**[OLD] Exposure correction**

* Find average color of the page using threshold to select. Once average is found make a layer of it and desaturate it using luminance mode.
* Make LLL layers and whatnot.
* Find out which ones you need by finding the first one that doesn’t change an average color pixel.
* Use Gaussian blur (25) on all the ones you need then alpha levels (64)

**[OLD] Color correction**

For images with a color reference target

1. Make a copy of the layer you want to correct and desaturate it on average mode.
2. Set the grayscale layer to difference mode.
3. Within the blackest, whitest, and middle grayest boxes of the color reference target(s), select the most neutral pixels that have RGB values within 5-251. When the grayscale layer is in difference mode, the most neutral pixels are indicated by the lowest RGB values.
4. Using the color balance feature with preserve luminosity ON, adjust the highlights, midtones, and shadows of the original layer so that the selected pixels are completely neutral.

**FINAL PROCESS**

**Step 1: Determining if a source is relevant/usable**

To be considered for use, a source must present new information that directly relates to proboscideans, their position within a system (e.g., their evolutionary relationships to other living things, how they are viewed in human societies, etc.), or the position of something within a  proboscidean system (e.g. the chemical composition and interactions of elephant hormones, proboscidean diets, etc.)

**Step 2: Determining if a source was created digitally or created physically then digitized**

A document was most likely created digitally if…

* The publication date is 1995 or later
* The text of the document displays anti-aliasing
* The document was created/produced using: Acrobat Distiller, Adobe Pagemaker, Adobe InDesign, QuarkXPress Passport
* For scientific papers, the paper was published before its issue or volume

If a source was created digitally, no editing is required.

The document was most likely created physically then digitized if…

* The publication date is earlier than 1995
* The text of the document does not display anti-aliasing
* The document was created/produced using: Adobe Paper capture plug-in, Acrobat Image Conversion Plug-in
* For scientific papers, the paper was published later than its issue or volume

**Step 3: Algorithm for Correcting Scanned Documents (AfCSD)**

**Some things to keep in mind**

* The program should:
  + Be repeatable with the same results
  + Make different scans of the same documents look nearly identical
* I think shadows are the 25% lowest RGB values, highlights are the 25% highest RGB values, and midtones are the 50% in between
* GIMP calculates the luminance desaturation with the following formula
  + (0.22 × R) + (0.72 × G) + (0.06 × B)
* According to FADGI:
  + All RGB values should be within 5-251
  + Color temperature of light used should be 5000K with a CRI of 90
  + Backing material should not be used as part of tonal calibration
  + Do not use the CYMK color mode as it has less color gamut
  + Negatives may have a tonal range of 0-255

**1. Determining scanned image quality**

* Below, the red highlighted factors are ones that you probably won’t be able to program in. Yellow ones are uncertain, greens are good to go.
  + Completeness: 10 points
  + Edition: 9 points
  + Crop margins: 8 points
    - If the image has been cropped at all beforehand then 0 points are awarded, if the image has not been cropped then 8 points
  + Resolution: 7 points
  + Image file type:
    - TIFF: 6
    - JPG2000: 4.8 points
    - PNG: 3.6 points
    - JPG: 2.4 points
    - GIF: 1.2 points
    - BMP: 0 points
  + Presence of a color reference target: 5 points
  + Tampering (annotations, etc.): 4 points
  + Needed deskewing and perspective correction: 3 points
    - Find the difference between the RGB values of the uncorrected and the corrected image
  + Needed exposure correction: 2 points
  + Moire: 1 point
* Scan quality = Total points ÷ 10

**2. Color and contrast correction:**

**Might need to make contrast and color different things because you’re probably gonna wanna keep contrast high for easier edge detection**

Useful links

* Automatic white balancing with grayworld assumption: <https://stackoverflow.com/questions/46390779/automatic-white-balancing-with-grayworld-assumption>
* Automatic color correction based on Xrite Passport color card: <https://gist.github.com/bhive01/492101637c0c81f4bbc889901857a377#file-play3-py>

Program skeleton

* Scan through the images of the pages of a document
  + If the image is from Internet Archive
    - If there is a color chart
      * Decrease the contrast so the lowest RGB value is 30 and the highest is 240, and the rest of the RGB values are stretched accordingly
        + Find the whitest, most middling gray, and blackest squares in the color chart

Within these squares, take the 50% of the pixels that are closest to absolute white (RBG 255, 255, 255), absolute gray (127.5, 127.5, 127.5), and absolute black (0, 0, 0).

