DIGITAL IMAGE PROCESSING

RECOGNITION

REAL-TIME FACE RECOGNITION AND TRACKING PROJECT

VIKTORIIA POPOVA
5111900503
JIANGSU UNIVERSITY

TABLE OF CONTENTS

COURSE PROJECT PURPOSE

The purpose of face recognition project is to design software that can detect human faces from the incoming video.
The great difficulty is ensuring that this process is carried out in real-time, something that is not available to all biometric facial recognition software providers.

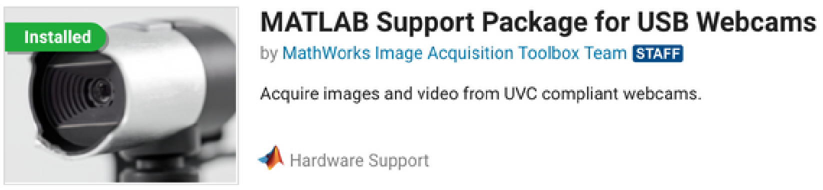
COURSE PROJECT THEORY

What is facial recognition?
Facial recognition is a way to identify or confirm a person's identity through their face. Facial recognition systems can be used to identify people in photos, videos or in real time.
Facial recognition is a biometric safety category. Other forms of biometric software include speech recognition, fingerprint recognition and retina or iris recognition. The technology is mainly used for protection and security forces, although there is growing interest in other areas of use.
How does facial recognition work?
Many people are familiar with FaceID facial recognition technology used to unlock iPhones (however, this is just one of the uses of this technology). In general, facial recognition does not depend on a massive database of photos to determine a person's identity; it simply identifies and recognizes an individual as the sole owner of the device, while limiting access to others.
In addition to unlocking phones, facial recognition works by comparing the faces of people passing in front of special cameras with images of people on a checklist. Checklists can contain photographs of anyone, including those suspected of any unlawful act, and the images can come from anywhere, including our social media accounts. Facial technology systems may vary, but in general they tend to work as follows:
Step 1: Facial recognition
The camera detects and locates the image of a face, either independently or as part of a crowd. The image can show the person in front or profile.
Step 2: Facial analysis
An image of the face is then captured and analyzed. Most facial recognition technology depends on 2D images instead of 3D, since a 2D image can be compared more easily with public photos or those of a database. The software reads the geometry of your face. Key factors include the distance between the eyes, the depth of the eye sockets, the distance from the forehead to the chin, the shape of the cheekbones and the contour of the lips, ears and chin. The goal is to identify the facial landmarks that are key to distinguishing a face.
Step 3: Conversion from image to data
The face capture process transforms analog information (a face) into a set of digital information (data) based on the person's facial features. Basically, the analysis of the face becomes a mathematical formula. The numerical code is called a face fingerprint. Just as fingerprints are unique, each person has their own facial fingerprint.
Step 4: Search for a match
The facial fingerprint is compared to a database of other known faces. For example, the FBI has access to up to 650 million photos from various state databases. On Facebook, any photo tagged with a person's name becomes part of Facebook's database, which can also be used for facial recognition. If the facial fingerprint matches an image in a facial recognition database, then a determination will be made.
Of all biometric measurements, facial recognition is considered the most natural. This follows intuitive logic, since we usually recognize ourselves and others by looking at faces, instead of fingerprints and irises. It is estimated that, periodically, more than half of the world's population is affected by facial recognition technology.

STEPS

After the launching the MATLAB and creating 'New Script'.
We need to access the webcam, to perform realtime face detection and tracking,

To access the webcam using Matlab, we need an 'add-on' named 'MATLAB Support Package for USB Webcams'.



Installed

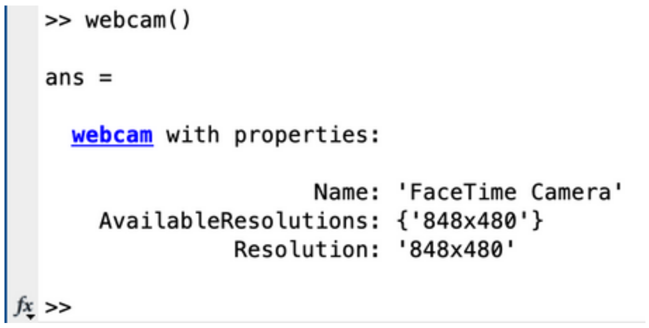
MATLAB Support Package for USB Webcams

by MathWorks Image Acquisition Toolbox Team STAFF

Acquire images and video from UVC compliant webcams.

