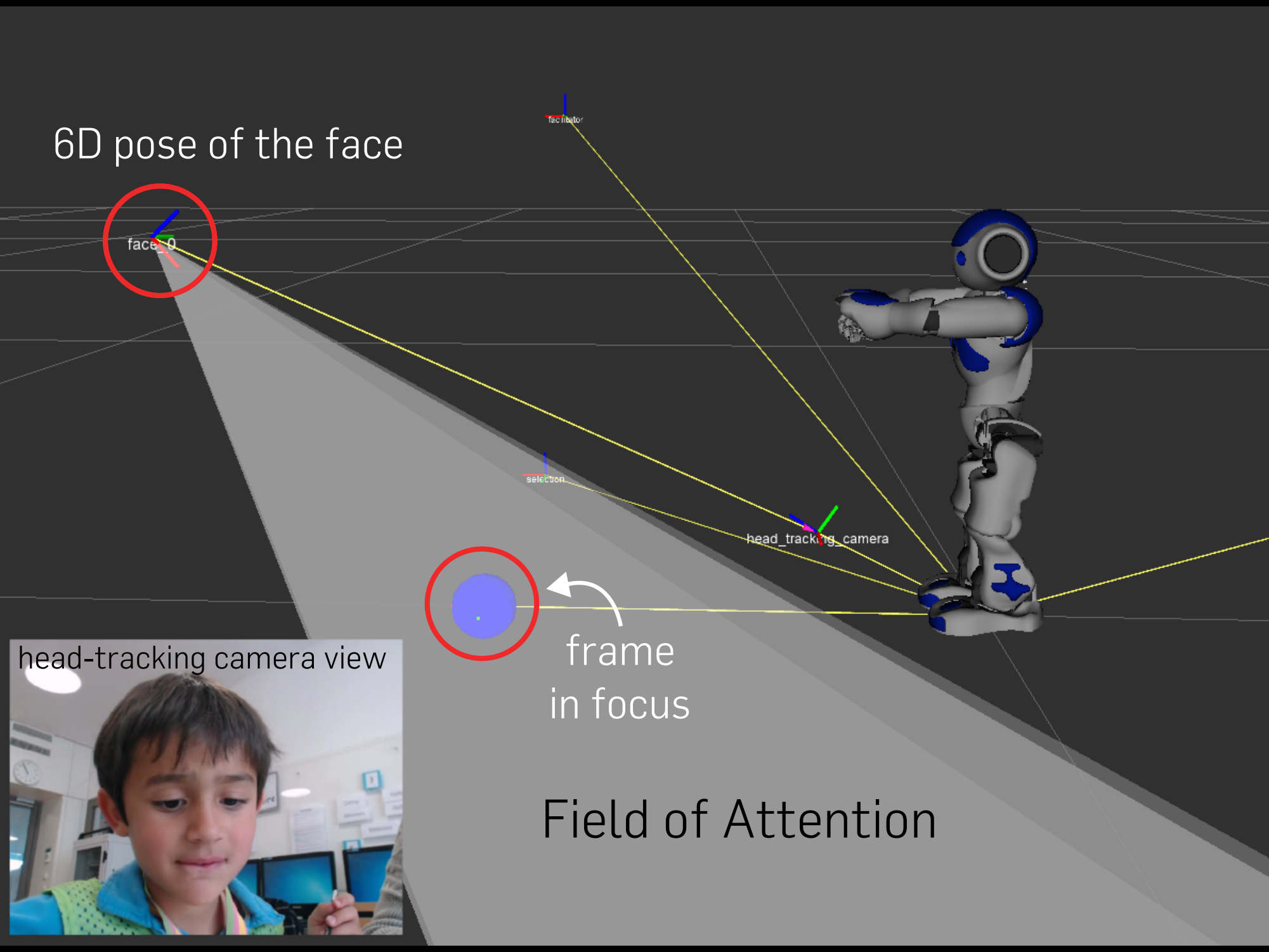
ATTENTION TRACKING

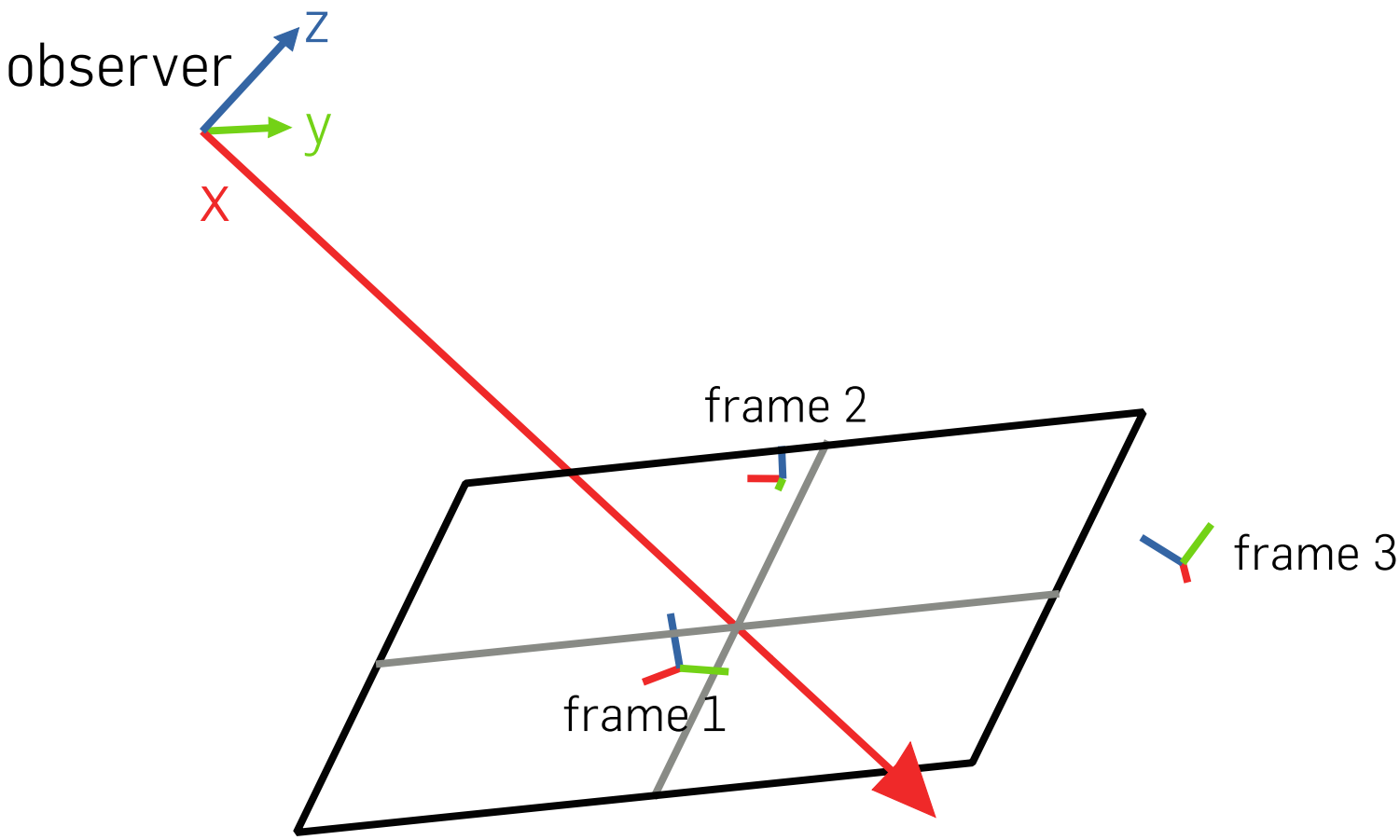
FIELD OF VIEW VS FIELD OF ATTENTION

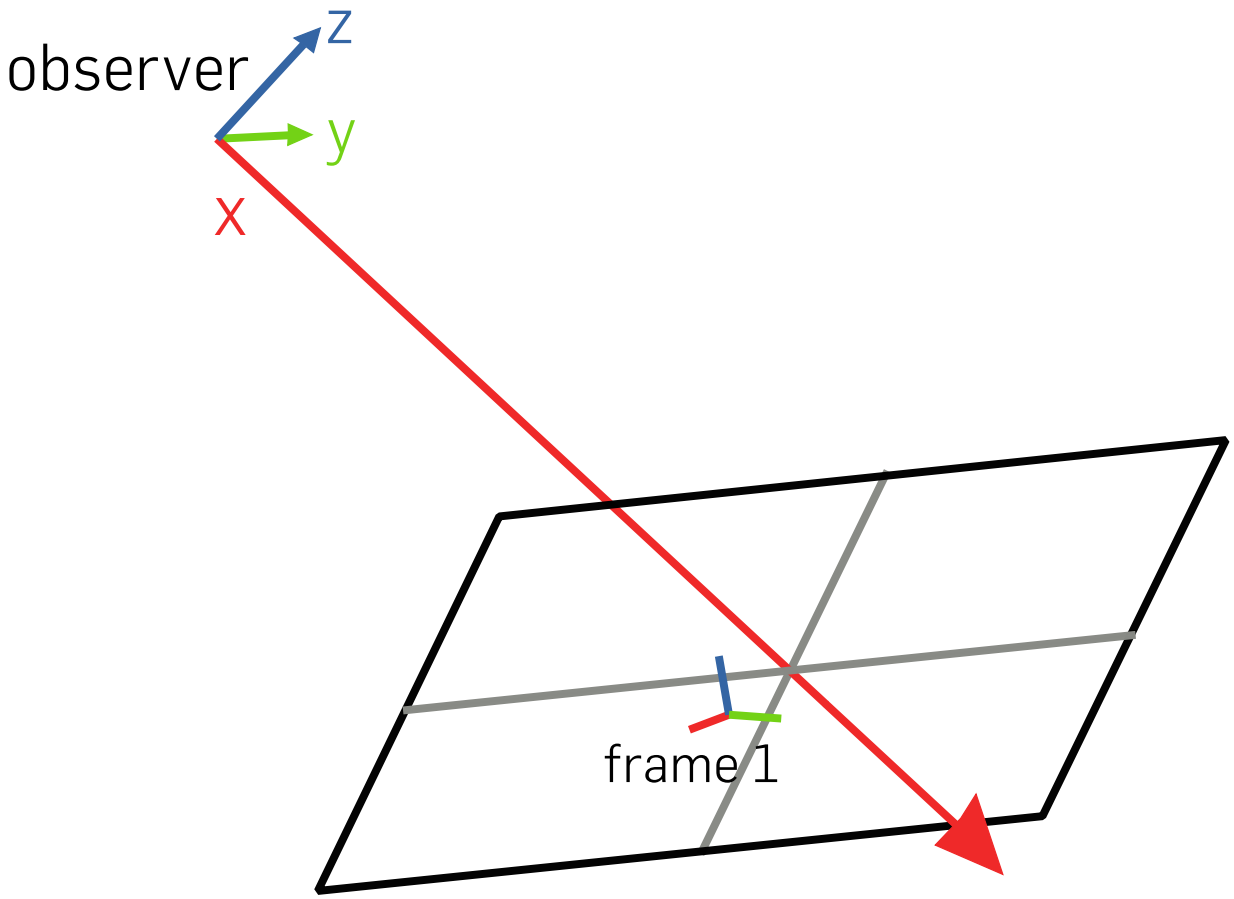


