

# 2018 Fall Embedded System Lab Final Project Report

## Face Dance Machine on Raspberry Pi3

---

Author: Yu-Shan Huang (黃郁珊) Szu-Yu Mo (莫絲羽)  
Student ID: b04901091 b04901164  
Email: [qazwsx860809@gmail.com](mailto:qazwsx860809@gmail.com) [evamo.tw1@gmail.com](mailto:evamo.tw1@gmail.com)  
Department: Electrical Engineering Department, National Taiwan University  
Advisor: Professor Sheng-De Wang  
Github repo: <https://github.com/NTUEE-ESLab/2018Fall-FaceDanceMachine>  
Demo video: <https://youtu.be/WfL82hLJuYI>  
Project date: January 2019

---

## Content

- Abstract
- Motivation
- Implementation
- Challenge
- Achievement
- Reference

## **Abstract**

In this project, we have successfully developed a “face dancing machine” game on Raspberry Pi3. By “face dancing”, we simply make facial expressions identical to the cartoon emojis on the screen, just like what we do to our legs when playing traditional “dance dance revolution”. We integrated several OpenCV and Dlib functions with our own algorithms to meet the goal of recognizing facial expression in real-time. Moreover, the GUI interface is implemented via the Pygame package on Python3. The main contribution is that we have completed complex computations on an embedded system rather than on a pc, where the former only runs on ARM Cortex-A53 with 1024MB RAM.

## Motivation

On one of the course class, our teach assistant had introduced us OpenCV face detection as well as Dlib facial landmark detector. With a picamera in hand, we start to think about what we can do with the video taken by it, and how we can integrate OpenCV and Dlib in the system.

Lacking facial expressions is also one characteristic of us. Thus, we feel like developing a game that our facial muscles could exercise. Motivated by the desire to develop a game on a portable device, the face dance machine is about to born.

## Implementation

### I. Environment setup

All of our programs run on Python3 in Raspbian Jessie operating system embedded in Raspberry Pi3 B+. In addition, several crucial packages are required for running the scripts. Those packages include OpenCV 3.4.3, Pygame 1.9.3, Imutils 0.5.2, Dlib 19.16.99 and several dependencies. The installation of them, as well as the usage of this project, would be mentioned in the following instructions.

#### A. OpenCV 3.4.3

The most important part to install OpenCV is that we have to turn on NEON and VFPV3 hardware optimizations when compiling via source. ARM NEON is an optimization architecture extension for ARM processors. It was designed by ARM engineers specifically for faster video processing, image processing, speech recognition, and machine learning. This optimization supports Single Instruction Multiple Data (SIMD), which describes an architecture where multiple processing elements in the pipeline perform operations on multiple data points (hardware) all executed with a single instruction. The ARM engineers also built VFPV3, a floating point optimization, into the chip our Raspberry Pi 3's use. Specifically, since we are running Python3, we have to compile the source for Python3. The following shows the steps of installation:

1. Expand filesystem and reclaim space

```
$ sudo raspi-config
choose "expand filesystem"
$ sudo reboot
$ sudo apt-get purge wolfram-engine
$ sudo apt-get purge libreoffice*
$ sudo apt-get clean
$ sudo apt-get autoremove
```
2. Install dependencies

```
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install build-essential cmake pkg-config
```

```

$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev
libpng12-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk2.0-dev libgtk-3-dev
$ sudo apt-get install libcanberra-gtk*
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python2.7-dev python3-dev
3. Download the OpenCV source code
$ cd ~
$ wget -O opencv.zip
https://github.com/opencv/opencv/archive/3.3.0.zip
$ unzip opencv.zip
$ wget -O opencv_contrib.zip
https://github.com/opencv/opencv\_contrib/archive/3.3.0.zip
$ unzip opencv_contrib.zip
4. Install Numpy
$ sudo pip3 install numpy
5. Compile and install the optimized OpenCV library for Raspberry Pi
$ cd ~/opencv-3.3.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
-D ENABLE_NEON=ON \
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_EXAMPLES=OFF \
-D PYTHON_DEFAULT_EXECUTABLE=$(which python3) ..
$ sudo vim /etc/dphys-swapfile
change CONF_SWAPSIZE from 100 to 1024
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
make sure we are in the opencv-3.3.0/build/ directory
$ sudo make -j4
$ sudo make install
$ sudo ldconfig
go back to reset the CONF_SWAPSIZE value and restart the swap
service

