דו"ח סיכום הפרוייקט בקורס עיבוד תמונות סיפרתי

שם: אינה קרייזמן ת.ז: 314467812

הפרוייקט שאותו עשיתי הוא: H. Document Image Tools.
בפרוייקט עבדתי עם הספריות: OpenCV ו-Manuscript. יש להוסיף את הקבצים הבאים לספריית ה-Manuscript:

1. MorphologyTextLineExtractor.h
2. MorphologyTextLineExtractor.cpp
3. StrokeWidthEstimatorDerived.h
4. StrokeWidthEstimatorDerived.cpp
5. TextLineHeightEstimator.h
6. TextLineHeightEstimatorDerived.h
7. TextLineHeightEstimatorDerived.cpp

כמו כן, ביצעתי שינוי בקובץ TextLine.h, לכן יש לעדכן אותו.

הקובץ DocImgTools.cpp הוא קובץ ה-main ואינו חלק מהספריה, אך יש צורך בשתי הספריות הנ"ל על מנת להריץ את התוכנית.

הרצת התוכנית מתבצעת ע"י הפקודה DocImgTools, לאחריה האות H או W (כדי לחשב text line height או stroke width בהתאמה), ולאחר מכן נתיב התמונה.

בקובץ DocImgTools.cpp פונקציית ה-main בודקת מה הקלט בארגומנט הראשון ומפעילה את הפונקציה המתאימה לקלט. לאחר מכן, קולטת תמונה מהנתיב שנקלט בארגומנט השני ויוצרת אובייקט מסוג DImage אשר מאותחל ע"י התמונה. כמו כן, היא יוצרת אובייקט מסוג TextLineHeightEstimator או StrokeWidthEstimator ומפעילה את הפונקציה textLineHeight או strokeWidth של Dimage, בהתאם למה שביקשנו לחשב, ולאחר מכן מדפיסה את הערך שחזר מהפונקציה.

<u>סעיף 1:</u>
בסעיף 1 התבקשתי לחשב את רוחב הקו הממוצע. הקבצים 3 ו-4 אחראיים על ביצוע חישוב זה. הפונקציה estimate פועלת על תמונה בגווני אפור. פונקציה זו מחשבת את רוחב הקו הממוצע ומחזירה אותו. בנוסף, היא מחשבת ומדפיסה את ממוצע הגבהים של השינויים בערכי הפיקסלים, את סטיית התקן של גבהים אלה ואת החציון של רוחב הקו.

<u>הסבר הקוד:</u>
יצירת וקטורים של ערכי הפיקסלים עבור השורות, העמודות והאלכסונים:

```
vector<vector<uchar> > vertical(cols, vector<uchar>(rows,0));
vector<vector<uchar> > horizontal(rows, vector<uchar>(cols,0));
vector<vector<uchar> > plus45(rows+cols-1, vector<uchar>(0));
vector<vector<uchar> > minus45(rows+cols-1, vector<uchar>(0));

//horizontal and vertical
for(i=0; i<rows; i++){
        for(j=0; j<cols; j++){
                horizontal[i][j]=img.at<uchar>(i,j);
                vertical[j][i]=horizontal[i][j];
        }
}

//-45 and +45
int k;
for(k=0; k<rows; k++){
        for(i=k, j=0; i<rows && j<cols ;i++, j++){
                minus45[k].push_back(horizontal[i][j]);
                plus45[k].push_back(horizontal[i][cols-1-j]);
```

```
            }
        }
        int idx=k-1;
        for(k=1; k<cols; k++){
            for(i=0, j=k; i<rows && j<cols; i++, j++){
                minus45[idx+k].push_back(horizontal[i][j]);
                plus45[idx+k].push_back(horizontal[i][cols-1-j]);
            }
        }
```

קריאה לפונקציה המחשבת את הגבהים ורוחב הקו עבור כל גובה ומאחסנת את הנתונים בוקטורים heights ו-widths בהתאמה:

```
calcHeightsAndWidths(horizontal, &heights, &widths);
calcHeightsAndWidths(vertical, &heights, &widths);
calcHeightsAndWidths(plus45, &heights, &widths);
calcHeightsAndWidths(minus45, &heights, &widths);
```

חישוב ממוצע הגבהים וסטיית התקן:

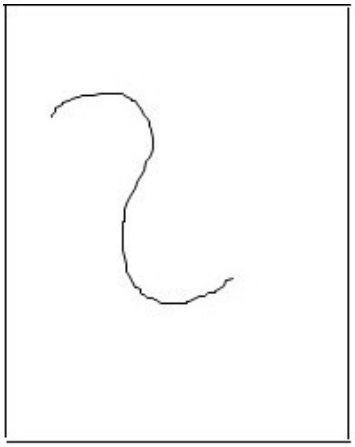
```
        if(sizeH>0){
            float sum=0;
            for(i=0;i<sizeH;i++){
                sum=sum+heights[i];
            }
            avgH=sum/sizeH;

            float sumOfSquares=0;
            for(i=0;i<sizeH;i++){
                sumOfSquares=sumOfSquares+pow((heights[i]-avgH), 2);
            }
            stnDev=sqrt(sumOfSquares/sizeH);
        }
```

חישוב ממוצע וחציון רוחב הקו:

```
        vector<int> newWidths;
        float avgW=0, median=0;
        int size= widths.size();
        if(size>0){
            float sum2=0;
            for(i=0;i<size;i++){
                if(heights[i] > avgH-(stnDev+5)){
                        newWidths.push_back(widths[i]);
                    sum2=sum2+widths[i];
                }
            }
            int sizeW=newWidths.size();
            avgW=sum2/sizeW;

            sort(newWidths.begin(), newWidths.end());
            int medIdx=sizeW/2;
            if(sizeW%2==0){
                median = (newWidths[medIdx]+newWidths[medIdx+1])/2;
            }
            else
                median = newWidths[medIdx+1];
        }
```

דוגמאות לפלטים:

רוחב קו זהה, צבעים שונים וצורת קו שונה:



```
The average height is: 204.93
The standard deviation of the heights is: 55.4917
The median of the stroke widths is: 1
The average stroke width is: 2.03156
```

```
The average height is: 96.6643
The standard deviation of the heights is: 36.8331
The median of the stroke widths is: 1
The average stroke width is: 2.00451
```

קווים בעובי שונה:

```
The average height is: 183.353
The standard deviation of the heights is: 41.1898
The median of the stroke widths is: 3
The average stroke width is: 5.06916
```

```
The average height is: 161.937
The standard deviation of the heights is: 33.6248
The median of the stroke widths is: 9
The average stroke width is: 11.259
```


```
The average height is: 90.8135
The standard deviation of the heights is: 20.2343
The median of the stroke widths is: 6
The average stroke width is: 7.77853
```


```
The average height is: 73.4049
The standard deviation of the heights is: 14.304
The median of the stroke widths is: 6
The average stroke width is: 7.78731
```

סעיף 2:

בסעיף 2 הופנתי למאמר:

Morphology-based text line extraction, Jui-Chen Wu, Jun-Wei Hsieh, Yung-Sheng Chen, Machine Vision and Applications, 2008, Volume 19, Issue 3, pp 195-207.

ובעזרת האלגוריתם המתואר שם למציאת שורות טקסט בתמונה התבקשתי לחשב את הגובה הממוצע של הטקסט.
פעלתי לפי האלגוריתם המתואר במאמר על מנת למצוא את שורות הטקסט ולאחר מכן חישבתי את הגובה הממוצע.

המחלקה TextLineHeightEstimatorDerived מממשת את הפונקציה estimate אשר מחשבת את הגובה הממוצע של שורות הטקסט בתמונה. היא קוראת לפונקציה extract של המחלקה MorphologyTextLineExtractor אשר תפקידה הינו למצוא את שורות הטקסט בתמונה (לפי האלגוריתם המתואר במאמר) ולהחזיר וקטור של TextLines. לכל TextLine יש שדה הנקרא rect_, שזהו המלבן החוסם את שורת הטקסט בתמונה. בעזרת גובה המלבנים החוסמים בוצע חישוב הגובה הממוצע של שורות הטקסט.