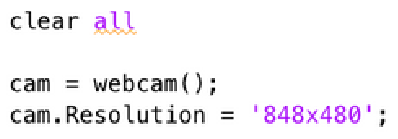
Hardware Support

Once you dowloaded this package, In the command window, type 'webcam ()' and press enter.
If the package is installed, it would show the camera specifications.
So Our system is now ready for 'realtime face detection and tracking'.



```
>> webcam()

ans =

  webcam with properties:

                    Name: 'FaceTime Camera'
    AvailableResolutions: {'848x480'}
              Resolution: '848x480'

fx >>
```

While working with realtime face detection and tracking, variable stored in the workspace, may cause errors. That is why, to remain in safe side, add this 'clear all' at the beginning of the code. It will clear everything from the workspace.
Then create a variable named 'cam' and initiate an 'webcam()' object. After this run the code and write – 'cam.AvailableResolutions' and hit enter. Here are the available resolutions. Depending on your webcam, the available resolution may be different. I was using this one, because I didn't have any opinions.
Now, write cam.Resolution = 848x480.



```
clear all

cam = webcam();
cam.Resolution = '848x480';
```

STEPS

After that, take a variable named ‘video_Frame’ and we are going to use ‘snapshot()’ function to read the frames one by one from the ‘cam’ object.
Now it is time to initiate the video player object. For that, create a variable named ‘video_Player’ and we going to assign the ‘video player’ object to it. The first argument of this object is the ‘Position’. In the second argument, in a set of square brackets, we define the position. The first two values are for left and bottom corner. The second two values are width and height of the of the video.
Create another variable named ‘face-Detector’ and assign the ‘cascade object detection()’ object to it. We use this object to detect the face.

```
video_Frame = snapshot(cam);

video_Player = vision.VideoPlayer('Position', [100 100 848 480]);

face_Detector = vision.CascadeObjectDetector();
point_Tracker = vision.PointTracker('MaxBidirectionalError', 2);
```

To track the face, we need a point tracker object. Create a variable named ‘point_Tracker’ and put the ‘point tracker object’ in it.
Now, I initiated three variables for while loop. The first one is ‘run_loop’ equals true. The second one is ‘number of points’ equals 0 and the third one is ‘frame_Count’ equals 0.

```
run_loop = true;
number_of_Points = 0;
frame_Count = 0;
```

Now, declare a while loop. It will keep looping as long as run_loop is true and frame_Count is less than 800 frames. If you want to run the webcam for longer, you can increase the number of frames. Then I ended this loop. Inside this loop, i did the detection and tracking. First, inside the ‘video_Frame’ variable, we are going to store the frames from the ‘cam’ object using ‘snapshot()’ function. Then using the ‘rgb2gray()’ function, we are converting the frames into grayscale and storing them in a variable named ‘gray_Frame’.
Then write, ‘frame_Counter = frame_Counter+1’ to increase the ‘frame_Counter’ by 1 ...

```
while run_loop && frame_Count <800

video_Frame = snapshot(cam);
gray_Frame = rgb2gray(video_Frame);
frame_Count = frame_Count+1;
```

STEPS

Initiate an if condition. It is true when the 'number of points' is less than 10.
At the beginning of this if condition, we are locating the rectangle that encloses the face on 'gray_Frame' using step function and face 'face_Detector' object.
Now, if the 'face_Rectangle' is not empty, then using 'detect Min Eigen Features' function, we are going to find the points of the rectangle. The first argument of this function is the frame where the image is located. In our case, it is the 'gray_Frame'. Then we need to tell the function whether we are interested to get feature of entire image, or a particular region. We are interested for a particular region, that is why we are specifying region of interest or 'ROI' here. The ROI is the location where the 'face_Rectangle' is located. We are using this 'one comma colon' to get the value of the first row of the 'face rectangle' matrix. Which is actually the starting location of this rectangle.

```
if number_of_Points < 10
face_Rectangle = face_Detector.step(gray_Frame);

if ~isempty(face_Rectangle)
points = detectMinEigenFeatures(gray_Frame, 'ROI', face_Rectangle(1, :));
```