```

#### B. Pygame 1.9.3

```
$ sudo apt-get install python3-pygame
```

#### C. Imutils 0.5.2

```
$ sudo pip3 install imutils
```

#### D. Dlib 19.16.99

We can actually install Dlib in a relatively quick way via pip3 install. However, if we would like to leverage the use of ARM NEON hardware optimization, we would have to install it via source. Since the compilation process requires lots of memory, we have to reclaim as much memory as possible, just like what we did for installing OpenCV.

##### 1. Update swap file size, boot options, and memory split

```
$ sudo vim /etc/dphys-swapfile
```

change CONF\_SWAPSIZE from 100 to 1024

```
$ sudo /etc/init.d/dphys-swapfile stop
```

```
$ sudo /etc/init.d/dphys-swapfile start
```

```
$ sudo raspi-config
```

select Boot options => Desktop / CLI => Console Autologin

go back to the main screen

select Advanced Options => Memory Split

Update the GPU memory value to be 16MB and then exit

```
$ sudo reboot
```

##### 2. Install dlib prerequisites

```
$ sudo apt-get update
```

```
$ sudo apt-get install build-essential cmake
```

```
$ sudo apt-get install libgtk-3-dev
```

```
$ sudo apt-get install libboost-all-dev
```

##### 3. Download the OpenCV source code

```
$ git clone https://github.com/davisking/dlib.git
```

```
$ cd dlib
```

```
$ python3 setup.py install --yes USE_NEON_INSTRUCTIONS
```

##### 4. Reset your swap file size, boot options, and memory split

go back to reset the CONF\_SWAPSIZE value

restart the swap service

reset your GPU/RAM split to 64MB

update the boot options to boot into the desktop interface

```
$ sudo reboot
```

#### E. Usage

After setting up all the dependencies, we can clone the whole repository to anywhere on our computer. A model for detecting faces has to be downloaded into the same directory as well. Finally, we can run and play with the program by executing the 'main.py' script. Instructions are shown as the followings:

\$ git clone <https://github.com/NTUEE-ESLab/2018Fall-FaceDanceMachine.git>  
 download the model here: <https://goo.gl/Z2JCch> and put it in the above directory  
 To execute the game, run  
 \$ python3 main.py

## II. System structure

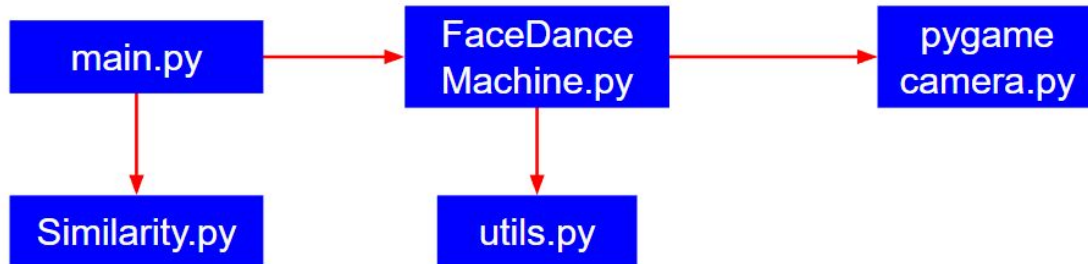


Fig. 1 System structure

As figure 1 shows, five python scripts together form our system. To utilize the advantage of object-oriented language, we wrapped “FaceDanceMachine”, “pygamecamera”, and “Similarity” into classes. In “main.py”, we initialized pygame and constructed a window for display. We also initialized a “Similarity” instance and a “FaceDanceMachine” instance. While the former represents the facial expression similarity calculator, the latter stands for the main game loop. As for the “FaceDanceMachine.py”, we initialized the camera object for capturing images from picamera and utilized several useful functions provided in “utils.py”, being mainly functions for loading images.