הסבר הקוד של הפונקציה extract במחלקה MorphologyTextLineExtractor:

נרצה לעבוד על תמונה בשחור לבן, לכן נשנה אותה בהתאם. לפי האלגוריתם המתואר במאמר, נטשטש את התמונה ע"י אלמנט בגודל 3X3, ולאחר מכן נבצע פעולות closing ו-opening עם אלמנט 1X7.

```
Mat src=convertMat();
Mat blured, closed, opened, diff, dst;
blur(src, blured, Size(3,3));
Mat element = getStructuringElement( MORPH_RECT, Size(1, 7), Point(-1, -1));
morphologyEx(blured,closed,MORPH_CLOSE,element);
morphologyEx(blured,opened,MORPH_OPEN,element);
```

חישוב מטריצת ההפרשים ופעולת closing עם אלמנט בגודל 5X5, ולאחר מכן פעולת threshold.

```
absdiff(closed, opened, diff);
element = getStructuringElement( MORPH_RECT, Size(5, 5), Point(-1, -1));
morphologyEx(diff,closed,MORPH_CLOSE,element);
threshold( closed, dst, 0, 255 ,0);
```

קיבלנו מטריצה בינארית, dst, שאיתה נעבוד. המטרה היתה לזהות בתמונה המקורית חלקים עם contrast גבוה מכיוון שזה מאפיין שורות טקסט בתמונה.

נשמור בוקטור contours את חלקי התמונה שבהם יכול להיות טקסט (כלומר, חלקים שבהם זיהינו שיש חדות גבוהה). לחלקים אלה נחשב את הזוויות ומרכזי הכובד שלהם ע"י מומנטים. וכמו כן, נשמור בוקטור box מלבנים אשר עוטפים חלקים אלה.

```
//Moment-based orientation estimation
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
findContours( dst, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

/// Get the moments
vector<Moments> mu(contours.size() );
for( int i = 0; i < (int)contours.size(); i++ )
{
        mu[i] = moments( contours[i], false );
}
///  Get the mass centers and orientations:
vector<Point2f> mc( contours.size() );
vector<double> theta( contours.size() );
vector<RotatedRect> box(contours.size());
for( int i = 0; i < (int)contours.size(); i++ )
{
        mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 );
        if((mu[i].m20 - mu[i].m02)!=0){
                theta[i] = 0.5 * atan(2 * mu[i].m11 /(mu[i].m20 - mu[i].m02) );
                theta[i] = (theta[i] / M_PI) * 180;
        }
        else
```

```
                theta[i] = 0;

            // Find the minimum area enclosing bounding box
            box[i] = minAreaRect(contours[i]);
        }
```

לפי האלגוריתם שבמאמר, ישנם 3 תנאים שצריכים להתקיים כדי שחלק כלשהו ששמרנו יהיה עם טקסט (סעיף 4.2). נבצע את החישובים ונבדוק את התנאים. במידה ואחד מהתנאים לא מתקיים נמחק את החלק והמלבן שעוטף אותו.

```
//Text candidate selection
        float width=0, height=0;
        double area=0;
        for( int i = 0; i < (int)contours.size(); i++ ){
            Point2f vtx[4];
            box[i].points(vtx);
            width=box[i].size.width;
            height=box[i].size.height;
            area=contourArea(contours[i], false );
            if(area/(width*height)<=0.2 || width/height<=0.5 || area<=2.5){
                box[i]=RotatedRect();
                contours[i].clear();
            }
        }
```

לפי סעיף 4.3 במאמר, ישנם 4 תנאים אשר אם שני חלקים (contours) מקיימים אותם אזי ניתן לאחד אותם לחלק אחד. נבצע את החישובים הדרושים ונבדוק תנאים אלה. ולאחר מכן ניצור מחדש את המלבנים עבור החלקים.

```
//Text line merging
for( int i = 0; i < (int)contours.size(); i++ ){
    if(!contours[i].empty()){
        for( int j = i+1; j < (int)contours.size(); j++ ){
            if(!contours[j].empty()){
                if(shouldMerge(contours[i], contours[j], theta[i], theta[j], mc[i], mc[j], box[i], box[j])){
                    contours[j]=mergeConts(contours[i], contours[j]);
                        contours[i].clear();
                        break;
                }
            }
        }
    }
}
for( int i = 0; i < (int)contours.size(); i++ ){
        if(!contours[i].empty()){
            box[i] = minAreaRect(contours[i]);
        }
        else
            box[i]=RotatedRect();
}
```

