ДОДАТОК А
Лістинг коду першого модулю програми

Файл iris_recognition.py

```
import collections
from array import *
import numpy as np
import cv2
import os
import sys
import math
import random
import cPickle as pickle
import copy
import gzip
import inspect
from Tkinter import *
import tkFileDialog
from PIL import ImageTk, Image
import points

import itertools

from matplotlib import pyplot as plt


def compareImages(filepath1, filepath2):
    print "Analysing " + filepath1
    rois_1 = load_rois_from_image(filepath1)
    print "Analysing " + filepath2
    rois_2 = load_rois_from_image(filepath2)
    getall_matches(rois_1, rois_2, 0.8, 10, 0.15, show=True)

def compare_binfiles(bin_path1, bin_path2):
    print "Analysing " + bin_path1
    rois_1 = load_rois_from_bin(bin_path1)
    print "Analysing " + bin_path2
    rois_2 = load_rois_from_bin(bin_path2)
    getall_matches(rois_1, rois_2, 0.88, 10, 0.07, show=True)

def load_rois_from_image(filepath):
    img = load_image(filepath, show=True)
    print "Getting iris boundaries.."
    pupil_circle, ext_iris_circle = get_iris_boundaries(img, show=True)
    if not pupil_circle or not ext_iris_circle:
        print "Error finding iris boundaries!"
        return

    print "Equalizing histogram .."
    roi = get_equalized_iris(img, ext_iris_circle, pupil_circle, show=True)
```

```
    print "Getting roi iris images ..."
    rois = get_rois(roi, pupil_circle, ext_iris_circle, show=True)
    print "Searching for keypoints ... \n"

    sift = cv2.xfeatures2d.SIFT_create()
    load_keypoints(sift, rois, show=True)
    load_descriptors(sift, rois)
    return rois

def load_image(filepath, show=False):
    img = cv2.imread(filepath, 0)
    if show:
        cv2.imshow(filepath, img)
        ch = cv2.waitKey(0)
        cv2.destroyAllWindows()
    return img

def get_iris_boundaries(img, show=False):
    # Finding iris inner boundary
    pupil_circle = find_pupil(img)

    if not pupil_circle:
        print 'ERROR: Pupil circle not found!'
        return None, None

    # Finding iris outer boundary
    radius_range = int(math.ceil(pupil_circle[2] * 1.5))
    multiplier = 0.25
    center_range = int(math.ceil(pupil_circle[2] * multiplier))
    ext_iris_circle = find_ext_iris(
        img, pupil_circle, center_range, radius_range)

    while (not ext_iris_circle and multiplier <= 0.7):
        multiplier += 0.05
        print 'Searching exterior iris circle with multiplier ' + \
            str(multiplier)
        center_range = int(math.ceil(pupil_circle[2] * multiplier))
        ext_iris_circle = find_ext_iris(img, pupil_circle,
                                        center_range, radius_range)
    if not ext_iris_circle:
        print 'ERROR: Exterior iris circle not found!'
        return None, None

    if show:
        cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        draw_circles(cimg, pupil_circle, ext_iris_circle,
                     center_range, radius_range)
        cv2.imshow('iris boundaries', cimg)
        ch = cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
    return pupil_circle, ext_iris_circle

def find_pupil(img):
    def get_edges(image):
        edges = cv2.Canny(image, 20, 100)
        kernel = np.ones((3, 3), np.uint8)
        edges = cv2.dilate(edges, kernel, iterations=2)
        ksize = 2 * random.randrange(5, 11) + 1
        edges = cv2.GaussianBlur(edges, (ksize, ksize), 0)
        return edges

    param1 = 200  # 200
    param2 = 120  # 150
    my_array = array('i', [])
    index = 0
    pupil_circles = []
    while (param2 > 35 and len(pupil_circles) < 100):
        for mdn, thrs in [(m, t)
                          for m in [3, 5, 7]
                          for t in [20, 25, 30, 35, 40, 45, 50, 55, 60]]:
            # Median Blur
            median = cv2.medianBlur(img, 2 * mdn + 1)

            # Threshold
            ret, thres = cv2.threshold(
                median, thrs, 255,
                cv2.THRESH_BINARY_INV)

            # Fill Contours
            con_img, contours, hierarchy = \
                cv2.findContours(thres.copy(),
                                 cv2.RETR_EXTERNAL,
                                 cv2.CHAIN_APPROX_NONE)
            draw_con = cv2.drawContours(thres, contours, -1, (255), -1)

            # Canny Edges
            edges = get_edges(thres)

            # HoughCircles
            if len(cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 1, param1, param2)) != 4:
                circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 1, param1, param2)
                circles = np.uint16(np.around(circles))
            else:
                circles = None

            if circles is not None:
                # convert the (x, y) coordinates and radius of the circles
                # to integers
                circles = np.round(circles[0, :]).astype("int")
                for c in circles:
                    pupil_circles.append(c)
```

```
            param2 = param2 - 1
        cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        return get_mean_circle(pupil_circles)

    def get_mean_circle(circles, draw=None):
        if not circles:
            return
        mean_0 = int(np.mean([c[0] for c in circles]))
        mean_1 = int(np.mean([c[1] for c in circles]))
        mean_2 = int(np.mean([c[2] for c in circles]))

        if draw is not None:
            draw = draw.copy()
            # draw the outer circle
            cv2.circle(draw, (mean_0, mean_1), mean_2, (0, 255, 0), 1)
            # draw the center of the circle
            cv2.circle(draw, (mean_0, mean_1), 2, (0, 255, 0), 2)
            cv2.imshow('mean circle', draw)
            ch = cv2.waitKey(0)
            cv2.destroyAllWindows()

        return mean_0, mean_1, mean_2

    def find_ext_iris(img, pupil_circle, center_range, radius_range):
        def get_edges(image, thrs2):
            thrs1 = 0  # 0
            edges = cv2.Canny(image, thrs1, thrs2, apertureSize=5)
            kernel = np.ones((3, 3), np.uint8)
            edges = cv2.dilate(edges, kernel, iterations=1)
            ksize = 2 * random.randrange(5, 11) + 1
            edges = cv2.GaussianBlur(edges, (ksize, ksize), 0)
            return edges

        def get_circles(hough_param, median_params, edge_params):
            crt_circles = []
            for mdn, thrs2 in [(m, t)
                               for m in median_params
                               for t in edge_params]:
                # Median Blur
                median = cv2.medianBlur(img, 2 * mdn + 1)

                # Canny Edges
                edges = get_edges(median, thrs2)

                # HoughCircles
                circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 2, 32,
                                           200, hough_param)
                if circles is not None:
                    # convert the (x, y) coordinates and radius of the
                    # circles to integers
                    circles = np.round(circles[0, :]).astype("int")
                    for (c_col, c_row, r) in circles:
```

```
                if point_in_circle(
                        int(pupil_circle[0]), int(pupil_circle[1]),
                        center_range, c_col, c_row) and \
                        r > radius_range:
                    crt_circles.append((c_col, c_row, r))
        return crt_circles

    param2 = 120  # 150
    total_circles = []
    while param2 > 40 and len(total_circles) < 50:
        crt_circles = get_circles(
            param2, [8, 10, 12, 14, 16, 18, 20], [430, 480, 530])
        if crt_circles:
            print param2
            total_circles += crt_circles
        param2 = param2 - 1

    if not total_circles:
        print "Running plan B on finding ext iris circle"
        param2 = 120
        while (param2 > 40 and len(total_circles) < 50):
            crt_circles = get_circles(
                param2, [3, 5, 7, 21, 23, 25], [430, 480, 530])
            if crt_circles:
                total_circles += crt_circles
            param2 = param2 - 1

    if not total_circles:
        return

    cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    filtered = filtered_circles(total_circles)
    return get_mean_circle(filtered)

def point_in_circle(c_col, c_row, c_radius, p_col, p_row):
    return distance(c_col, c_row, p_col, p_row) <= c_radius

def filtered_circles(circles, draw=None):
    # what if there are only 2 circles - which is alpha?
    def get_alpha_radius(circles0):
        alpha_circle = None
        dist_min = None
        circles1 = circles0[:]
        circles2 = circles0[:]
        for crt_c in circles1:
            dist = 0
            for c in circles2:
                dist += math.fabs(float(crt_c[2]) - float(c[2]))
            if not dist_min or dist < dist_min:
                dist_min = dist
                alpha_circle = crt_c
        return alpha_circle[2]
```

```
if not circles:
    print 'Error: empty circles list in filtered_circles() !'