Now we will briefly introduce each member and function of a class.

### A. FaceDanceMachine

#### 1. member variable

- a) black: the tuple representing the color black
- b) display: the pygame surface for display
- c) camera: the Camera object for capturing images
- d) similarity: the Similarity object for recognizing expressions
- e) imgs: numpy array consists of game images
- f) buttons: numpy array consists of game buttons
- g) level: the level that the user has chosen
- h) result: the facial expression recognition result
- i) sim: the similarity of the facial expression recognition

#### 2. member function

- a) welcome(): show the welcome page and the start button
- b) menu(): show the menu page with three level buttons
- c) countdown(): play the countdown animation
- d) game(): the main while loop
- e) job(frame, mission): the facial expression recognition thread target function

- f) pause(): the pause page with resume and exit buttons
- g) exit(): show exit page for one sec and shut down the program
- h) end(score): show the game score with replay and exit buttons
- i) run(): the function that wraps the system together

## B. Camera

- 1. member variable
  - a) clist: list of available camera
  - b) camera: the pygame camera instance
  - c) screen: the pygame surface for display
- 2. member function
  - a) capture(): return an image surface captured by the camera
  - b) pg2cv(frame): transform pygame surface to OpenCV numpy array.
  - c) stop(): the function to stop the pygame camera, releasing the memory so that another process can access the picamera

## C. Similarity

- 1. member variable
  - a) detector: the dlib face detector
  - b) predictor: the dlib facial landmark predictor
- 2. member function
  - a) Frown(landmark): detect frowning or not
  - b) RightEyeWink(landmark): detect the right eye winking or not
  - c) MouthOpen(landmark): detect whether the mouth is open
  - d) MouthClosed(landmark): detect whether the mouth is closed
  - e) MouthLeft(landmark): detect whether the mouth is at the left side
  - f) MouthRight(landmark): detect whether the mouth is at the right side
  - g) Smile(landmark): detect whether the person is smiling or not
  - h) PointDown(landmark): detect whether the center of mouth is higher than both sides of it
  - i) MouthOval(landmark): detect whether the mouth is an oval
  - j) MouthCircle(landmark): detect whether the mouth is a circle
  - k) MouthWide(landmark): detect whether the mouth is wide enough
  - l) FaceLeft(landmark): detect whether the face is at the left side
  - m) FaceRight(landmark): detect whether the face is at the right side
  - n) face\_dance(target, face\_mission): judging whether the target image satisfies the standard for each face mission, and return which mission the target completes and the similarity

Finishing the introduction of our classes, we should briefly talk about the main game loop in our system and how the GUI looks. The workflow can be observed through figure 2.

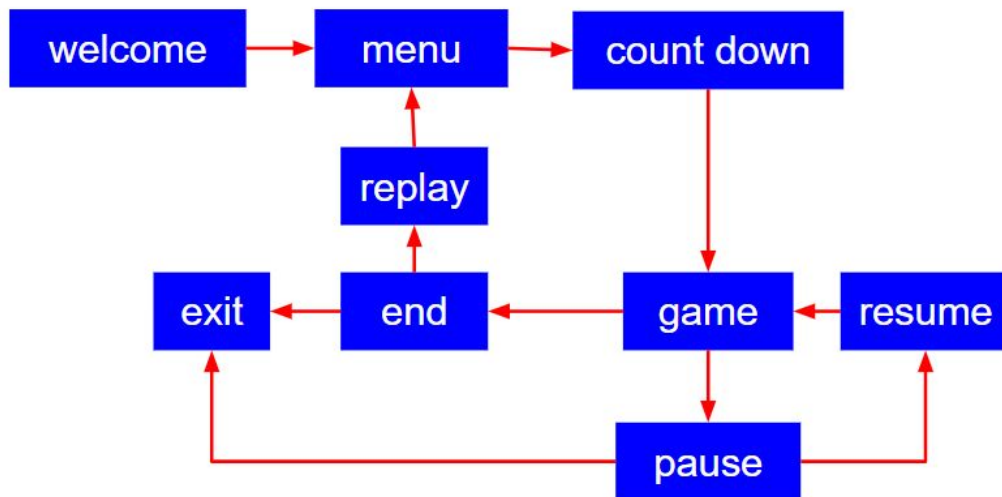


Fig. 2 Workflow of the game loop

The welcome page (fig. 3), the menu page (fig. 4), an example of the screen in the game (fig. 5), and the exit page (fig. 6) are shown below. All images are original.



