```
import numpy as np
import cv2
import os
import sys
import math
import random
import cPickle as pickle
import copy
import gzip
import inspect
import itertools

from matplotlib import pyplot as plt


def compare_images(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
    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
            circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 1,
                                       np.array([]), param1, param2)
            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, 1, 1,
                                       np.array([]), 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:
            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()

    return numberof_matches


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


if __name__ == "__main__":

    # Specify 2 image paths
    filepath1 = r'./S2005R07.jpg'
    filepath2 = r'./S2005R09.jpg'

    if os.path.isfile(filepath1) and os.path.isfile(filepath2):
        compare_images(filepath1, filepath2)
```

BUSINESS IDEAS

BUSINESS TYPE: JOINT VENTURE

A joint venture is a business arrangement in which two or more parties agree to pool their resources for the purpose of accomplishing a specific goal. The participants in a joint venture retain their individual business identity, but they share the revenues, expenses, and control of the venture.

As a SACCO the venture can be termed as a Savings and Credit Cooperative Organization that challenges its members to work together to accomplish certain goals.

The great advantage of being in a Sacco is that all members have equal voting rights with the organization despite having different savings with the Sacco.

Saccos are advantaged because they solve community-based problems especially during the COVID-19 pandemic. An example of a community-based solution Super Metro came up with is the use of "civil" ways of conductor and customer relations. This does not disadvantage the customers who want to board.

THE IDEA IS BASED ON **SUPER METRO**

IDEA: **ONLINE PAYMENT MANAGEMENT SYSTEM**

SOLUTION: Allows for the management of payment within the bus as the bus number must be provided.

This business idea allows for entreprenurship that was highly championed by:

Alfred D Chandler

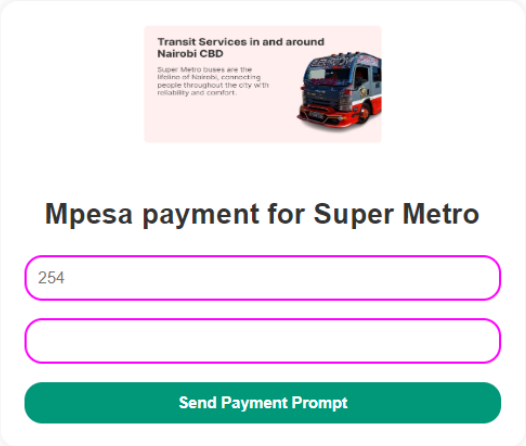
1. Chandler's views on entrepreneurship are closely tied to his analysis of the evolution of business structures and the rise of managerial capitalism.

   He was able to emphasize on

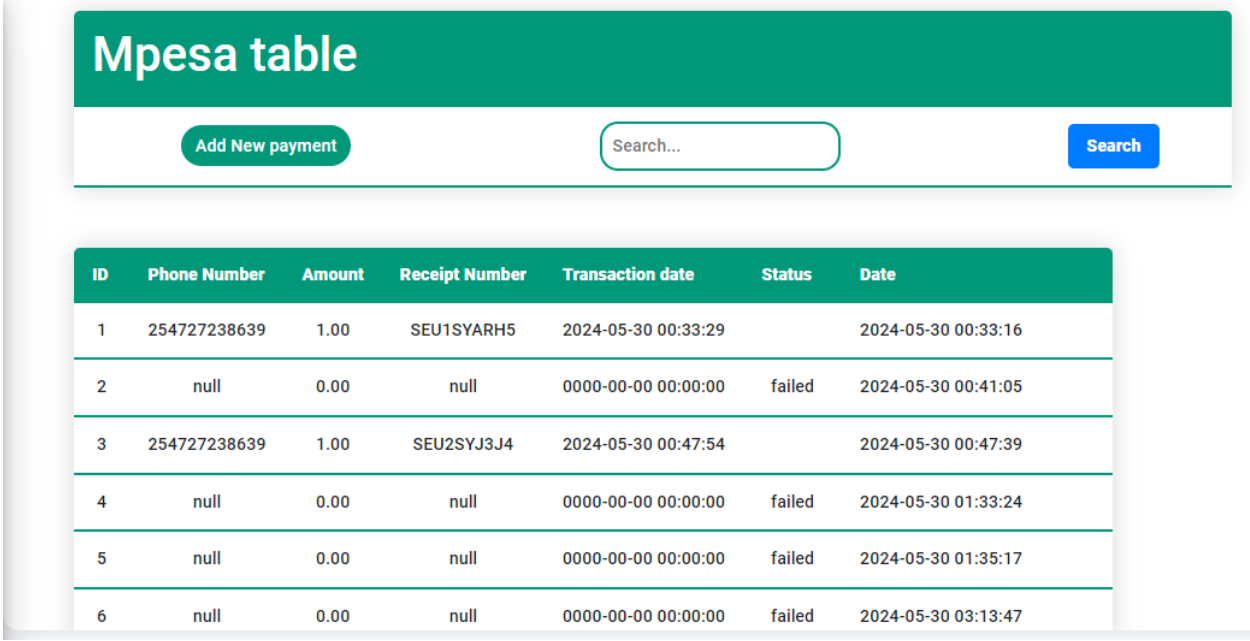
   The Role of Entrepreneurs in Early Industrialization: Chandler emphasized that entrepreneurs played a crucial role during the early phases of industrialization. They were the pioneers who identified new opportunities, mobilized resources, and established new enterprises.

SECURITY

- Payments can be easily tracked by the bus attendant and customers



A management System for the payments will pop up as below



According to the S.W.O.T analysis. The system will have the strength of allowing efficient payment from customers on the bus without having to read and type numbers into the phone. They only need to type their own number in the system and the bus number with the amount. The system will be able to track the loyal customers later, handing them trip payment reprieves sometimes. This will be a strategy of attracting more people to have the payments made online.