    return []
c_0_mean, c_0_dev = standard_dev([int(i[0]) for i in circles])
c_1_mean, c_1_dev = standard_dev([int(i[1]) for i in circles])
filtered = []
filtered_pos = []
not_filtered = []
ratio = 1.5
for c in circles[:]:
    if c[0] < c_0_mean - ratio * c_0_dev or \
            c[0] > c_0_mean + ratio * c_0_dev or \
            c[1] < c_1_mean - ratio * c_1_dev or \
            c[1] > c_1_mean + ratio * c_1_dev:
        not_filtered.append(c)
    else:
        filtered_pos.append(c)
if len([float(c[2]) for c in filtered_pos]) < 3:
    filtered = filtered_pos
else:
    alpha_radius = get_alpha_radius(filtered_pos)
    mean_radius, dev_radius = standard_dev(
        [float(c[2]) for c in filtered_pos])
    max_radius = alpha_radius + dev_radius
    min_radius = alpha_radius - dev_radius
    for c in filtered_pos:
        if c[2] < min_radius or \
                c[2] > max_radius:
            not_filtered.append(c)
        else:
            filtered.append(c)

if draw is not None:
    draw = draw.copy()
    for circle in not_filtered:
        # draw the outer circle
        cv2.circle(draw, (circle[0], circle[1]), circle[2], (255, 0, 0), 1)
        # draw the center of the circle
        cv2.circle(draw, (circle[0], circle[1]), 2, (255, 0, 0), 2)
    for circle in filtered:
        # draw the outer circle
        cv2.circle(draw, (circle[0], circle[1]), circle[2], (0, 255, 0), 1)
        # draw the center of the circle
        cv2.circle(draw, (circle[0], circle[1]), 2, (0, 255, 0), 2)
    cv2.imshow('filtered_circles() total={0} filtered_pos={1} filtered={2}'. \
               format(len(circles), len(filtered_pos), len(filtered)),
               draw)
    ch = cv2.waitKey(0)
    cv2.destroyAllWindows()
return filtered
```

```
def draw_circles(cimg, pupil_circle, ext_iris_circle,
                 center_range=None, radius_range=None):
    # draw the outer pupil circle
    cv2.circle(cimg, (pupil_circle[0], pupil_circle[1]), pupil_circle[2],
               (0, 0, 255), 1)
    # draw the center of the pupil circle
    cv2.circle(cimg, (pupil_circle[0], pupil_circle[1]), 1, (0, 0, 255), 1)
    if center_range:
        # draw ext iris center range limit
        cv2.circle(cimg, (pupil_circle[0], pupil_circle[1]), center_range,
                   (0, 255, 255), 1)
    if radius_range:
        # draw ext iris radius range limit
        cv2.circle(cimg, (pupil_circle[0], pupil_circle[1]), radius_range,
                   (0, 255, 255), 1)
    # draw the outer ext iris circle
    cv2.circle(cimg, (ext_iris_circle[0], ext_iris_circle[1]),
               ext_iris_circle[2], (0, 255, 0), 1)
    # draw the center of the ext iris circle
    cv2.circle(cimg, (ext_iris_circle[0], ext_iris_circle[1]),
               1, (0, 255, 0), 1)

def get_equalized_iris(img, ext_iris_circle, pupil_circle, show=False):
    def find_roi():
        mask = img.copy()
        mask[:] = (0)

        cv2.circle(mask,
                   (ext_iris_circle[0], ext_iris_circle[1]),
                   ext_iris_circle[2], (255), -1)
        cv2.circle(mask,
                   (pupil_circle[0], pupil_circle[1]),
                   pupil_circle[2], (0), -1)

        roi = cv2.bitwise_and(img, mask)

        return roi

    roi = find_roi()

    # Mask the top side of the iris
    for p_col in range(roi.shape[1]):
        for p_row in range(roi.shape[0]):
            theta = angle_v(ext_iris_circle[0], ext_iris_circle[1],
                            p_col, p_row)
            if theta > 50 and theta < 130:
                roi[p_row, p_col] = 0

    ret, roi = cv2.threshold(roi, 50, 255, cv2.THRESH_TOZERO)

    equ_roi = roi.copy()
```

```
    cv2.equalizeHist(roi, equ_roi)
    roi = cv2.addWeighted(roi, 0.0, equ_roi, 1.0, 0)

    if show:
        cv2.imshow('equalized histogram iris region', roi)
        ch = cv2.waitKey(0)
        cv2.destroyAllWindows()

    return roi

def get_rois(img, pupil_circle, ext_circle, show=False):
    bg = img.copy()
    bg[:] = 0

    init_dict = {'img': bg.copy(),
                 'pupil_circle': pupil_circle,
                 'ext_circle': ext_circle,
                 'kp': None,
                 'img_kp_init': bg.copy(),
                 'img_kp_filtered': bg.copy(),
                 'des': None
                 }

    rois = {'right-side': copy.deepcopy(init_dict),
            'left-side': copy.deepcopy(init_dict),
            'bottom': copy.deepcopy(init_dict),
            'complete': copy.deepcopy(init_dict)
            }

    for p_col in range(img.shape[1]):
        for p_row in range(img.shape[0]):
            if not point_in_circle(pupil_circle[0], pupil_circle[1],
                                   pupil_circle[2], p_col, p_row) and \
                point_in_circle(ext_circle[0], ext_circle[1], ext_circle[2],
                                p_col, p_row):
                theta = angle_v(ext_circle[0], ext_circle[1], p_col, p_row)
                if theta >= -50 and theta <= 50:
                    rois['right-side']['img'][p_row, p_col] = img[p_row, p_col]
                if theta >= 130 or theta <= -130:
                    rois['left-side']['img'][p_row, p_col] = img[p_row, p_col]
                if theta >= -140 and theta <= -40:
                    rois['bottom']['img'][p_row, p_col] = img[p_row, p_col]
                rois['complete']['img'][p_row, p_col] = img[p_row, p_col]

    rois['right-side']['ext_circle'] = \
        (0, int(1.25 * ext_circle[2]), int(ext_circle[2]))
    rois['left-side']['ext_circle'] = \
        (int(1.25 * ext_circle[2]),
         int(1.25 * ext_circle[2]),