Fig 3 Welcome page



Fig. 4 Menu page



Fig 5 game page



Fig. 6 exit page



### III. Facial Express Comparison

We select ten facial expressions as galleries. The photos captured by camera are first transformed into grayscale images, then we detect faces in images, transform them into landmarks, and eventually compare those landmarks (target) with galleries and compute their similarity.

#### A. Facial Landmark

##### 1. Face Detection

We first transform original images into grayscale. Then, we employ a pretrained detector provided by Dlib to detect human faces in grayscale images. The first detected face is chosen as an input image and passed to next process.

##### 2. Landmark Detection

The first detected face images are then marked with 68 facial landmarks after passed into another pretrained detector "shape\_predictor\_68\_face\_landmarks." The target landmarks are shown in fig. 7. The numbers shown in fig. 7 are used as fixed index. For example, we can always locate mouth positions by accessing landmarks 49~68.



Fig. 7

left eyebrow	landmark[18:22]
right eyebrow	landmark[13:27]
left eye	landmark[37:42]
right eye	landmark[43:48]
nose	landmark[28:36]
mouth	landmark[49:68]
face contour	landmark[1:17]

Table 1

The following computation methods are all based on position of those landmarks.

#### B. Facial Expressions Comparison

As shown in section II part C, there are several member function related to local facial expression. Those are conditions we used to distinguish if a target belongs to each galleries and compute their similarity. Their computation algorithms will be illustrated in this part. For further explanation, our computing algorithm only use ratio or relative size since the target size won't be the same. [Landmark are replaced by L]

## 1. local facial expression

### a. Frown

We first compute the ratio of distance between eyebrows( $L[43,X]-L[40,X]$ ) over distance between eyes(canthuses)( $L[23,X]-L[22,X]$ ). If all the conditions that (1)the ratio $<0.7$ , (2)the vertical distance between left/right canthus and left/right inner eyebrows is smallest among left/right eyes and left/right eyebrows landmarks, (3)the one between left/right middle eye and left/right middle eyebrow is largest, and (4) the left/right inner eyebrows( $L[22,Y]/L[23,Y]$ ) is the lowest point among left/right eyebrows, we determine the target is frowning. [similarity= $(1-\text{eyebrow\_dis}/\text{eye\_dis}) * 1.5$ ]

### b. RightEyeWink

We first calculate the distance between upper eye contour and lower eye contour. Two upper point correspond to two lower point in each eye. After summing up distances in each eye. The right one should be the lower. The second condition is the vertical distance between eyebrows, we assume right eyebrows is lower when people wink their right eye. We determine the target is winking his/her right eye when either the first or the second condition is fulfilled. [similarity=1]

### c. MouthOpen

If the vertical distance between upper mouth and lower mouth ( $L[67,Y]-L[63,Y]$ ) is bigger than thickness of mouth( $L[63,Y]-L[52,Y]$ ), we determine the target is opening his/her mouth. [similarity=1]

### d. MouthClosed

If the vertical difference between mean of upper mouth points ( $L[66:68,Y]$ ) and mean of lower mouth point ( $L[62:64,Y]$ ) is smaller than thickness of mouth( $L[63,Y]-L[52,Y]$ ), we determine the target is closing his/her mouth. [similarity=1]

### e. MouthLeft

We set middle point of nose( $L[28,X]$ ) as point of reference. After calculating the difference between mouth landmarks and the point of reference. If the amount of point whose difference is positive(viewed as right point) is less than 8/10, we determine the target's mouth is in left side. [similarity= $1-\text{positive\_num}/20 * 0.5$ ]

### f. MouthRight

We set middle point of nose( $L[28,X]$ ) as point of reference. After calculating the difference between mouth

landmarks and the point of reference. If the amount of point whose difference is negative(viewed as left point) is less than 8/10, we determine the target's mouth is in right side.