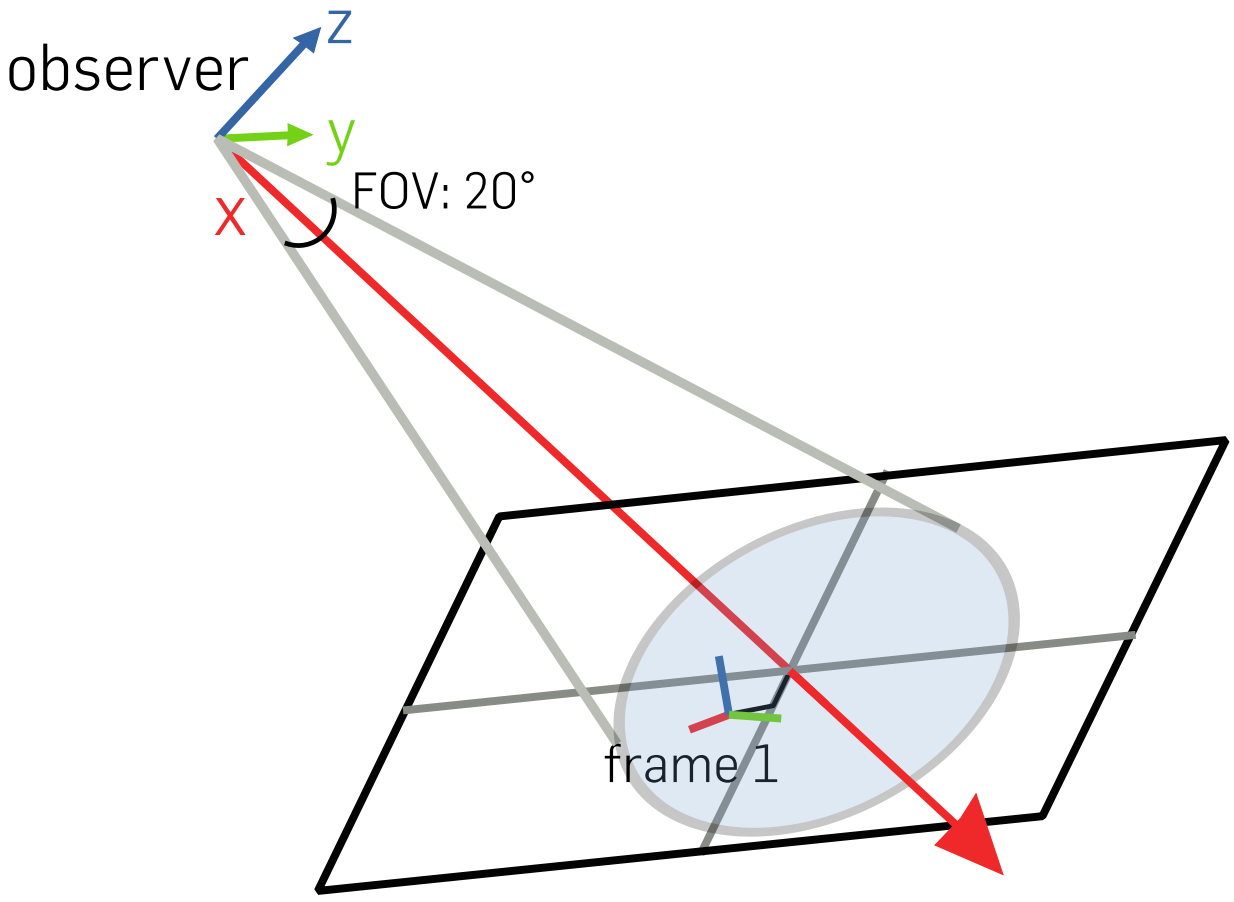
Type	horizontal	vertical
FoV (Holmqvist 2011)	±40°	±25°
FoV (Walker 1980)	±95°	+60°, −75°
FoA (Sisbot 2011)	±30°	±30°
FoA (Lemaignan 2016)	±40°	±40°