Next, create a variable named 'xy_Points' and write 'points.Location' to convert the points to 'x y values'. Now the x values of 'xy_Points' variable has the number of points we need to initialize the point tracker.
Tracker create a variable named "number of points" and use the size function to get the X values only. the first argument of this function is "XY_points" and the second argument is 1. the second argument specifies which value we will get here 1 means the X values. Then I released the point tracker to empty it. then initialize it using "initialize" function. the first argument of this function is the object we want to initialize, here it is "point_tracker", the second argument is the XY value where we want to initiate the tracker and the third argument specifies on which frame we are going to initialize it in our case it is gray frame.

```
xy_Points = points.Location;
number_of_Points = size(xy_Points, 1);
release(point_Tracker);
initialize(point_Tracker, xy_Points, gray_Frame);
```

Once the tracker is initialized, assign the actual points to a variable named "previous points" so that later we can compare the distance between new XY points and these previous points.

```
previous_Points = xy_Points;
```

STEPS

Initiate an if condition. It is true when the ‘number of points’ is less than 10.
At the beginning of this if condition, we are locating the rectangle that encloses the face on ‘gray_Frame’ using step function and face ‘face_Detector’ object.
Now, if the ‘face_Rectangle’ is not empty, then using ‘detect Min Eigen Features’ function, we are going to find the points of the rectangle. The first argument of this function is the frame where the image is located. In our case, it is the ‘gray_Frame’. Then we need to tell the function whether we are interested to get feature of entire image, or a particular region. We are interested for a particular region, that is why we are specifying region of interest or ‘ROI’ here. The ROI is the location where the ‘face_Rectangle’ is located. We are using this ‘one comma colon’ to get the value of the first row of the ‘face rectangle’ matrix. Which is actually the starting location of this rectangle.

```
if number_of_Points < 10
face_Rectangle = face_Detector.step(gray_Frame);

if ~isempty(face_Rectangle)
points = detectMinEigenFeatures(gray_Frame, 'ROI', face_Rectangle(1, :));
```

Next, create a variable named ‘xy_Points’ and write ‘points.Location’ to convert the points to ‘x y values’. Now the x values of ‘xy_Points’ variable has the number of points we need to initialize the point tracker.
Tracker create a variable named “number of points” and use the size function to get the X values only. the first argument of this function is “XY_points” and the second argument is 1. the second argument specifies which value we will get here 1 means the X values. Then I released the point tracker to empty it. then initialize it using “initialize” function. the first argument of this function is the object we want to initialize, here it is “point_tracker”, the second argument is the XY value where we want to initiate the tracker and the third argument specifies on which frame we are going to initialize it in our case it is gray frame.

```
xy_Points = points.Location;
number_of_Points = size(xy_Points, 1);
release(point_Tracker);
initialize(point_Tracker, xy_Points, gray_Frame);
```

Once the tracker is initialized, assign the actual points to a variable named “previous points” so that later we can compare the distance between new XY points and these previous points.

```
previous_Points = xy_Points;
```

STEPS

Create a variable named “rectangle”. now we are converting the “face_rectangle” which is around the detected face into points using Bbox two-points function. now create a variable named “face polygon”. When the face will move at different angles, the face rectangle enclosing the face will be transformed into polygons to adjust with the geometric orientation of the face.

```
rectangle = bbox2points(face_Rectangle(1, :));
face_Polygon = reshape(rectangle', 1, []);
```

It is done using “reshape” function. the first input argument is the transpose of the rectangle variable where the points of the rectangle are stored. then, with this one here we are creating a single row matrix. this empty set of square brackets has been used to automatically adjust the dimension. in the video frame variable, we are going to assign the output from “insertShape” function. the first input argument of this function is the “video_frame” where we are going to insert the shape, second argument is the name of the shape which is “polygon”, the third argument is the location of where it will be drawn, the fourth argument is the “line width” and the fifth argument is the value of the language.
In this “video frame” variable we are going to assign the output from “insertMarker” function. this function is going to insert marker on video frame where there are “X Y points” the symbol of the marker is “+” and color is “white”. and this is the end of the first “if” condition. this if condition is true when the number of points is less than 10. when the number of points is more than 10, we consider that there has been significant change in our face orientation and we need to transform the shape of the rectangle geometrically.

```
video_Frame = insertShape(video_Frame, 'Polygon', face_Polygon, 'LineWidth', 3);
video_Frame = insertMarker(video_Frame, xy_Points, '+', 'Color', 'white');
end
```