[similarity=1-negative\_num/20\*0.5]

g. Smile

We aim to detect if the mouth corners are rising. If mean of upper mouth point( $L[61:68,Y]+L[49,Y]+L[55,Y]$ ) is lower than left/right mouth corners( $L[49,Y]/L[55,Y]$ ), we determine the target is smiling.

[similarity=0.8+0.2\*(height- $L[49,Y]+L[55,Y]$ )/2)/height]

h. PointDown

We aim to detect if the mouth corners are pointing down. If mean of upper mouth point( $L[65:68,Y]+L[60,Y]$ ) is higher than left/right mouth corners( $L[49,Y]/L[55,Y]$ ), we determine the corner of target's mouth are pointing down.

[similarity=0.8+0.2\*(height- $L[49,Y]+L[55,Y]$ )/2)/height]

i. MouthOval

We first calculate the width( $L[55,X]-L[49,X]$ ) and the height( $L[58,Y]-L[52,Y]$ ) of mouth. If the ratio of height over width is larger than 1, we determine the contour of target's mouth is oval. [similarity=1]

j. MouthCircle

We first calculate the width( $L[55,X]-L[49,X]$ ) and the height( $L[58,Y]-L[52,Y]$ ) of mouth. If (height/width-1) is smaller than 0.5, we determine the contour of target's mouth is circle. [similarity=1-abs(height/width-1)\*0.5]

k. MouthWide

We first calculate the width( $L[55,X]-L[49,X]$ ) and the height( $L[58,Y]-L[52,Y]$ ) of mouth. If height is larger width, we determine the target's mouth is widely open.

[similarity=0.8+0.2\*width/height\*0.3]

l. FaceLeft








We set middle point of nose( $L[28,X]$ ) as point of reference. We calculate the horizontal distance between left/right face( $L[1,X]/L[17,X]$ ) and point of reference. If left\_dis/right\_dis>1.5, we determine the target is turning to left. [similarity=1]




m. FaceRight

We set middle point of nose( $L[28,X]$ ) as point of reference. We calculate the horizontal distance between left/right face( $L[1,X]/L[17,X]$ ) and point of reference. If

right\_dis/left\_dis>1.5, we determine the target is turning to right. [similarity=1]

## 2. galleries

facial expression		
	conditions	Frown MouthOpen, MouthWide
	similarity	frown_score*0.6+mouth_wide_score*0.4
	conditions	MouthClosed, Not Smile, Not PointDown Not Frown
	similarity	1
	conditions	MouthOpen, Smile
	similarity	smile_score*0.7+mouth_open*0.3
	conditions	Not Frown MouthLeft, MouthCircle FaceLeft
	similarity	mouth_left_score*0.5+mouth_circle_score*0.5
	conditions	Not Frown MouthRight, MouthCircle FaceRight
	similarity	mouth_right_score*0.5+mouth_circle_score*0.5
	conditions	MouthLeft, MouthClosed, Not smile
	similarity	mouth_left_score*0.5+mouth_closed_score*0.5
	conditions	PointDown
	similarity	mouth_pointDown_score

	conditions	MouthOpen, Not MouthLeft, MouthCircle or MouthOval Not Frown
	similarity	$\text{mouth\_open\_score} * 0.3 +$ $(\max(\text{mouth\_oval\_score}, \text{mouth\_circle\_score})) * 0.7$
	conditions	RightEyeWink MouthLeft, MouthOpen
	similarity	$\text{mouth\_left\_score} * 0.8 + \text{Reye\_score} * 0.2$
	conditions	RightEyeWink Smile
	similarity	$\text{smile\_score} * 0.8 + \text{Reye\_score} * 0.2$

## Challenge

### I. Conflict between PyQt5 & OpenCV video

Initially, we plan to use PyQt5 for the GUI of our game. However, it was later found out that when PyQt5 is integrated with OpenCV, the video process of OpenCV and the GUI process of PyQt5 would result in a segmentation fault when exiting the program. The accumulation of segmentation faults would then cause the CPU to be busy, and thus we cannot run any other programs for a period of time.