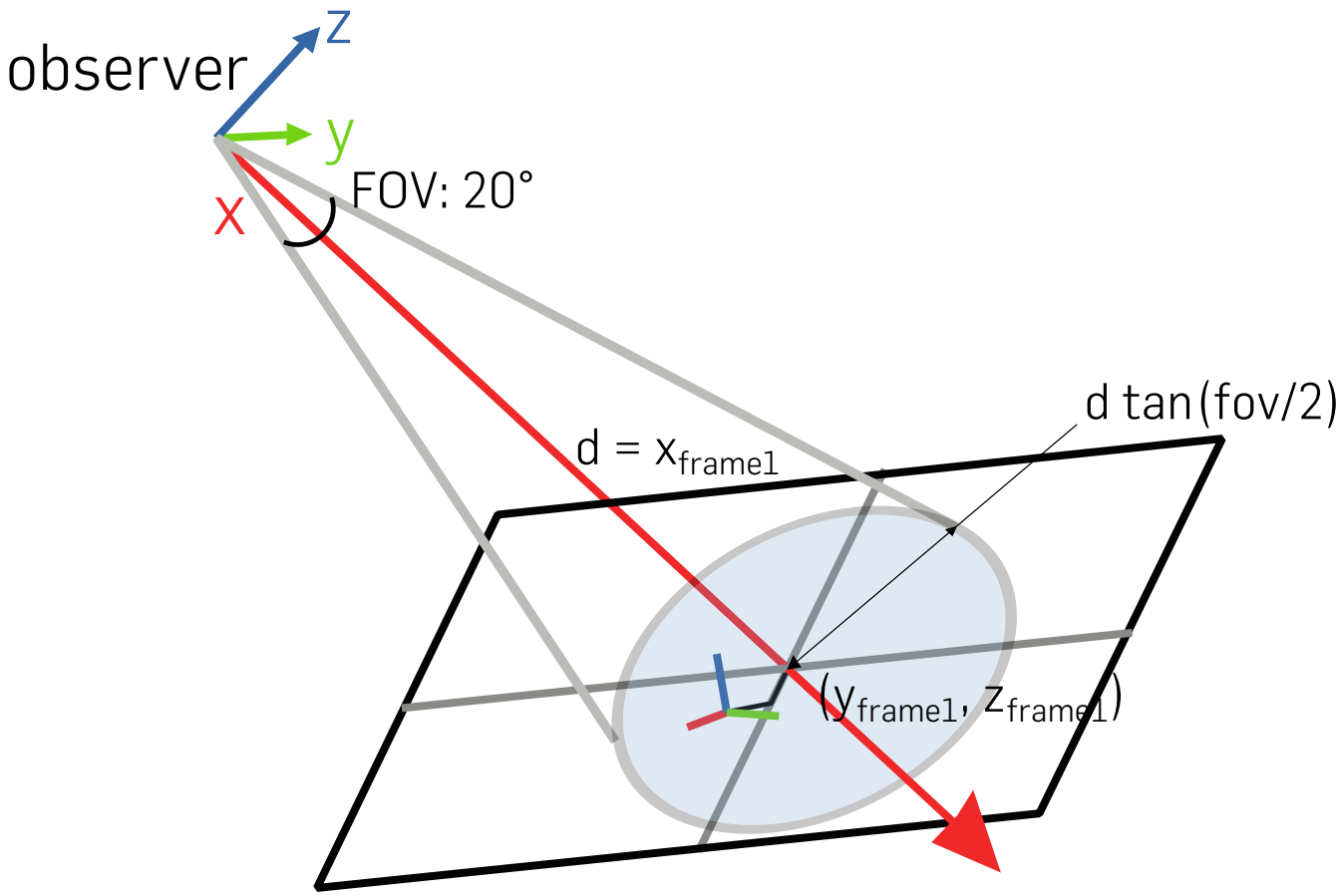
[*Source: S. Lemaignan; F. Garcia; A. Jacq; P. Dillenbourg; From Real-time Attention Assessment to "With-me-ness" in Human-Robot Interaction*]


6D pose of the face
face_0
head_tracking_camera
frame
in focus
head-tracking camera view
Field of Attention