         int(ext_circle[2]))
    rois['bottom']['ext_circle'] = \
        (int(1.25 * ext_circle[2]), 0, int(ext_circle[2]))
```

```
    rois['complete']['ext_circle'] = \
        (int(1.25 * ext_circle[2]),
         int(1.25 * ext_circle[2]),
         int(ext_circle[2]))

    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        tx = rois[pos]['ext_circle'][0] - ext_circle[0]
        ty = rois[pos]['ext_circle'][1] - ext_circle[1]
        rois[pos]['pupil_circle'] = (int(tx + pupil_circle[0]),
                                     int(ty + pupil_circle[1]),
                                     int(pupil_circle[2]))
        M = np.float32([[1, 0, tx], [0, 1, ty]])
        rois[pos]['img'] = cv2.warpAffine(
            rois[pos]['img'], M,
            (img.shape[1], img.shape[0]))

    rois['right-side']['img'] = \
        rois['right-side']['img'][0:2.5 * ext_circle[2], 0:1.25 * ext_circle[2]]
    rois['left-side']['img'] = \
        rois['left-side']['img'][0:2.5 * ext_circle[2], 0:1.25 * ext_circle[2]]
    rois['bottom']['img'] = \
        rois['bottom']['img'][0:1.25 * ext_circle[2], 0:2.5 * ext_circle[2]]
    rois['complete']['img'] = \
        rois['complete']['img'][0:2.5 * ext_circle[2], 0:2.5 * ext_circle[2]]

    if show:
        plt.subplot(2, 2, 1), plt.imshow(rois['right-side']['img'], cmap='gray')
        plt.title('right-side'), plt.xticks([]), plt.yticks([])
        plt.subplot(2, 2, 2), plt.imshow(rois['left-side']['img'], cmap='gray')
        plt.title('left-side'), plt.xticks([]), plt.yticks([])
        plt.subplot(2, 2, 3), plt.imshow(rois['bottom']['img'], cmap='gray')
        plt.title('bottom'), plt.xticks([]), plt.yticks([])
        plt.subplot(2, 2, 4), plt.imshow(rois['complete']['img'], cmap='gray')
        plt.title('complete'), plt.xticks([]), plt.yticks([])
        plt.show()

    return rois

def load_keypoints(sift, rois, show=False):
    bf = cv2.BFMatcher()
    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        rois[pos]['kp'] = sift.detect(rois[pos]['img'], None)

        # Create image with non-filtered keypoints
        rois[pos]['img_kp_init'] = cv2.drawKeypoints(
            rois[pos]['img'], rois[pos]['kp'],
            color=(0, 255, 0), flags=0,
            outImage=None)
        cv2.circle(
            rois[pos]['img_kp_init'],
            (rois[pos]['pupil_circle'][0], rois[pos]['pupil_circle'][1]),
            rois[pos]['pupil_circle'][2], (0, 0, 255), 1)
```

```
        cv2.circle(
            rois[pos]['img_kp_init'],
            (rois[pos]['ext_circle'][0], rois[pos]['ext_circle'][1]),
            rois[pos]['ext_circle'][2], (0, 255, 255), 1)

        # Filter detected keypoints
        inside = 0
        outside = 0
        wrong_angle = 0
        for kp in rois[pos]['kp'][:]:
            c_angle = angle_v(rois[pos]['ext_circle'][0],
                              rois[pos]['ext_circle'][1],
                              kp.pt[0], kp.pt[1])
            if point_in_circle(rois[pos]['pupil_circle'][0],
                               rois[pos]['pupil_circle'][1],
                               rois[pos]['pupil_circle'][2] + 3,
                               kp.pt[0], kp.pt[1]):
                rois[pos]['kp'].remove(kp)
                inside += 1
            elif not point_in_circle(rois[pos]['ext_circle'][0],
                                     rois[pos]['ext_circle'][1],
                                     rois[pos]['ext_circle'][2] - 5,
                                     kp.pt[0], kp.pt[1]):
                rois[pos]['kp'].remove(kp)
                outside += 1
            elif (pos == 'right-side' and (c_angle <= -45 or c_angle >= 45)) or \
                 (pos == 'left-side' and (c_angle <= 135 and c_angle >= -135)) or \
                 (pos == 'bottom' and (c_angle <= -135 or c_angle >= -45)):
                rois[pos]['kp'].remove(kp)
                wrong_angle += 1

        # Create images with filtered keypoints
        rois[pos]['img_kp_filtered'] = cv2.drawKeypoints(
            rois[pos]['img'], rois[pos]['kp'],
            color=(0, 255, 0), flags=0,
            outImage=None)
        cv2.circle(
            rois[pos]['img_kp_filtered'],
            (rois[pos]['pupil_circle'][0], rois[pos]['pupil_circle'][1]),
            rois[pos]['pupil_circle'][2], (0, 0, 255), 1)
        cv2.circle(
            rois[pos]['img_kp_filtered'],
            (rois[pos]['ext_circle'][0], rois[pos]['ext_circle'][1]),
            rois[pos]['ext_circle'][2], (0, 255, 255), 1)

    # Show keypoints images
    if show:
        i = 0
        for pos in ['right-side', 'left-side', 'bottom']:
            plt.subplot(3, 2, 2 * i + 1), \
            plt.imshow(rois[pos]['img_kp_init'])
            plt.xticks([]), plt.yticks([])
```

```
            plt.subplot(3, 2, 2 * i + 2), \
            plt.imshow(rois[pos]['img_kp_filtered'])
            plt.xticks([]), plt.yticks([])
            i += 1
        plt.show()

def load_descriptors(sift, rois):
    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        rois[pos]['kp'], rois[pos]['des'] = \
            sift.compute(rois[pos]['img'], rois[pos]['kp'])

def getall_matches(rois_1, rois_2, dratio,
                   stdev_angle, stdev_dist, show=False):
    img_matches = []
    numberof_matches = {'right-side': 0,
                        'left-side': 0,
                        'bottom': 0,
                        'complete': 0}

    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        if not rois_1[pos]['kp'] or not rois_2[pos]['kp']:
            print "KeyPoints not found in one of rois_x[pos]['kp'] !!!"