At the beginning of the else condition we are going to find the number of “XY points” and their presence. here “is found” is 1 if there are points and 0 if there is no point. now create a variable named “new point”. then store the values of the first row of X Y points by using this “isfound” comma colon (:). create another variable named “old points”. this time stored the values of the first row of previous points in it. now the number of points is the number of points stored in no point variable.

```
else
[xy_Points, isFound] = step(point_Tracker, gray_Frame);
new_Points = xy_Points(isFound, :);
old_Points = previous_Points(isFound, :);

number_of_Points = size(new_Points, 1);
```

STEPS

Create an if condition. it is true when the number of points is more than ten. at the beginning of the if condition we are going to create a row matrix. the elements are x for all points and new points variable. then we are going to estimate the geometric transformation between all points and new points using estimate geometry transform function. the input argument of this function is the previous old point, new point, similarity, max distance and the threshold. here the threshold is 4 which means minimum for pair of points will be candidate for estimation based on their similarity and maximum distance. it has become a long line! use3 dots to break the line.

```
if number_of_Points >=10
[xform, old_Points, new_Points] = estimateGeometricTransform(...
old_Points, new_Points, 'similarity', 'MaxDistance', 4);
```

now take the rectangle variable and perform the transformation using transform points function. the first argument of this function is X for variable where the estimated transformation information is stored, the second argument is the rectangle where we are going to perform the transformation. and the transformed rectangle is stored in the rectangle variable. we cannot insert this rectangle directly on the frame we need to resize it to generate x1 y1 x2 y2 x3 y3 and x4 y4 pairs of points. create a variable named “face polygon” then use the “reshape” function the argument of this reshape function is the transpose of rectangle matrix. this one is generating a single row vector and this empty set of square brackets has been used to define the dimension automatically. now we can insert this “face polygon” in video frame. for that take the video frame variable again,

```
rectangle = transformPointsForward(xform, rectangle);
face_Polygon = reshape(rectangle', 1, []);
```

then take an “insert shape” function. the first argument of this function is the video frame where we want to insert shape. then the name of the shape, it is polygon here, the third argument is the variable when the polygon is stalled. it is the face polygon variable. then I'm setting the “line width” to 3. now we are going to insert the marker in the video frame variable again in the same way we did in the previous if condition. now set the new points to previous points variable, and set the point tracker to this updated previous points.

this is the end of this if condition, and this is the end of the previous if condition.

```
video_Frame = insertShape(video_Frame, 'Polygon', face_Polygon, 'LineWidth', 3);
video_Frame = insertMarker(video_Frame, new_Points, '+', 'Color', 'white');
previous_Points = new_Points;
setPoints(point_Tracker, previous_Points);
end
end
```