observer
z
y
x
frame 2
frame 3
frame 1


observer
z
y
x
frame 1

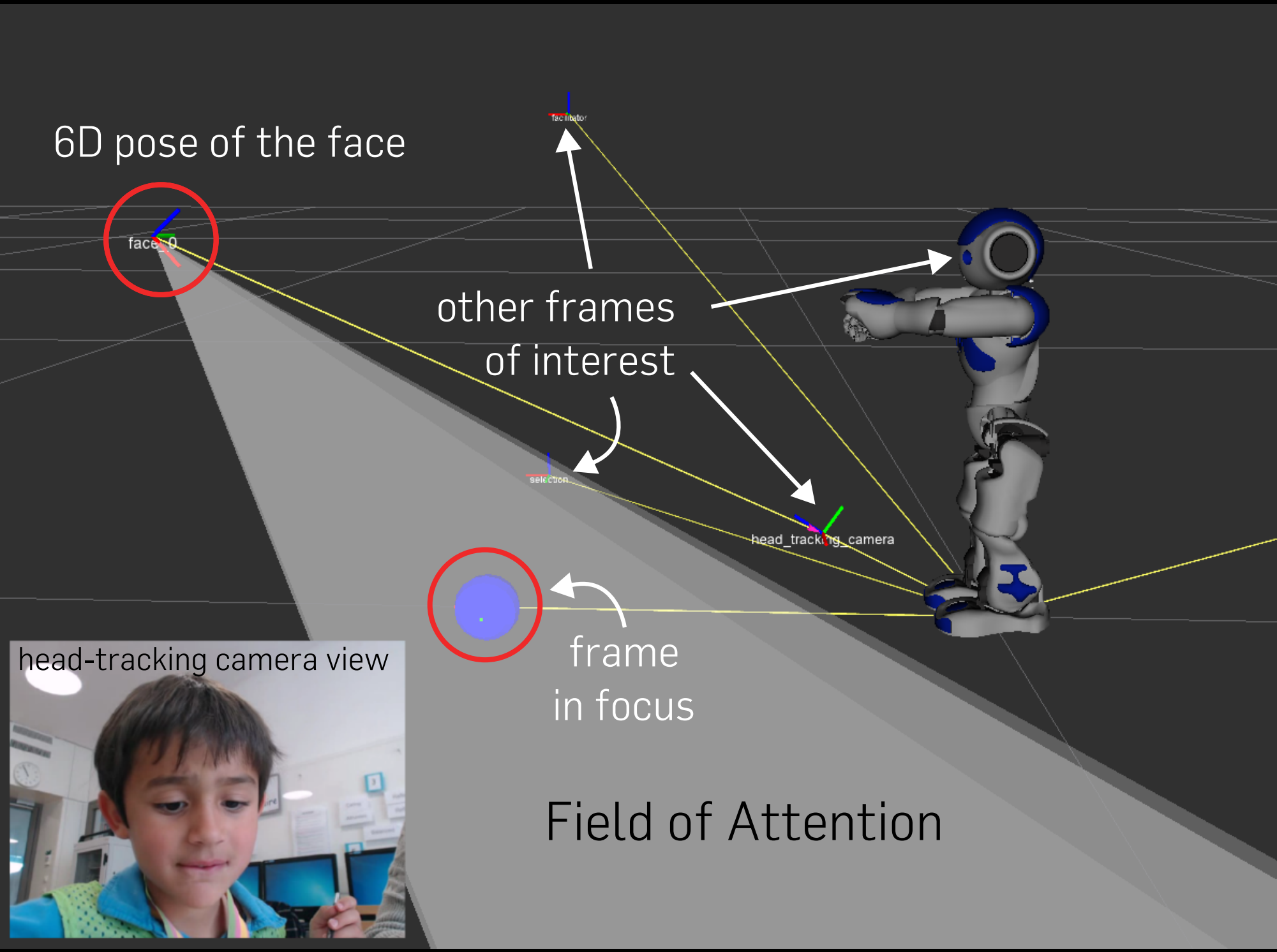

observer
z
y
x
FOV: 20°
frame 1

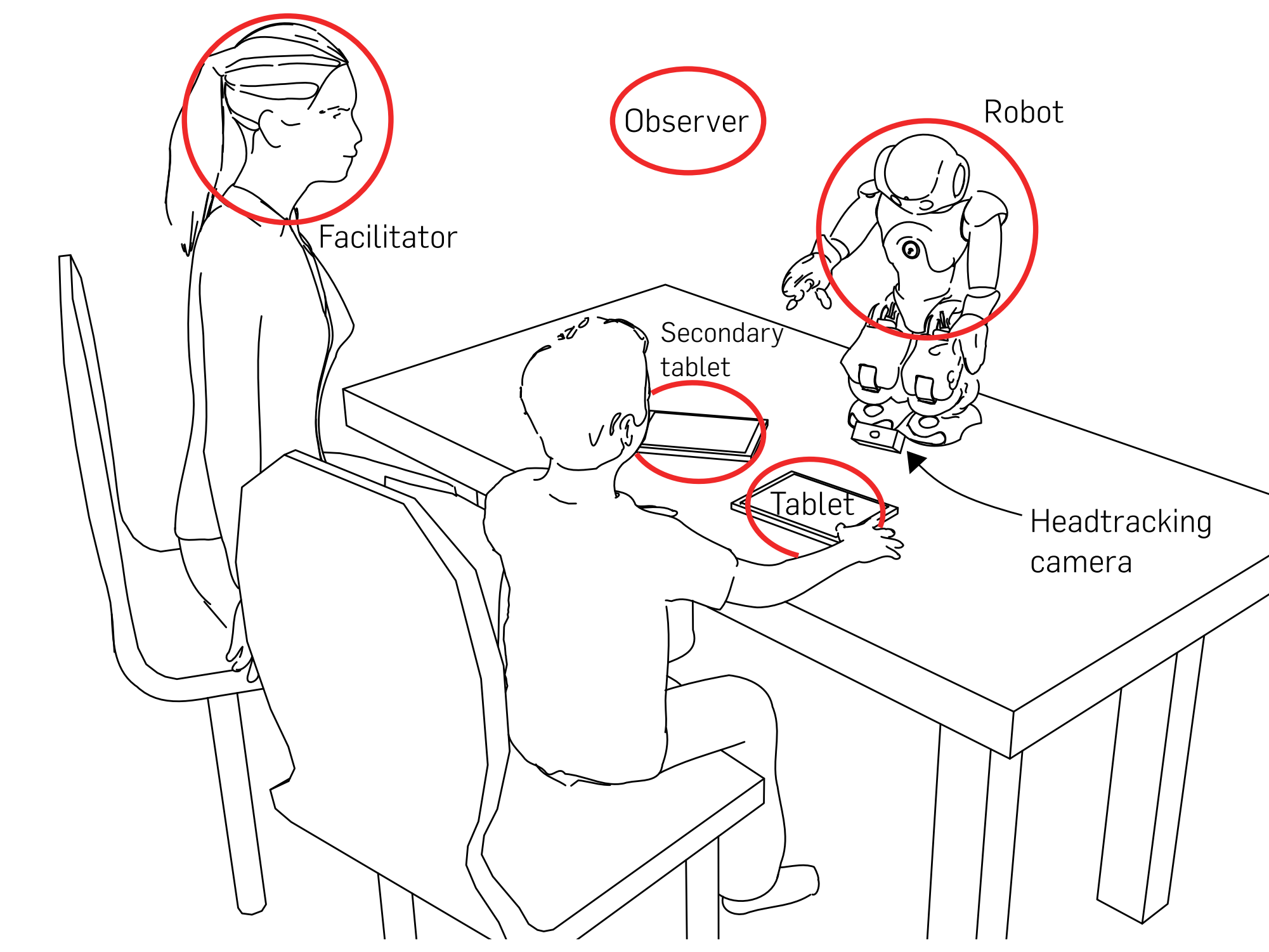

observer
z
y
x
FOV: 20°
d tan(fov/2)
d = x_frame1
(y_frame1, z_frame1)

IN THE FIELD OF VIEW?

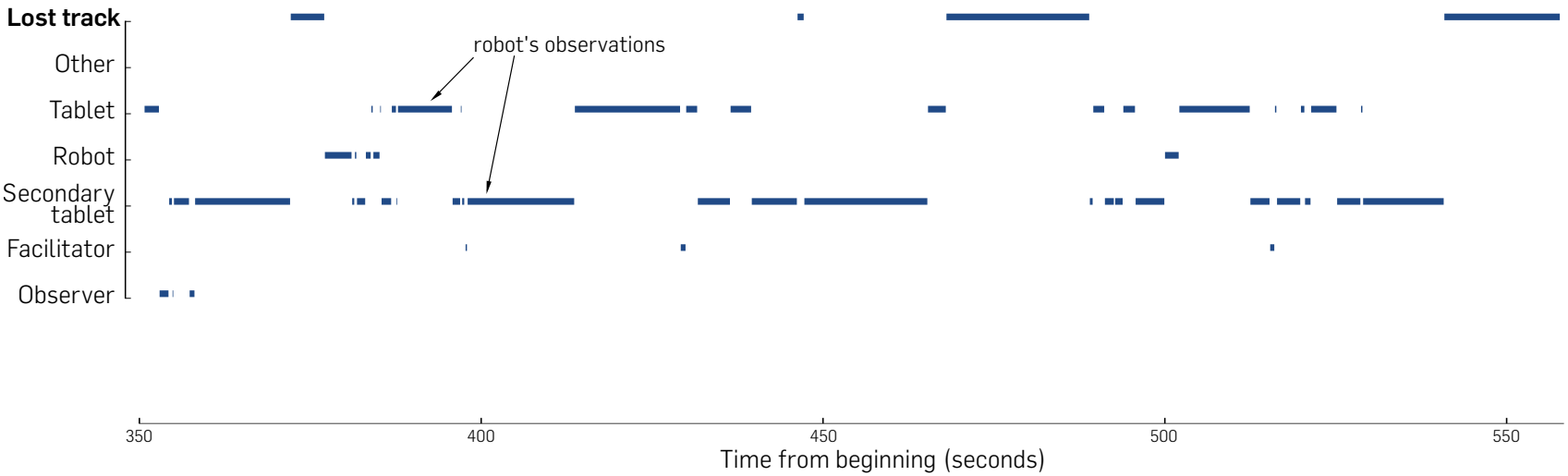
```
static const double FOV = 20. / 180 * M_PI; // radians

bool isInFieldOfView(const tf::TransformListener& listener,
                     const string& target, const string& observer) {

    // compute the location of target from observer's PoV
    tf::StampedTransform transform;
    listener.lookupTransform(observer, target, ros::Time(0), transform);

    // the field of view's main axis is the observer's X axis.
    // So, distance to main axis is simply sqrt(y^2 + z^2)
    double distance_to_main_axis = sqrt(pow(transform.getOrigin().y(), 2) +
                                        pow(transform.getOrigin().z(), 2));

    double fov_radius_at_x = tan(FOV/2) * transform.getOrigin().x();

    if (distance_to_main_axis < fov_radius_at_x) return true;
    else return false;

}
```

Source: gazr


6D pose of the face
face_0
facilitator
other frames
of interest
selection
head_tracking_camera
frame
in focus
head-tracking camera view
Field of Attention

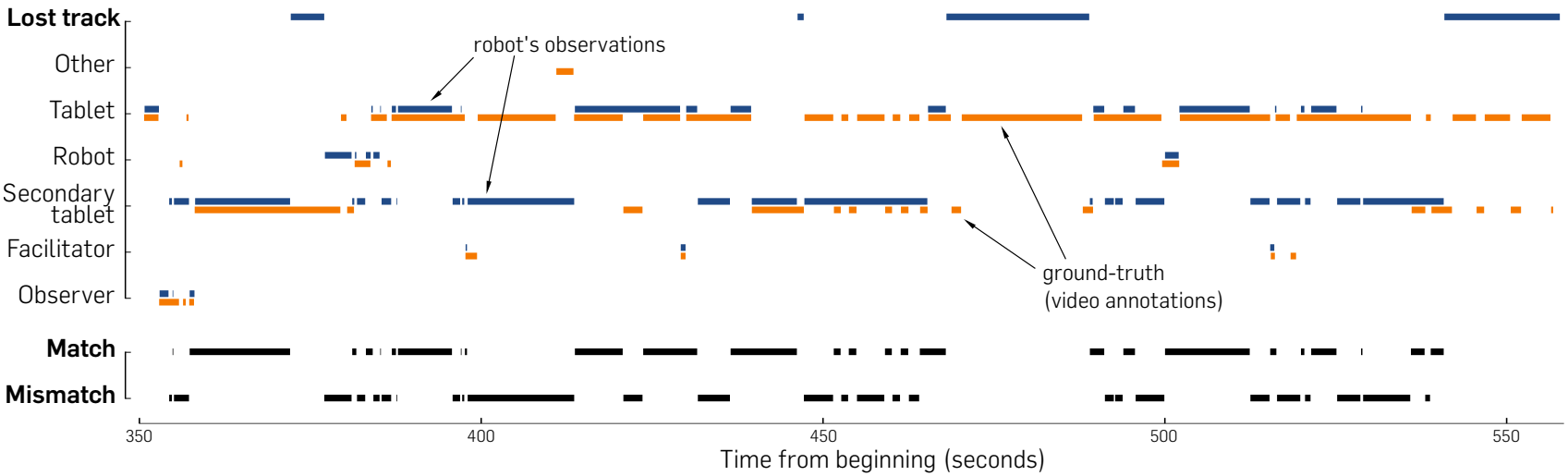

Observer
Robot
Facilitator
Secondary
tablet
Tablet
Headtracking
camera

HOW GOOD IS THE TRACKING?