            print " -->", pos, len(rois_1[pos]['kp']), len(rois_2[pos]['kp'])
        else:
            matches = get_matches(rois_1[pos], rois_2[pos],
                                  dratio, stdev_angle, stdev_dist)
            numberof_matches[pos] = len(matches)

        if show:
            print "{0} matches: {1}".format(pos, str(len(matches)))
            crt_image = cv2.drawMatchesKnn(
                rois_1[pos]['img'], rois_1[pos]['kp'],
                rois_2[pos]['img'], rois_2[pos]['kp'],
                [matches], flags=2, outImg=None)

            img_matches.append(crt_image)
            cv2.imshow('matches', crt_image)
            ch = cv2.waitKey(0)
            cv2.destroyAllWindows()
    if numberof_matches['complete'] < 30:
        vidget("Comparison was negative! Please try again!", "#ff0000")
    else:
       points.vidget("Your IRIS was verified, please choose two fingerprint images to continue
verification!")

def get_matches(roipos_1, roipos_2,
                dratio, stdev_angle, stdev_dist):
    if not roipos_1['kp'] or not roipos_2['kp']:
        print "KeyPoints not found in one of roipos_x['kp'] !!!"
        return []

    bf = cv2.BFMatcher()
```

```
matches = bf.knnMatch(roipos_1['des'], roipos_2['des'], k=2)
kp1 = roipos_1['kp']
kp2 = roipos_2['kp']

diff_dist_1 = roipos_1['ext_circle'][2] - roipos_1['pupil_circle'][2]
diff_dist_2 = roipos_2['ext_circle'][2] - roipos_2['pupil_circle'][2]

diff_angles = []
diff_dists = []
filtered = []
for m, n in matches:
    if (m.distance / n.distance) > dratio:
        continue

    x1, y1 = kp1[m.queryIdx].pt
    x2, y2 = kp2[m.trainIdx].pt

    angle_1 = angle_v(
        x1, y1,
        roipos_1['pupil_circle'][0],
        roipos_1['pupil_circle'][1])
    angle_2 = angle_v(
        x2, y2,
        roipos_2['pupil_circle'][0],
        roipos_2['pupil_circle'][1])
    diff_angle = angle_1 - angle_2
    diff_angles.append(diff_angle)

    dist_1 = distance(x1, y1,
                roipos_1['pupil_circle'][0],
                roipos_1['pupil_circle'][1])
    dist_1 = dist_1 - roipos_1['pupil_circle'][2]
    dist_1 = dist_1 / diff_dist_1

    dist_2 = distance(x2, y2,
                roipos_2['pupil_circle'][0],
                roipos_2['pupil_circle'][1])
    dist_2 = dist_2 - roipos_2['pupil_circle'][2]
    dist_2 = dist_2 / diff_dist_2

    diff_dist = dist_1 - dist_2
    diff_dists.append(diff_dist)

    filtered.append(m)

# Remove bad matches
if True and filtered:
    median_diff_angle = median(diff_angles)
    median_diff_dist = median(diff_dists)
    # print "median dist:", median_diff_dist
    for m in filtered[:]:
        x1, y1 = kp1[m.queryIdx].pt
```

```
            x2, y2 = kp2[m.trainIdx].pt

            angle_1 = angle_v(
                x1, y1,
                roipos_1['pupil_circle'][0],
                roipos_1['pupil_circle'][1])
            angle_2 = angle_v(
                x2, y2,
                roipos_2['pupil_circle'][0],
                roipos_2['pupil_circle'][1])
            diff_angle = angle_1 - angle_2

            good_diff_angle = \
                (diff_angle > median_diff_angle - stdev_angle and \
                 diff_angle < median_diff_angle + stdev_angle)

            dist_1 = distance(x1, y1,
                              roipos_1['pupil_circle'][0],
                              roipos_1['pupil_circle'][1])
            dist_1 = dist_1 - roipos_1['pupil_circle'][2]
            dist_1 = dist_1 / diff_dist_1

            dist_2 = distance(x2, y2,
                              roipos_2['pupil_circle'][0],
                              roipos_2['pupil_circle'][1])
            dist_2 = dist_2 - roipos_2['pupil_circle'][2]
            dist_2 = dist_2 / diff_dist_2

            diff_dist = dist_1 - dist_2
            good_dist = (diff_dist > median_diff_dist - stdev_dist and \
                         diff_dist < median_diff_dist + stdev_dist)

            if good_diff_angle and good_dist:
                continue

            filtered.remove(m)

    return filtered

def angle_v(x1, y1, x2, y2):
    return math.degrees(math.atan2(-(y2 - y1), (x2 - x1)))

def distance(x1, y1, x2, y2):
    dst = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return dst

def mean(x):
    sum = 0.0
    for i in range(len(x)):
        sum += x[i]
    return sum / len(x)
```

```
def median(x):
    return np.median(np.array(x))

def standard_dev(x):
    if not x:
        print 'Error: empty list parameter in standard_dev() !'