Adjust the RGB values of these pixels to ensure the RGB values are equal within each pixel, making them “neutral” colors

* + - * Apply the color adjustment made to the blackest pixels to the shadows of all the images, the color adjustment made to the grayest pixels to the midtones of all the images, and the color adjustments made to the whitest pixels to the highlights of all the images
        + If there are more neutral colors on a color reference target than just one shade of black, one shade of gray, and one shade of white, then divide up the shadows, midtones, and highlights accordingly
    - If there’s no color chart
      * Decrease the contrast to that the lowest RGB value is 12% higher and the highest RGB value is 6% lower, and the rest of the RGB values are stretched accordingly
      * Find one or more blank pages or pages with considerable whitespace
      * Average the RGB colors of the whitespace (aka without the words or pictures for documents that have them) together and invert them
      * Match the brightness of this inverted color to the brightness of the original average color of the page
      * Put a layer of this inverted color over all the pages and overlay it slightly so the hue of the page caused by tinted lighting (not by the page yellowing) is cancelled out
        + Maye just do it on a page by page basis (ie do this entire process but instead of averaging the colors of all the pages together, just average each page for itself)
      * (Maybe test this out by trying it out on the processed version of a scan from Internet Archive that also has an unprocessed version)
  + If the image is not from Internet archive
    - If there is a color chart
      * Find the whitest, most middling gray, and blackest squares in the color chart
        + Within these squares, take the 50% of the pixels that are closest to absolute white (RBG 255, 255, 255), absolute gray (127.5, 127.5, 127.5), and absolute black (0, 0, 0).

Adjust the RGB values of these pixels to ensure the RGB values are equal within each pixel, making them “neutral” colors

* + - * Apply the color adjustment made to the blackest pixels to the shadows of all the images, the color adjustment made to the grayest pixels to the midtones of all the images, and the color adjustments made to the whitest pixels to the highlights of all the images
        + If there are more neutral colors on a color reference target than just one shade of black, one shade of gray, and one shade of white, then divide up the shadows, midtones, and highlights accordingly
    - If there’s no color chart
      * Reduce the contrast so that all RGB values are between 5 and 251 if that is not already true.
      * Find one or more blank pages or pages with considerable whitespace
      * Average the RGB colors of the whitespace (aka without the words or pictures for documents that have them) together and invert them
      * Match the brightness of this inverted color to the brightness of the original average color of the page
      * Put a layer of this inverted color over all the pages and overlay it slightly so the hue of the page caused by tinted lighting (not by the page yellowing) is cancelled out
        + Maye just do it on a page by page basis (ie do this entire process but instead of averaging the colors of all the pages together, just average each page for itself)
      * (Maybe test this out by trying it out on the processed version of a scan from Internet Archive that also has an unprocessed version)

**Making sure the page is evenly lit:**

Useful links

* GIMP method for evening exposure and dewarping pages: <https://news.ycombinator.com/item?id=12312408>

Program skeleton

* Take a document (page, picture, etc)
* Find a few differently lit areas where the color is supposed to be more or less the same (this may require user input, or you need to make the program display RGB values of a given pixel like GIMP does)
* Take the RGB values of these areas, making sure the ratio of R to G to B are more or less the same
* Take the average RGB value of these areas to get the lightness
* Change all the RGB values of the selected areas so they average to the lightness RGB value
* All the pixels closest to the selected areas should be modified by the same amount, and in turn the pixels closest to them should be modified the same amount.