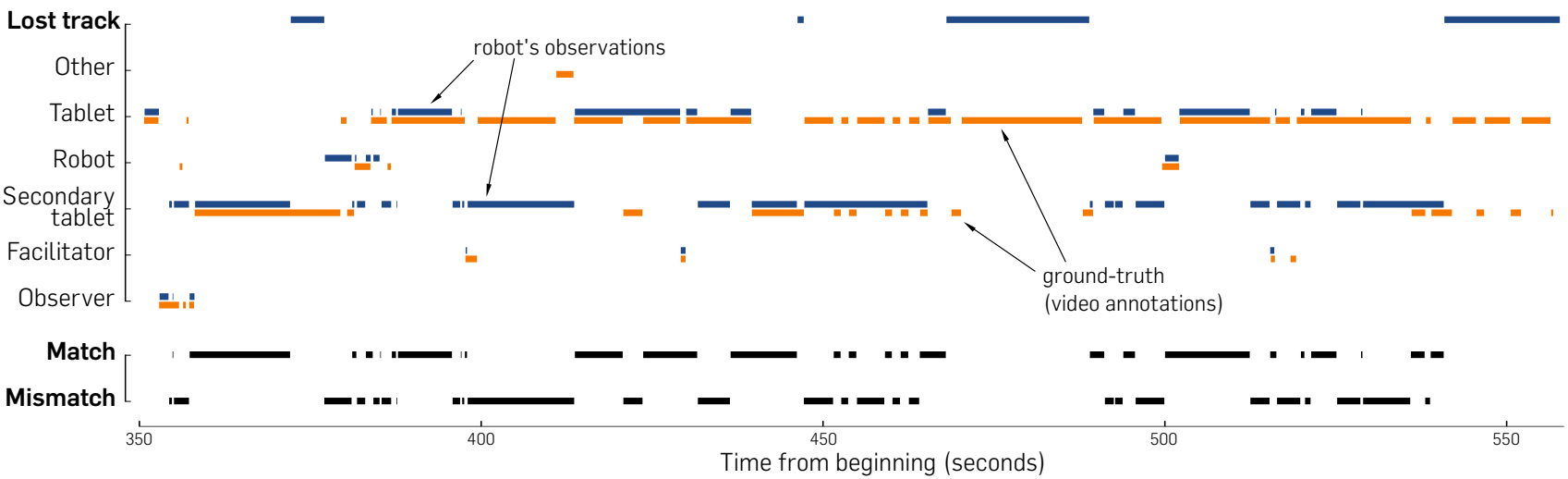
Subject	1	2	3	4	5	6	**M**	*SD*
Head pose tracking (%)	88.2	83.5	90.5	83.1	87.9	85.0	**86.4**	*3.0*

HOW GOOD IS THE TRACKING?



Subject	1	2	3	4	5	6	**M**	*SD*
Head pose tracking (%)	88.2	83.5	90.5	83.1	87.9	85.0	**86.4**	*3.0*

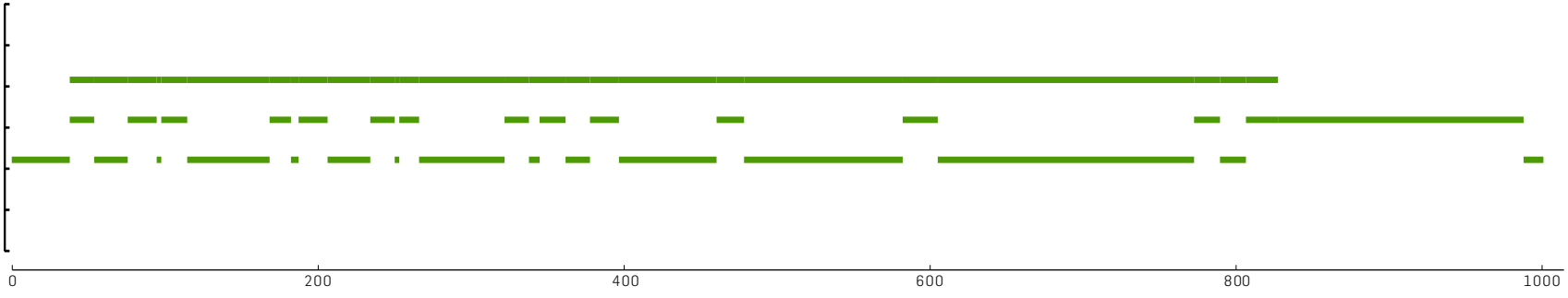
HOW GOOD IS THE TRACKING?



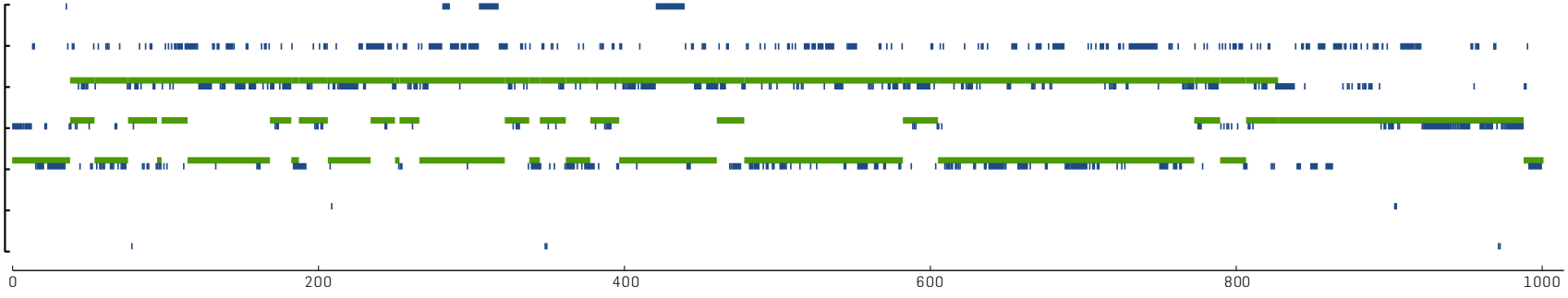
Subject	1	2	3	4	5	6	**M**	*SD*
Head pose tracking (%)	88.2	83.5	90.5	83.1	87.9	85.0	**86.4**	*3.0*
Agreement (%)	58.9	67.1	79.2	48.3	65	77.1	**65.9**	*11.5*
Cohen's κ	0.48	0.56	0.68	0.26	0.47	0.68	**0.52**	*0.16*

CASE STUDY IN ATTENTION TRACKING: WITH-ME-NESS

Robot’s expectations:

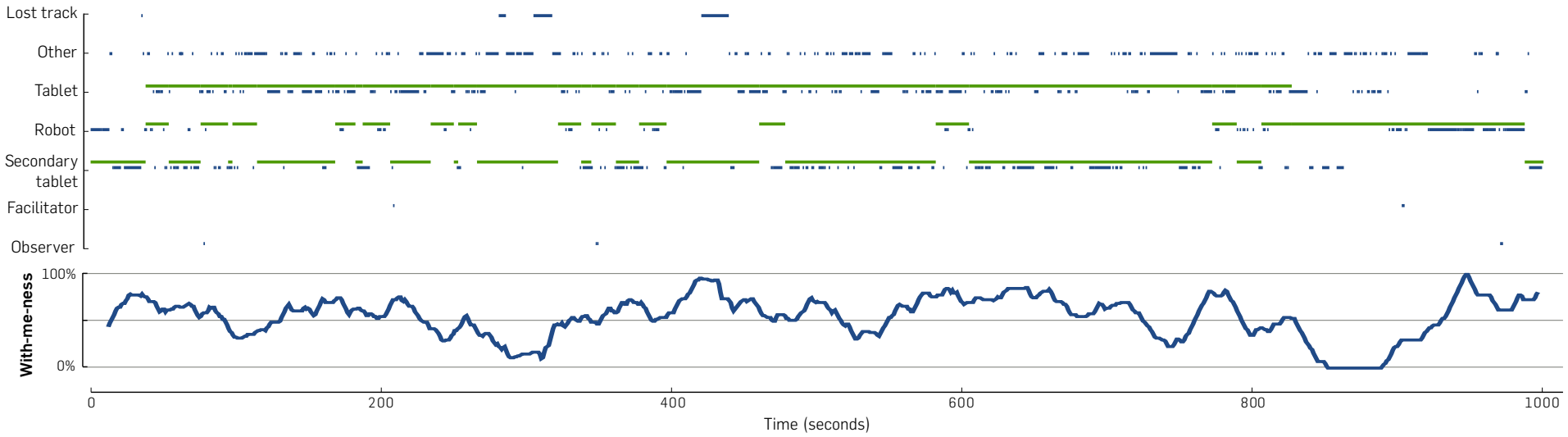


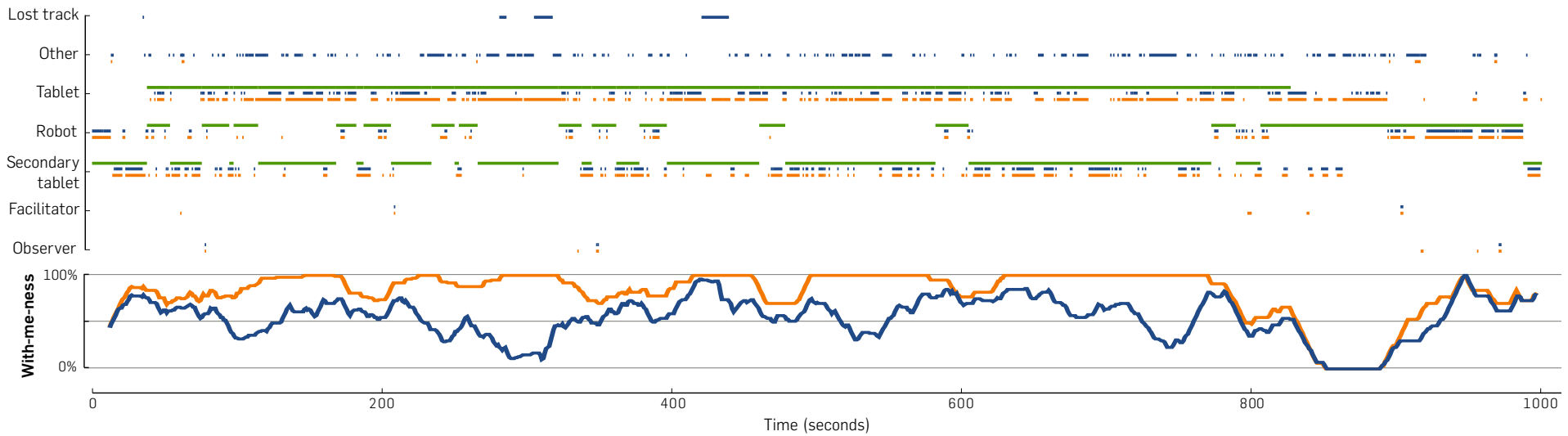
Expectations vs observations:



```
def withmeness(startime, endtime):

    withme_duration = expected_duration = 0
    dt = 0.1 #typically in sec

    for t in range(starttime, endtime, dt):
        task = task_at(t)
        expected_foci = expected_foci(task)
        focus = user_focus_at(t)

        if task and expected_foci and focus:
            if focus in expected_foci:
                withme_duration += dt

            expected_duration += dt

    return withme_duration / expected_duration
```

[*Source: S. Lemaignan; F. Garcia; A. Jacq; P. Dillenbourg; From Real-time Attention Assessment to "With-me-ness" in Human-Robot Interaction*]


Lost track
Other
Tablet
Robot
Secondary tablet
Facilitator
Observer
With-me-ness
100%
0%
0
200
400
600
800
1000
Time (seconds)



$$r(973) = 0.58, p < .001$$

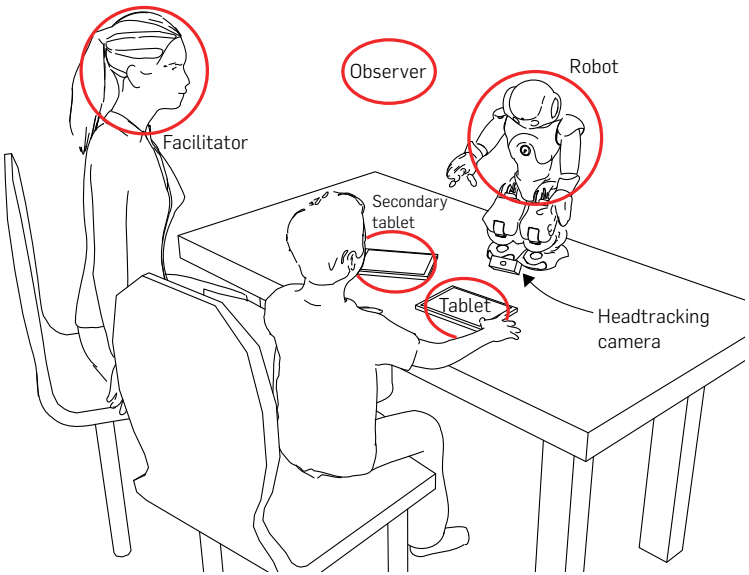
EXPECTED FOCUS

```
def expected_foci(task)?
```

EXPECTED FOCUS

```
def expected_foci(task)?
```

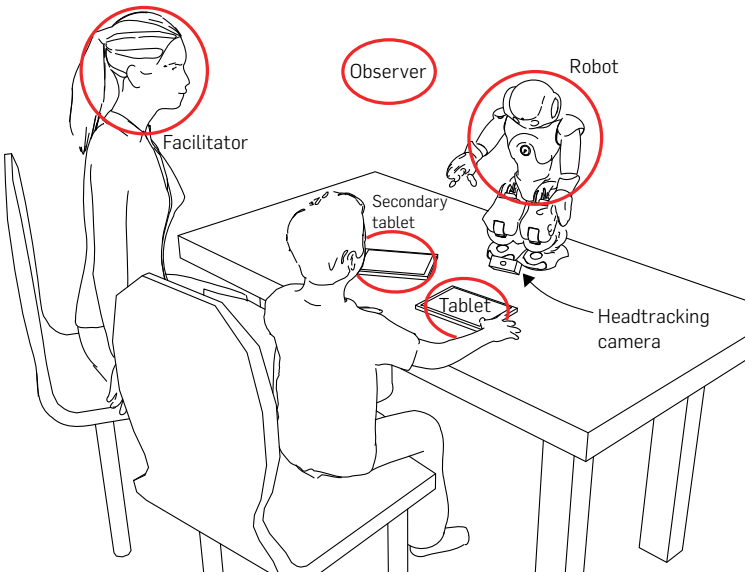
Interaction Phase	Expected targets
Presentation	robot
Waiting for word	secondary tablet
Writing word	tablet robot
Waiting for feedback	tablet secondary tablet
Story telling	robot
Bye	robot



EXPECTED FOCUS

```
def expected_foci(task)?
```

Interaction Phase	Expected targets
Presentation	robot
Waiting for word	secondary tablet
Writing word	tablet robot
Waiting for feedback	tablet secondary tablet
Story telling	robot
Bye	robot



Resulting with-me-ness value **very sensitive to this mapping**
With-me-ness is a **relative metric**!

WITH-ME-NESS IS...

- An **objective** & **quantitative** precursor of engagement...
- ...based on matching the **user's focus of attention** with a set of **prior expectations**
- Can be computed **on-line** by the robot...
- ...and **sensitive to** the (task-dependent) **set of expectations**
- $\Rightarrow$ **relative** metric!

That's all for today, folks!