Since the above problem seems unable to be solved, we decided to look for other GUI packages. That is the time we decided to use Pygame rather than PyQt5. Pygame not only provides GUI interface, but also provides the picamera interface that can perfectly match the display system. The came-along pygame.camera module made displaying video stream rather easy.

### II. Resource limitation of picamera

To predict the facial expression, we have to get the numpy array representing each image. Nevertheless, the only hardware picamera can only be used by one process, so it is impossible to get a stream via pygame and another stream via OpenCV. The only solution is that we will have to conduct a data conversion between Pygame surface and OpenCV numpy array.

Here the challenge is the time consumed for the data conversion. At first, we were using the “array3d” function of pygame to transfer the pygame surface into a numpy array. However, it was time-consuming due to the fact that the function copies the array. We then later found out that another function “pixels3d” can bring out the

same result but with referencing the data. This has made the conversion process faster.

### **III. Slow face detection of Dlib**

This is the most time-consuming part of the whole process. By testing each line of our program, we found that to detect the face in the image takes approximately 0.6 seconds, which is unacceptable for playing a game. Each frame is then delayed by at least 3 seconds before showing on the screen. Since it is the problem of dlib, we tried to search for dlib optimization. We found out that installing Dlib via source with NEON support on can speed up Dlib processes. As a result, we \$pipe uninstall dlib, and then cloned the dlib libraries down and compiled it. That was a tough task though, since compiling Dlib takes lots of RAMs. After the installation, the detection speeds up to 0.5 seconds per frame. Although it was an improvement, it was way not enough for our real-time game.

Noting that we have already met the hardware limits, we started to think if we can do something in software to “deceive” the players, making them believe that the game is running smoothly without finding out the slow detection of face. We then thought of the thread programming that the professor has mentioned in class. We created a thread for the calculation of facial expressions, including detecting the face, predicting the facial landmarks, and computing whether the landmarks meet the criteria of the face missions shown on screen. For every 50 frames, we wait for the thread to finish its work, resulting in the fact that the frame being calculated is actually the one 0.5 seconds ago. Through testing, we believed that the game is now smooth enough.

### **IV. Picamera Suddenly not working**

There was once that the picamera just stopped working. Searching for solutions on the Internet did not help at all. Recalling what we had done, we thought about the Dlib installation. It was mentioned that to install Dlib via source, we had to rearranged the memory for the GPU on Raspberry Pi from 128MB to 16MB. Nonetheless, we changed the memory back to 64MB rather than 128MB after the installation, thinking that maybe it will provide more RAM for CPU. The consequence was that 64MB was not enough for the picamera, so it stopped working. Changing the memory back to 128MB solved everything.

## Achievement

We came up with new algorithm to detect facial expressions based on advanced face detection toolkit, which runs much more faster than directly training models on Raspberry Pi3 but equally accurate. Besides we improve the delay when displaying caused by calculation and make the game more entertaining.

To sum up, we solved lots of problems, including environment setting, toolkit compatibility, camera accessing, image type transformation, runtime improvement, and computing methods. We also spend time designing our own surface. All patterns in the game are drawn by ourselves. Eventually, we have developed an interesting game.

## Reference

1. Rosebrock, A. (2017, October 9). Optimizing OpenCV on the Raspberry Pi. Retrieved January 6, 2019, from <https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/>
2. Rosebrock, A. (2017, May 1). Install dlib on the Raspberry Pi. Retrieved January 17, 2019, from <https://www.pyimagesearch.com/2017/05/01/install-dlib-raspberry-pi/>
3. Rosebrock, A. (2018, January 22). Install dlib (the easy, complete guide). Retrieved January 18, 2019, from <https://www.pyimagesearch.com/2018/01/22/install-dlib-easy-complete-guide/>
4. Wang, G. T. (2018, May 17). Python 多執行緒 threading 模組平行化程式設計教學. Retrieved January 18, 2019, from <https://blog.gtwang.org/programming/python-threading-multithreaded-programming-tutorial/>
5. Tseng, C. H. (2018, August 18). Face Landmark & Alignment. Retrieved January 15, 2019, from <https://chtseng.wordpress.com/2018/08/18/face-landmark-alignment/>