        print inspect.getouterframes(inspect.currentframe())[1]
        print
        return None, None
    m = mean(x)
    sumsq = 0.0
    for i in range(len(x)):
        sumsq += (x[i] - m) ** 2
    return m, math.sqrt(sumsq / len(x))

def load_rois_from_bin(bin_path):
    with gzip.open(bin_path, 'rb') as bin_file:
        rois = pickle.load(bin_file)
    unpickle_rois(rois)
    return rois

def unpickle_rois(rois):
    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        rois[pos]['kp'] = unpickle_keypoints(rois[pos]['kp'])

def unpickle_keypoints(array):
    keypoints = []
    for point in array:
        temp_kp = cv2.KeyPoint(x=point[0][0], y=point[0][1], _size=point[1],
                               _angle=point[2], _response=point[3],
                               _octave=point[4], _class_id=point[5])
        keypoints.append(temp_kp)
    return keypoints

def pickle_rois(rois):
    for pos in ['right-side', 'left-side', 'bottom', 'complete']:
        rois[pos]['kp'] = pickle_keypoints(rois[pos]['kp'])

def pickle_keypoints(keypoints):
    unfolded = []
    for point in keypoints:
        temp = (point.pt, point.size, point.angle, point.response,
                point.octave, point.class_id)
        unfolded.append(temp)

    return unfolded

def start():
    root.destroy()
    compare_images(inPut.get(), outPut.get())
```

```
def openRef(event):
    options = {}
    options['defaultextension'] = ''
    options['filetypes'] = [('text files', '.txt')]
    options['initialdir'] = 'C:\\'
    options['parent'] = root
    options['title'] = 'Open Input File'

    input = tkFileDialog.askopenfilename()
    if input:
        inPut.set(input)

def openVer(event):
    options = {}
    options['defaultextension'] = ''
    options['filetypes'] = [('text files', '.txt')]
    options['initialdir'] = 'C:\\'
    options['parent'] = root
    options['title'] = 'Open Output File'

    output = tkFileDialog.askopenfilename()
    if output:
        outPut.set(output)

def vidget(message, color):
    global root
    root = Tk()
    root.title("IRIS")
    root.geometry("800x325")
    path = "iris.jpg"
    img = ImageTk.PhotoImage(Image.open(path))
    panel = Label(image=img)
    panel.place(relx=.0, rely=.0)
    label = Label(text=message, fg="#eee", bg=color)
    label.place(relx=.4, rely=.1)

    global inPut
    global outPut
    global autoGraph
    inPut = StringVar()
    inPut.set('')
    outPut = StringVar()
    outPut.set('')
    autoGraph = StringVar()
    autoGraph.set('')

    inEntry = Entry(root, textvariable=inPut)
    inEntry.place(relx=.1, rely=.3, height=30, width=400)
    outEntry = Entry(root, textvariable=outPut)
    outEntry.place(relx=.1, rely=.5, height=30, width=400)

    getInFile = Button(root, text='Open Reference')
```

```
    getInFile.place(relx=.7, rely=.3, height=30, width=100)
    getOutFile = Button(root, text='Open Verificable')
    getOutFile.place(relx=.7, rely=.5, height=30, width=100)

    getInFile.bind('<1>', openRef)
    getOutFile.bind('<1>', openVer)

    sign = Button(root, text='Start', command=start)
    sign.place(relx=.4, rely=.8, height=30, width=200)

    root.mainloop()

vidget("Please, choose two IRIS images for compare!", "#333")
```

ДОДАТОК Б
Лістинг коду другого модулю програми

Файл points.py

```
from Tkinter import *
import tkFileDialog
import points

def start():
   points.checkFinger(inPut.get(),outPut.get())

def openRef(event):
   options={}
   options['defaultextension'] = ''
   options['filetypes'] = [('text files', '.txt')]
   options['initialdir'] = 'C:\\'
   options['parent'] = root
   options['title'] = 'Open Input File'
   input=tkFileDialog.askopenfilename()
   if input:
      inPut.set(input)

def openVer(event):
   options={}
   options['defaultextension'] = ''
   options['filetypes'] = [('text files', '.txt')]
   options['initialdir'] = 'C:\\'