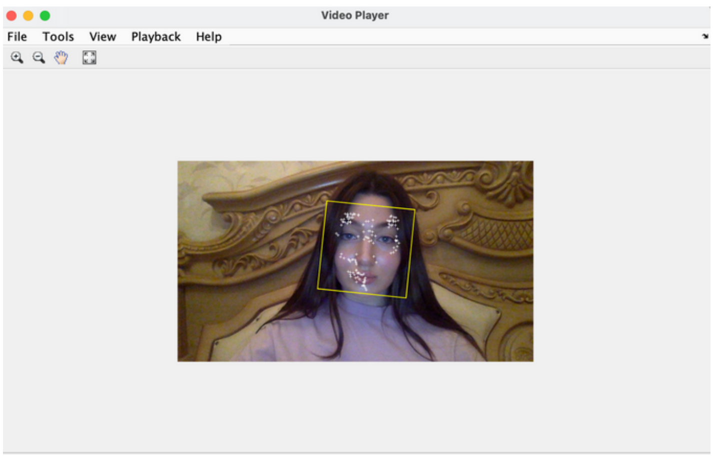
STEPS

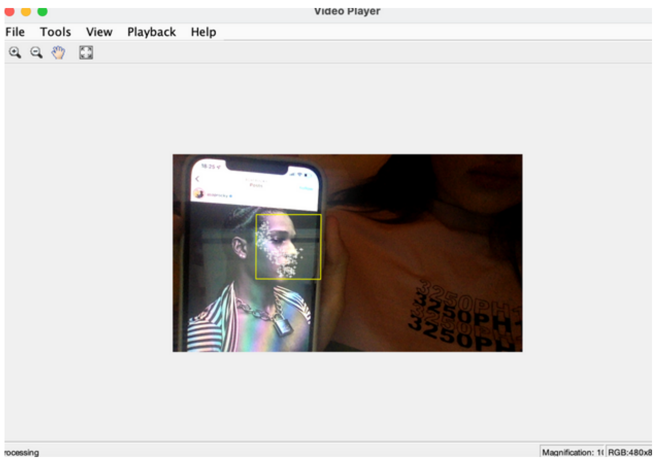
Tt is time to set the video frame to video player to see our program in action. we do it using "step"function. the first argument is the video player, and the second argument is the video frame. now take the round loop variable again and check if the video player is open using " is open" function. if we close the video player then run loop will be false and while loop will be terminated this is the end of the while loop. now we need to clear the camp object. we also need to release the "video player" "point tracker" and "face detector" object.

```
step (video_Player, video_Frame);
run_loop = isOpen(video_Player);
end

clear cam;

release(video_Player);
release(point_Tracker);
release(face_Detector);
```

Let's run the code.

RESULTS





CONCLUSION

The system is working perfectly.
This is how we can make a real-time face detection and tracking system.
I hope you have understood the procedure.

The goal to design software that can detect human faces from the incoming real-time video is achieved.

SOURCE CODE

```
clear all

cam = webcam();
cam.Resolution = '848x480';
video_Frame = snapshot(cam);

video_Player = vision.VideoPlayer('Position', [100 100 848 480]);

face_Detector = vision.CascadeObjectDetector();
point_Tracker = vision.PointTracker('MaxBidirectionalError', 2);

run_loop = true;
number_of_Points = 0;
frame_Count = 0;

while run_loop && frame_Count <800

video_Frame = snapshot(cam);
gray_Frame = rgb2gray(video_Frame);
frame_Count = frame_Count+1;

if number_of_Points < 10
face_Rectangle = face_Detector.step(gray_Frame);

if ~isempty(face_Rectangle)
points = detectMinEigenFeatures(gray_Frame, 'ROI', face_Rectangle(1, :));

xy_Points = points.Location;
number_of_Points = size(xy_Points, 1);
release(point_Tracker);
initialize(point_Tracker, xy_Points, gray_Frame);

previous_Points = xy_Points;

rectangle = bbox2points(face_Rectangle(1, :));
face_Polygon = reshape(rectangle', 1, []);

video_Frame = insertShape(video_Frame, 'Polygon', face_Polygon,
'LineWidth', 3);
video_Frame = insertMarker(video_Frame, xy_Points, '+', 'Color', 'white');
end
else
[xy_Points, isFound] = step(point_Tracker, gray_Frame);
new_Points = xy_Points(isFound, :);
old_Points = previous_Points(isFound, :);

number_of_Points = size(new_Points, 1);

if number_of_Points >=10
[xform, old_Points, new_Points] = estimateGeometricTransform(...
old_Points, new_Points, 'similarity', 'MaxDistance', 4);

rectangle = transformPointsForward(xform, rectangle);
face_Polygon = reshape(rectangle', 1, []);

video_Frame = insertShape(video_Frame, 'Polygon', face_Polygon,
'LineWidth', 3);
video_Frame = insertMarker(video_Frame, new_Points, '+', 'Color',
'white');
previous_Points = new_Points;
setPoints(point_Tracker, previous_Points);
end
end
step (video_Player, video_Frame);
run_loop = isOpen(video_Player);
end

clear cam;

release(video_Player);
release(point_Tracker);
release(face_Detector);
```