**Making sure the page is well lit:**

Useful links

Program skeleton

* Take the average RGB color of the page’s whitespace
* Make it grayscale and make the average RGB color middle gray (127.5, 127.5, 127.5)
* Add the color back

**Chromatic aberration correction:**

Useful links

* Estimation of chromatic aberration in Python: <https://stackoverflow.com/questions/9414204/chromatic-aberration-estimation-in-python>

**Moire correction:**

Useful links

* Removing moire in photographs using MATLAB: <https://stackoverflow.com/questions/20531110/removing-pattern-and-noise-in-an-image-using-fft-in-matlab>
* Mathematical method for removal of moire: <https://arxiv.org/ftp/arxiv/papers/1701/1701.09037.pdf>

Program skeleton

* Something where the program calculates the difference in RGB values between adjacent pixels, and if the pixels are hugely different it blurs the outlier pixel so its the average color of its surroundings.

**Deskewing:**

Useful links

* Sinc interpolation in MATLAB: <http://phaseportrait.blogspot.com/2008/06/sinc-interpolation-in-matlab.html>
* Perfect sinc interpolation in MATLAB and Python: <https://gist.github.com/endolith/1297227>
* Deskewing pages based on text in Python: <https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/>
* Automatic deskewing based on lines in Python: <https://stackoverflow.com/questions/46731947/detect-angle-and-rotate-an-image-in-python>

Program skeleton

* Scan through the images of a document
* Detect where the document is
* If the document is in color or in grayscale
  + Measure the angles of all ends of the document (or any that aren’t obscured by the crop margins, for Internet Archive posts the glass may be a helpful point of reference)
  + Identify the largest angle
  + Scale up the image size by 10x without interpolation
  + Rotate by the inverse of the largest angle using whatever interpolation method turns out to be best
    - B spline
    - Shearing method
  + Scale down the image size so that it’s 2x its original size, and increase the image resolution by 2x so it’s the same physical size
* If the document is just absolute black and absolute white
  + Find all the pages that have straight lines (maybe like text, pictures, or actual straight lines)
  + Identify the largest angle
  + Scale up the image size by 10x without interpolation
  + Rotate by the inverse of the largest angle using whatever interpolation method turns out to be best
    - B spline
    - Shearing method
  + Scale down the image size so that it’s 2x its original size, and increase the image resolution by 2x so it’s the same physical size

**Perspective warp:**

**Might not have to do this separately since the unbending method seems to take care of that**

Useful links

* ImageMagick program to correct perspective in scans: <https://github.com/ad-si/Perspec>
* ImageMagick program to correct perspective in scans: <https://github.com/ad-si/Perspec>

Program skeleton

**Warping to unbend pages:**

Useful links

* Paper for automatic detection of page borders: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.1467>
* Automatic detection of graph paper borders question in Python: <https://stackoverflow.com/questions/44649652/image-processing-border-detection-of-an-object-on-quite-the-same-background>
* Unbending pages based on text in Python: <https://mzucker.github.io/2016/08/15/page-dewarping.html>
  + Code for it: <https://github.com/mzucker/page_dewarp/blob/master/page_dewarp.py>
  + This method is the best method of page unbending: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.7439>
    - Mathematical explanation of said method (this is important because the code in github does not account for horizontal distortion): <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.552.8971>
  + Method used in Leptonica: <http://tpgit.github.io/UnOfficialLeptDocs/leptonica/dewarping.html>

Program skeleton

* Goal(s):
  + Make the page as perfectly rectangular as possible without distorting its contents
* Possible means of accomplish this
  + Lens distortion
  + Shearing

**Cropping:**

Useful links

* Detecting if an image is cropped

Program skeleton

* Detect the edges of a document and crop it accordingly

Goal of the transformations:

* Deskewing: Applies if all the page edges are tilted in the same direction by the same angle
* Perspective warp: Applies if the angles of two non-adjacent edges of the page are opposite, and the other to edges are not of equal length
* Unbending: Applies if any edges of the pages are nonlinear; also applies if any of the document contents that are supposed to be linear (e.g. text, edges of pictures) are nonlinear

**Conversion to a PDF:**

Useful links

Program skeleton

**Optical character recognition:** Ideally in different languages if that’s possible (namely English, Latin, German, French, Japanese, Chinese).

Useful links

* OCR Python code that Owen used that one time: <https://github.com/jbarlow83/OCRmyPDF>

Program skeleton

**Ideas**

* Automatic summaries of papers using natural language processing