   options['parent'] = root
   options['title'] = 'Open Output File'
   output=tkFileDialog.askopenfilename()
   if output:
      outPut.set(output)

global root
root=Tk()
root.title("FingerPrint")

global inPut
global outPut
global autoGraph
inPut=StringVar()
inPut.set('')
outPut=StringVar()
outPut.set('')
autoGraph=StringVar()
autoGraph.set('')

inEntry=Entry(root, textvariable=inPut)
outEntry=Entry(root, textvariable=outPut)
```

```
getInFile=Button(root, text='Open Reference')
getOutFile=Button(root, text='Open Verificable')

getInFile.bind('<1>',openRef)
getOutFile.bind('<1>',openVer)

sign=Button(root, text='Start', command=start)

inEntry.grid(row=0,column=0)
outEntry.grid(row=1,column=0)

getInFile.grid(row=0,column=1)
getOutFile.grid(row=1,column=1)

sign.grid(row=3,column=1)

root.mainloop()

from Tkinter import *
from PIL import Image
import skeletization as sk

def binaryImage(img):
    binaryImg=[]
    for i in range(img.size[0]):
        tmp=[]
        for j in range(img.size[1]):
            t=img.getpixel((i,j))
            p=t[0]*0.2990+t[1]*0.5870+t[2]*0.1140
            if p>128:
                p=1
            else:
                p=0
            tmp.append(p)
        binaryImg.append(tmp)
    return binaryImg

def removeDouble(x,y):
    z=[]
    for i in x:
        c=True
        for j in y:
            if i==j:
                c=False
        if c:
            z.append(i)
    for i in y:
        c=True
        for j in x:
            if i==j:
                c=False
```

```
        if c:
            z.append(i)
    return z

def deleteNoisePoint(r):
    tmp=[]
    tmp2=[]
    for i in r[1]:
        x=range(i[0]-5,i[0]+5)
        y=range(i[1]-5,i[1]+5)

        for j in r[0]:
            if j[0] in x and j[1] in y:
                tmp.append(i)
                tmp2.append(j)
    return (removeDouble(r[0],tmp2),removeDouble(r[1],tmp),r[2])#r[2]

def matchingSimilarPoints(r, v):
    all=0
    match=0
    for i in v[0]:
        x=range(i[0]-15,i[0]+15)
        y=range(i[1]-15,i[1]+15)
        all+=1
        for j in r[0]:
            if j[0] in x and j[1] in y:
                match+=1
                break
    for i in v[1]:
        x=range(i[0]-15,i[0]+15)
        y=range(i[1]-15,i[1]+15)
        all+=1
        for j in r[1]:
            if j[0] in x and j[1] in y:
                match+=1
                break
    for i in v[2]:
        x=range(i[0]-15,i[0]+15)
        y=range(i[1]-15,i[1]+15)
        all+=1
        for j in r[2]:
            if j[0] in x and j[1] in y:
                match+=1
                break
    return (match,all)

def countPoints(img, x, y):
    c=0
    for i in range(x-1,x+2):
        for j in range(y-1,y+2):
            if img[i][j]==0:
                c+=1
```

```
    return c-1

def countPPoints(img, x, y):
    c=0
    for i in range(x-1,x+1):
        for j in range(y-1,y+1):
            if img[i][j]==0:
                c+=1
    return c

def createListOfPoints(img):
    x=len(img)
    y=len(img[0])
    branchPoint=[]
    endPoint=[]
    pPoint=[]
    for i in range(x):
        for j in range(y):
            if img[i][j]==0:
                t=countPoints(img, i, j)
                if t==1:
                    endPoint.append((i,j))
                if t==3:
                    branchPoint.append((i,j))
    for i in range(x):
        for j in range(y):
            if img[i][j]==0:
                t=countPPoints(img, i,j)
                if t==4:
                    pPoint.append((i,j))
    return (branchPoint, endPoint, pPoint)

def checkFinger(r, v):
    reference=Image.open(r)
    ref=binaryImage(reference)
    sk.skeletization(ref)
    rp=createListOfPoints(ref)
    rp=deleteNoisePoint(rp)

    verf=Image.open(v)
    ver=binaryImage(verf)
    sk.skeletization(ver)
    vp=createListOfPoints(ver)
    vp=deleteNoisePoint(vp)

    res=matchingSimilarPoints(rp,vp)
    r=(res[0]*100)/(res[1]*1.)