עבור סעיף 5.1, נחשב את ה-x-projection, כדי לעשות זאת נחתוך כל חלק ונסובב במידת הצורך ונשלח לפונקציה x_projection אשר מחזירה vector עם כל הנתונים, אותו נשמור ב-x_proj.

```
        Mat M, rotated, cropped;
        float tw=85 ,th=85;
        for( int i = 0; i < (int)contours.size(); i++ ){
            if((contSize>0) && !contours[i].empty()){
                // get angle and size from the bounding box
                float angle = box[i].angle;
                Size rect_size = box[i].size;
                if (box[i].angle < -45.) {
                    angle += 90.0;
                    swap(rect_size.width, rect_size.height);
                }
                // get the rotation matrix
                M = getRotationMatrix2D(box[i].center, angle, 1.0);
                // perform the affine transformation
                warpAffine(blured, rotated, M, imgForCrop.size(), INTER_CUBIC);
                // crop the resulting image
                getRectSubPix(rotated, rect_size, box[i].center, cropped);

                vector<float> x_proj(cropped.cols);
                x_proj=x_projection(cropped);
```

נחפש את ה-valleys ונבדוק שהם מקיימים את התנאי: (הגובה של החלק) חלקי (8), כמתואר במאמר. במידה ולא, נמחק את החלק כי אינו מכיל טקסט. אם כל ה valleys בחלק מקיימים את התנאי, נמשיך לבדוק. נחשב את הרוחב הממוצע של כל אות ואת הגובה הממוצע ונבדוק את השונות בעזרת פונקציות העזר המתאימות.

```
                bool vall=checkValleys(x_proj, rect_size.height);
                if(!vall){
                        box[i]=RotatedRect();
                        contours[i].clear();
                }
                else{
                        float avgW = avgWidth(x_proj);
                        float varW= varianceWidth(x_proj, avgW);
                        float avgH = avgHeight(x_proj);
                        float varH= varianceHeight(x_proj, avgH);
                        if(varW>tw || varH>th){
                                box[i]=RotatedRect();
                                contours[i].clear();
                        }
                        else{
                                avgSW += strokeWidthAvg(cropped);
                                sumAvgH += avgH;
                                count++;
                        }
                }
        }
}
```

נסיר חלקים ששטחם קטן:

```
for( int i = 0; i < (int)contours.size(); i++ ){
        if((contSize>0) && !contours[i].empty() && (box[i].size.width*box[i].size.height<600)){
                if(contSize>1){
                        contours[i].clear();
                        box[i]=RotatedRect();
                }
        }
```

ניצור את כל ה-TextLines ונשמור בוקטור tl שקיבלנו כארגומנט:

```
        for( int i = 0; i < (int)contours.size(); i++ ){
                if(!contours[i].empty()){
                        TextLine* t= new TextLine();
                        t->setImage(_image);
                        t->setRect(box[i]);
                        tl.push_back(t);
                }
        }
```

ציור החלקים (contours) כדי שנוכל להציג אותם. (עבור דיבאגינג).

```
        RNG rng(12345);
        /// Draw contours
        Mat drawing = Mat::zeros( dst.size(), CV_8UC3 );
        for( int i = 0; i< (int)contours.size(); i++ )
        {
                Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
                drawContours( drawing, contours, i, color, 2, 8, hierarchy, 0, Point() );
                circle( drawing, mc[i], 4, color, -1, 8, 0 );
```

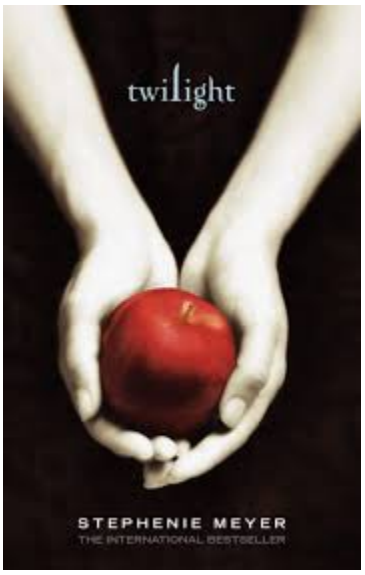
נצייר את המלבנים שקיבלנו העוטפים את הטקסט על התמונה המקורית ונציג את התוצאה:

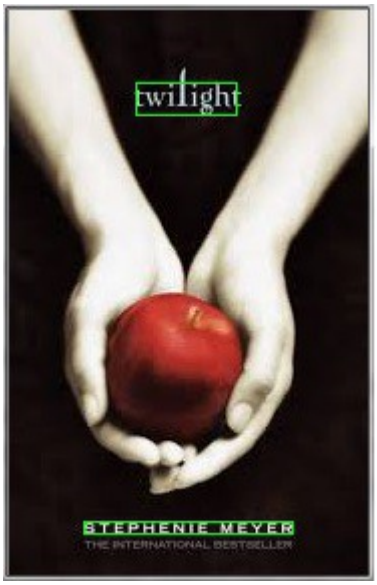
```
                // Draw the bounding box
                Point2f vtx[4];
                box[i].points(vtx);
                for( int k = 0; k < 4; k++ )
                        line(img2, vtx[k], vtx[(k+1)%4], Scalar(0, 255, 0), 1, 1);
        }

        /// Show in a window
        imshow( "dst", img2);
```