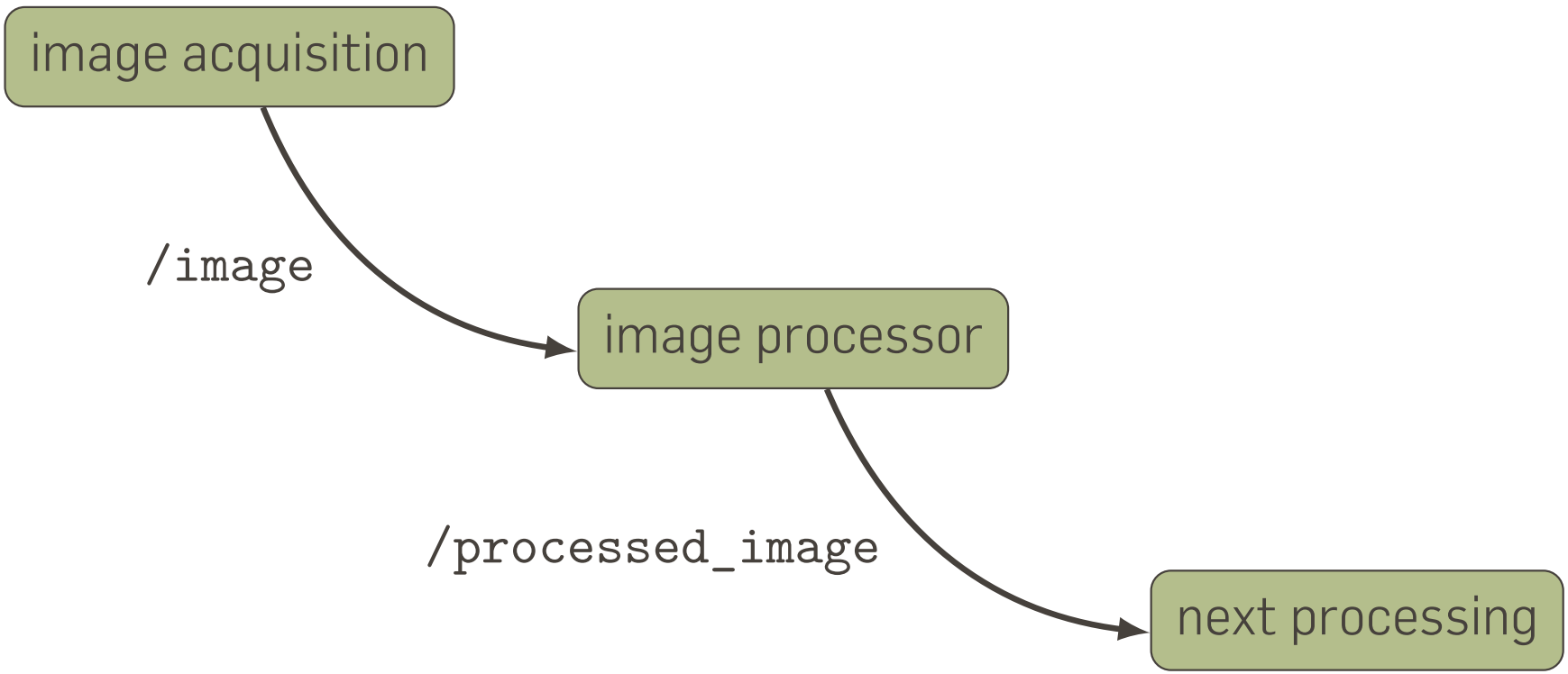
Questions:
Portland Square B316 or **severin.lemaignan@plymouth.ac.uk**

Slides:
github.com/severin-lemaignan/lecture-face-hri

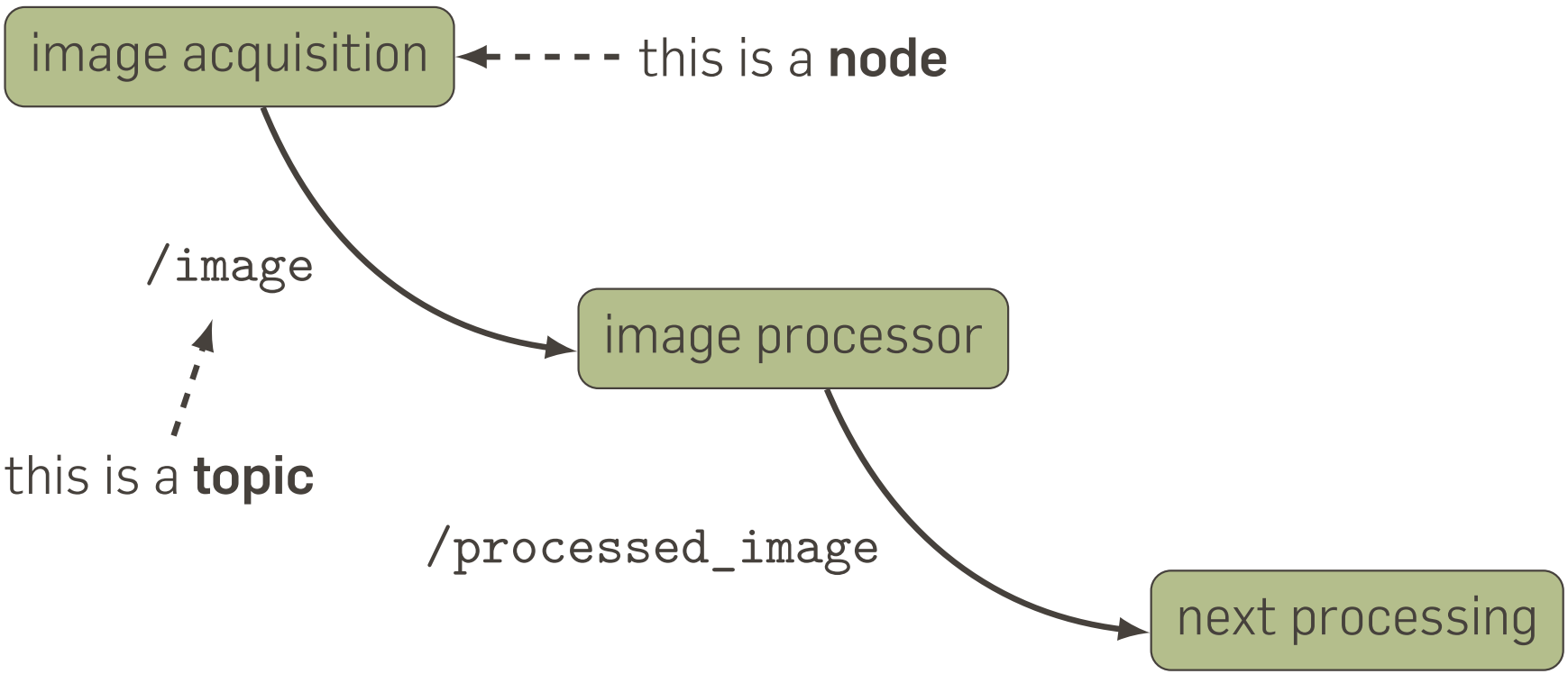
SUPPLEMENTARY MATERIAL

FACE RECOGNITION WITH ROS?

REMINDER: A SIMPLE IMAGE PROCESSING PIPELINE



REMINDER: A SIMPLE IMAGE PROCESSING PIPELINE



```
1  import sys, cv2, rospy
2  from sensor_msgs.msg import Image
3  from cv_bridge import CvBridge
4
5  def on_image(image):
6      cv_image = bridge.imgmsg_to_cv2(image, "bgr8")
7      rows, cols, channels = cv_image.shape
8      cv2.circle(cv_image, (cols/2, rows/2), 50, (0,0,255), -1)
9      image_pub.publish(bridge.cv2_to_imgmsg(cv_image, "bgr8"))
10
11 rospy.init_node('image_processor')
12 bridge = CvBridge()
13 image_sub = rospy.Subscriber("image",Image, on_image)
14 image_pub = rospy.Publisher("processed_image",Image)
15
16 while not rospy.is_shutdown():
17     rospy.spin()
```

HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

```
> rosrun usb_cam usb_cam_node
```

HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

```
> rosrun usb_cam usb_cam_node
```

Then, we run our code:

```
> python image_processor.py image:=/usb_cam/image_raw
```

HOW TO USE THIS CODE?