    root=Tk()
    w=len(ver)
    h=len(ver[0])
    C=Canvas(root, width=w*2, height=h)
```

```
    for i in range(w):
        for j in range(h):
            if ref[i][j]==0:
                C.create_line([(i,j),(i+1,j+1)])
            if ver[i][j]==0:
                C.create_line([(i+w+1,j+1),(i+w,j)])
    for i in rp[0]:
        C.create_oval([(i[0]-3,i[1]-3),(i[0]+3,i[1]+3)],outline="#ff0000")
    for i in rp[1]:
        C.create_rectangle([(i[0]-3,i[1]-3),(i[0]+3,i[1]+3)],outline="#0000ff")
    for i in rp[2]:
        C.create_rectangle([(i[0]-3,i[1]-3),(i[0]+3,i[1]+3)],outline="#009900")
    for i in vp[0]:
        C.create_oval([(i[0]-3+w,i[1]-3),(i[0]+3+w,i[1]+3)],outline="#ff0000")
    for i in vp[1]:
        C.create_rectangle([(i[0]-3+w,i[1]-3),(i[0]+3+w,i[1]+3)],outline="#0000ff")
    for i in vp[2]:
        C.create_rectangle([(i[0] - 3 + w, i[1] - 3), (i[0] + 3 + w, i[1] + 3)], outline="#009900")

    C.create_text((w, h*0.05), fill="#0000ff", text="Probabiliti ~
"+str(round(r,2))+"%",font='Arial,72')
    if r>75:
        C.create_text((w, h*0.95), fill="#009900", text="Result - Positive", font='Arial,72')
    else:
        C.create_text((w, h*0.95), fill="#ff0000", text="Result - Negative", font='Arial,72')
    C.pack()
    root.mainloop()
```

Файл skeletization.py

```
def skeletization(img):
    w=len(img)
    h=len(img[0])
    count=1
    while count!=0:
        count=mainDelete(img,w,h)
        if count:
            noiseDelete(img,w,h)

def mainDelete(img,w,h):
    count=0
    for i in range(1,h-1):
        for j in range(1,w-1):
            if img[j][i]==0:
                if mainDeletable(img,j,i):
                    img[j][i]=1
                    count+=1
    return count

def noiseDelete(img,w,h):
```

```
    for i in range(1,h-1):
        for j in range(1,w-1):
            if img[j][i]==0:
                if noiseDeletable(img,j,i):
                    img[j][i]=1

def noise(a):
    t=[[1,1,1,1,0,1,1,1,1],

       [1,1,1,1,0,1,1,0,0],
       [1,1,1,0,0,1,0,1,1],
       [0,0,1,1,0,1,1,1,1],
       [1,1,0,1,0,0,1,1,1],

       [1,1,1,1,0,1,0,0,1],
       [0,1,1,0,0,1,1,1,1],
       [1,0,0,1,0,1,1,1,1],
       [1,1,1,1,0,0,1,1,0],

       [1,1,1,1,0,1,0,0,0],
       [0,1,1,0,0,1,0,1,1],
       [0,0,0,1,0,1,1,1,1],
       [1,1,0,1,0,0,1,1,0]]
    for i in t:
        if a==i:
            return True

def main(a):
    t123457=[1,1,0,0,1,0]
    t013457=[1,1,1,0,0,0]
    t134567=[0,1,0,0,1,1]
    t134578=[0,0,0,1,1,1]
    t0123457=[1,1,1,0,0,0,0]
    t0134567=[1,0,1,0,0,1,0]
    t1345678=[0,0,0,0,1,1,1]
    t1234578=[0,1,0,0,1,0,1]

    t=[a[1],a[2],a[3],a[4],a[5],a[7]]
    if t == t123457:
        return True
    t=[a[0],a[1],a[3],a[4],a[5],a[7]]
    if t == t013457:
        return True
    t=[a[1],a[3],a[4],a[5],a[6],a[7]]
    if t == t134567:
        return True
    t=[a[1],a[3],a[4],a[5],a[7],a[8]]
    if t == t134578:
        return True
    t=[a[0],a[1],a[2],a[3],a[4],a[5],a[7]]
    if t == t0123457:
        return True
```

```
    t=[a[1],a[3],a[4],a[5],a[6],a[7],a[8]]
    if t == t1345678:
        return True
    t=[a[0],a[1],a[3],a[4],a[5],a[6],a[7]]
    if t == t0134567:
        return True
    t=[a[1],a[2],a[3],a[4],a[5],a[7],a[8]]
    if t == t1234578:
        return True

def mainDeletable(img,x,y):
    a=[]
    for i in range(y-1,y+2):
        for j in range(x-1,x+2):
            a.append(img[j][i])
    return main(a)

def noiseDeletable(img,x,y):
    a=[]
    for i in range(y-1,y+2):
        for j in range(x-1,x+2):
            a.append(img[j][i])
    return noise(a)
```