דוגמאות לתוצאות:

דוגמאות שעבדו:

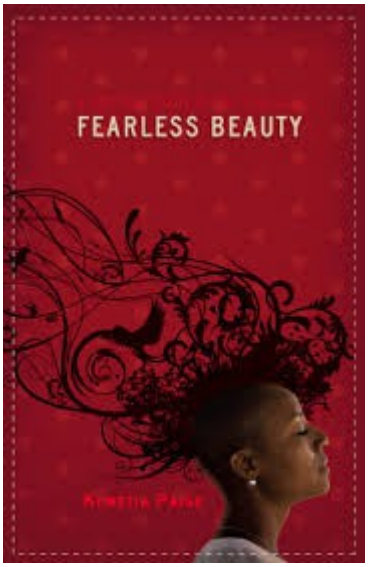
.1

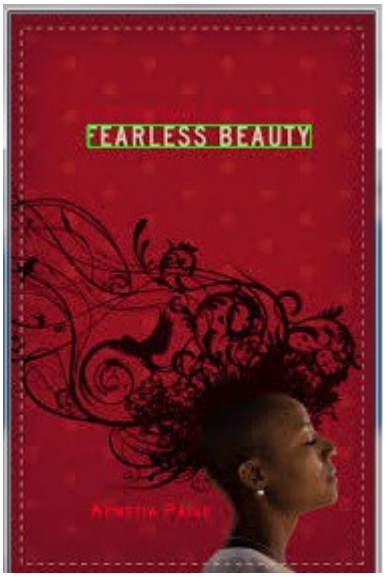





```
The average line height is: 11
```

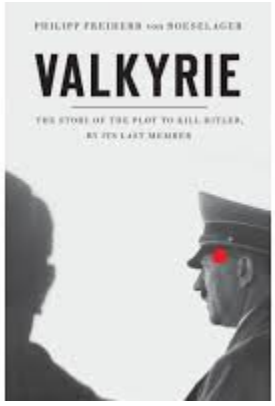
.2

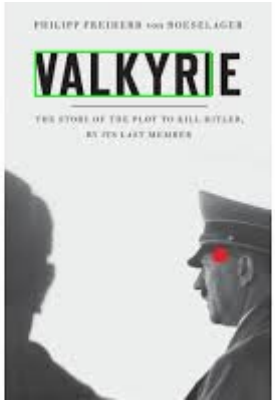





```
The average line height is: 11
```

.3






```
The average line height is: 22
```

<u>הערה:</u> כאשר ישנו כיתוב חלש או בצבע שדומה לצבע הרקע, החדות אינה גבוהה מספיק עבור אלגוריתם זה.

<u>דוגמאות נוספות:</u>

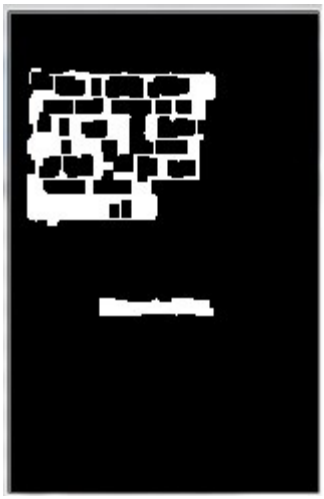
.1


```
The average line height is: 69.3333
```

במקרה זה, ניתן לראות כי ישנו מלבן מיותר ולכן החישוב אינו נכון. ניתן לתקן זאת ע"י הגדלת הערך של גודל המלבן המקסימלי אותו נרצה להסיר (מה שמתאים יותר לתמונות גדולות כמו התמונה הנ"ל).
לאחר התיקון נקבל:


```
The average line height is: 92.5
```

.2



```
The average line height is: 42.5
```

גם במקרה זה האלגוריתם לא עבד מכיוון שהפעולות המורפולוגיות שבתחילת האלגוריתם (blur, open, close) גרמו לשורות הטקסט שלמעלה להתמזג (ניתן לראות בתמונה הבינארית), ולכן זה חושב כ-contour יחיד.