A huge strength would be that it ensures that all the payment amounts are tracked. The time and date are tracked therefore the bus can be analyzed in terms of the profitability range.

WEAKNESSES

A weakness can be that the customer being ferried may not have data bundles or have access to a smart phone, which is rare in today's world of technology.

OPPORTUNITIES

Payments can be made online for other services like the transport of goods and services.

THREATS

Some of the threats on this idea can be mostly of high network tariffs that the government may want to impose on mobile data use.

SOURCES OF FINANCING

A source of financing can be from Retained Earnings, termed as a firm's net earnings after accounting for dividends. This source of finance can be used to accommodate for the integration of the payment system since all the buses have their till numbers.

AFTERSCHOOL TRAINING TOOLKIT

Tutoring to Enhance Science Skills

Tutoring Two: Learning to Make Data Tables

Sample Data for Data Tables

Use these data to create data tables following the Guidelines for Making a Data Table and Checklist for a Data Table.

Example 1: Pet Survey (GR 2–3)
Ms. Hubert's afterschool students took a survey of the 600 students at Morales Elementary School. Students were asked to select their favorite pet from a list of eight animals. Here are the results.

Lizard 25, Dog 250, Cat 115, Bird 50, Guinea pig 30, Hamster 45, Fish 75, Ferret 10

Example 2: Electromagnets—Increasing Coils (GR 3–5)
The following data were collected using an electromagnet with a 1.5 volt battery, a switch, a piece of #20 insulated wire, and a nail. Three trials were run. *Safety precautions in repeating this experiment include using safety goggles or safety spectacles and avoiding short circuits.*

Number of Coils	Number of Paperclips
5	3, 5, 4
10	7, 8, 6
15	11, 10, 12
20	15, 13, 14

Example 3: pH of Substances (GR 5–10)
The following are pH values of common household substances taken by three different teams using pH probes. *Safety precautions in repeating this experiment include hooded ventilation, chemical-splash safety goggles, gloves, and apron. Do not use bleach, ammonia, or strong acids with children.*

Lemon juice 2.4, 2.0, 2.2; Baking soda (1 Tbsp) in Water (1 cup) 8.4, 8.3, 8.7; Orange juice 3.5, 4.0, 3.4; Battery acid 1.0, 0.7, 0.5; Apples 3.0, 3.2, 3.5; Tomatoes 4.5, 4.2, 4.0; Bottled water 6.7, 7.0, 7.2; Milk of magnesia 10.5, 10.3, 10.6; Liquid hand soap 9.0, 10.0, 9.5; Vinegar 2.2, 2.9, 3.0; Household bleach 12.5, 12.5, 12.7; Milk 6.6, 6.5, 6.4; Household ammonia 11.5, 11.0, 11.5; Lye 13.0, 13.5, 13.4; and Sodium hydroxide 14.0, 14.0, 13.9; Anti-freeze 10.1, 10.9, 9.7; Windex 9.9. 10.2, 9.5; Liquid detergent 10.5, 10.0, 10.3; and Cola 3.0, 2.5, 3.2

Teaching tip: *The pH scale is from 0 to 14. Have students make two data tables, one with the data as given and one with the pH scale 0 to 14 with the substances' average pH in rank order on the scale (Battery acid at the lower end and Sodium hydroxide at the upper end) or create a pH graphic organizer.*

Example 4: Automobile Land Speed Records (GR 5-10)
In the first recorded automobile race in 1898, Count Gaston de Chasseloup-Laubat of Paris, France, drove 1 kilometer in 57 seconds for an average speed of 39.2 miles per hour (mph) or 63.1 kilometers per hour (kph). In 1904, Henry Ford drove his Ford Arrow across frozen Lake St. Clair, MI, at an average speed of 91.4 mph. Now, the North American Eagle is trying to break a land speed record of 800 mph. The Federation International de L'Automobile (FIA), the world's governing body for motor sport and land speed records, recorded the following land speed records. (Retrieved on February 5, 2006, from http://www.landspeed.com/lsrinfo.asp.)

Speed (mph)	Driver	Car	Engine	Date
407.447	Craig Breedlove	Spirit of America	GE J47	8/5/63
413.199	Tom Green	Wingfoot Express	WE J46	10/2/64
434.22	Art Arfons	Green Monster	GE J79	10/5/64
468.719	Craig Breedlove	Spirit of America	GE J79	10/13/64
526.277	Craig Breedlove	Spirit of America	GE J79	10/15/65
536.712	Art Arfons	Green Monster	GE J79	10/27/65
555.127	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/2/65
576.553	Art Arfons	Green Monster	GE J79	11/7/65
600.601	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/15/65
622.407	Gary Gabelich	Blue Flame	Rocket	10/23/70
633.468	Richard Noble	Thrust 2	RR RG 146	10/4/83
763.035	Andy Green	Thrust SSC	RR Spey	10/15/97

Example 5: Distance and Time (GR 8-10)
The following data were collected using a car with a water clock set to release a drop in a unit of time and a meter stick. The car rolled down an inclined plane. Three trials were run. Create a data table with an average distance column and an average velocity column, create an average distance-time graph, and draw the best-fit line or curve. Estimate the car's distance traveled and velocity at six drops of water. Describe the motion of the car. Is it going at a constant speed, accelerating, or decelerating? How do you know?

Time (drops of water)	Distance (cm)
1	10,11,9
2	29, 31, 30
3	59, 58, 61
4	102, 100, 98
5	122, 125, 127

© 2006 WGBH Educational Foundation. All rights reserved.