First, we need to write data onto the `/image` topic, for instance from a webcam:

```
> rosrun usb_cam usb_cam_node
```

Then, we run our code:

```
> python image_processor.py image:=/usb_cam/image_raw
```

Finally, we run a 3rd node to display the image:

```
> rqt_image_view image:=/processed_image
```

CREATING THE FACEREC ROS PACKAGE

LET'S CREATE A PROPER ROS PACKAGE

```
> cd $HOME
> mkdir src && cd src
> catkin_create_pkg facerec rospy
```

LET'S CREATE A PROPER ROS PACKAGE

```
> cd $HOME
> mkdir src && cd src
> catkin_create_pkg facerec rospy
```

```
> ls facerec
CMakeLists.txt  package.xml  src
```

FIRST, ADD SOME CODE

```
> cd facerec
> mkdir -p src/facerec && cd src/facerec
> touch __init__.py # required to create a Python module
> gedit recognition.py
```

FIRST, ADD SOME CODE

```
> cd facerec
> mkdir -p src/facerec && cd src/facerec
> touch __init__.py # required to create a Python module
> gedit recognition.py
```

Just a simple stub for a Python module:

```
def run(dataset):
    print('Dataset: ' + dataset)
```

FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

```
#! /usr/bin/env python

import facerec.recognition

if __name__ == '__main__':
    facerec.recognition.run("my_faces")
```

FIRST, ADD SOME INITIAL CODE

Create as well an executable (our future ROS node) in `scripts/`:

```
> cd ../..
> mkdir -p scripts && cd scripts
> gedit reco
```

```
#! /usr/bin/env python

import facerec.recognition

if __name__ == '__main__':
    facerec.recognition.run("my_faces")
```

```
> chmod +x reco
```

CONFIGURE THE PYTHON 'BUILD'

Because our node is written in Python, our `CMakeLists.txt` is simple:

```
cmake_minimum_required(VERSION 2.8.3)
project(facerec)

find_package(catkin REQUIRED COMPONENTS
  rospy
)

catkin_python_setup()
catkin_package()

install(PROGRAMS
   scripts/reco
   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

CONFIGURE THE PYTHON 'BUILD'

However, we need a `setup.py` (standard Python `distutils`-based packaging):

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# fetch values from package.xml
setup_args = generate_distutils_setup(
                    packages=['facerec'],
                    package_dir={'': 'src'},
                )

setup(**setup_args)
```

INSTALL THE NODE

We can now install our node:

```
> cd ..
> mkdir -p build && cd build
> cmake -DCMAKE_INSTALL_PREFIX=<install prefix> ..
> make install
```

INSTALL THE NODE

We can now install our node:

```
> cd ..
> mkdir -p build && cd build
> cmake -DCMAKE_INSTALL_PREFIX=<install prefix> ..
> make install
```

Assuming ROS is correctly installed, we can run our node:

```
> export ROS_PACKAGE_PATH=<prefix>/share:$ROS_PACKAGE_PATH
> rosrun facerec reco
Dataset: my_faces
```

IMAGE PROCESSING

Let's update the node `reco` and the library (Python *module*) `recognition.py` to perform simple image processing:

`recognition.py`:

```
import cv2

def run(image):
    rows, cols, channels = image.shape
    cv2.circle(image, (cols/2, rows/2), 50, (0,0,255), -1)
```

IMAGE PROCESSING

reco:

```
#! /usr/bin/env python

import sys, rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

import facerec.recognition

def on_image(image):
    cv_image = bridge.imgmsg_to_cv2(image, "bgr8")
    facerec.recognition.run(cv_image)
    image_pub.publish(bridge.cv2_to_imgmsg(cv_image, "bgr8"))

if __name__ == '__main__':
    rospy.init_node('image_processor')
    bridge = CvBridge()
    image_sub = rospy.Subscriber("image",Image, on_image)
    image_pub = rospy.Publisher("processed_image",Image, queue_size=1)

    while not rospy.is_shutdown():
        rospy.spin()
```

TO USE THE NODE

```
> rosrun usb_cam usb_cam_node
```

```
> rosrun facerec reco image:=/usb_cam/image_raw
```

```
> rqt_image_view image:=/processed_image
```

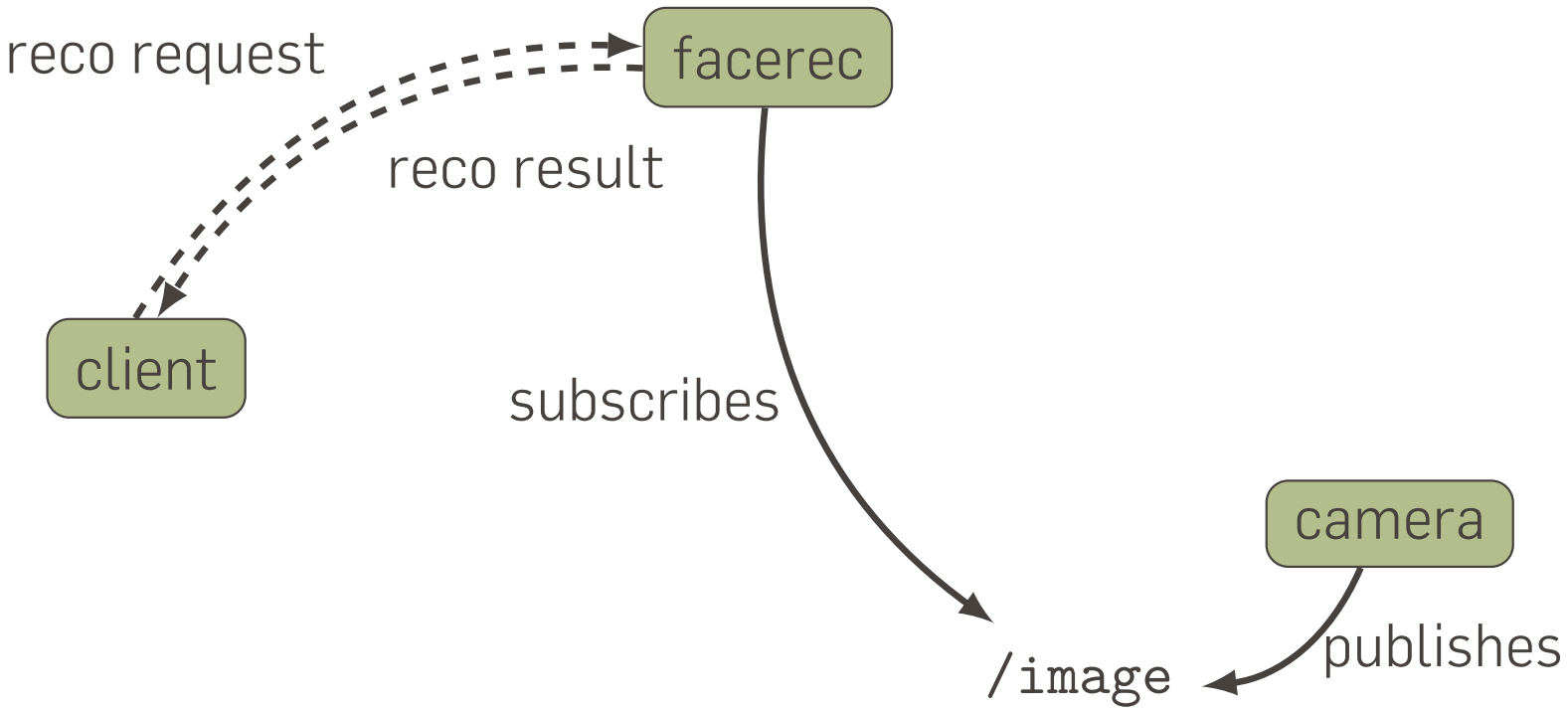
Let’s try it!

WHAT ARE THE NEXT STEPS FOR FACE RECOGNITION?

We need to:

- acquire reference images for each of the face we want to recognise
- re-train the model every time we add new faces to the dataset
- when requested, attempt to recognise the person $\rightarrow$ ROS action

POSSIBLE NETWORK



RECOGNITION LOGIC

Inside `facerec`:

- when incoming request, attempt recognition
- if recognition fails: acquire a couple of images of that person; create a new class; re-train
- if recognition succeeds: add image to corresponding class; re-train
- if unsure: ask for confirmation

RECOGNITION LOGIC

Inside `facerec`:

- when incoming request, attempt recognition
- if recognition fails: acquire a couple of images of that person; create a new class; re-train
- if recognition succeeds: add image to corresponding class; re-train
- if unsure: ask for confirmation

Implementation left as an exercice!