Documented Design

My problem requires a solution that lies well within the realms of Artificial Intelligence and Image Processing. As I would like the program to operate as a mobile document scanner, it would be beneficial for me to build my program into an app. To do this, I’ve decided to focus on using Python with OpenCV as the main processing aspect, with this being the backend, mainly because of Pythons ability to perform extremely well with image recognition and mathematical models. My image recognition and text ripping Python programs will be paired with a frontend built with Swift, to allow the program to run locally on an iOS device as an app.

Another advantage of using Python as a backend is that I will only require one version of a program to be updated whenever the program is modified. This also means I could run a version on a server which could then be accessed through a website for desktop users, as well as creating a version that android users could use as a native app.

Hardware Choice for a server isn’t important as of now, however in the circumstances that I was building a host to run a website with my Python backend on, I would use a raspberry pi 3.

Input Data: Photo/Image data (temp.jpg file from app)

Processes: Image data, Transformation Requirements, Spell Checker

Storage: .txt files of text in Image, Reprocessed Image Files, List of common words

Outputs: Image Text Data

The data requirements for the system are quite simple, as I only require an image to be sent to the back end by the user. Therefore, none of the text files will need to be validated by the user from the program. The user will be able to interact with the file after processing has been completed, but for the sake of simplicity the program will not read data from any of the text files it has written.

One of the most important parts of the program is the perspective transform for the document scanner to get a near perfect image of the document that is being captured. The code isn’t incredibly long, but it allows us to move from the image below left to the image below right:

Perspective Transform Module Pseudocode (allows us to covert original image as above):

USING NumPy

SUBROUTINE order\_points(pts):

rect ← ArrayOfZeros((4, 2), datatype←"float32") # From Numpy

s ← pts.sum(axis←1) # From Numpy

rect[0] ← pts[np.argmin(s)]

rect[2] ← pts[np.argmax(s)]

dIF f ← np.dIF f(pts, axis←1 THEN

ENDIF

rect[1] ← pts[np.argmin(dIF f) THEN

ENDIF

rect[3] ← pts[np.argmax(dIF f) THEN

ENDIF

RETURN rect

ENDSUBROUTINE

SUBROUTINE four\_point\_transform(image, pts):

ENDFOR

rect ← order\_points(pts)

(tl, tr, br, bl) ← rect

widthA ← np.sqrt(((br[0] - bl[0]) \*\* 2) + ((br[1] - bl[1]) \*\* 2))

widthB ← np.sqrt(((tr[0] - tl[0]) \*\* 2) + ((tr[1] - tl[1]) \*\* 2))

maxWidth ← max(STRING\_TO\_INT(widthA), STRING\_TO\_INT(widthB))

heightA ← np.sqrt(((tr[0] - br[0]) \*\* 2) + ((tr[1] - br[1]) \*\* 2))

heightB ← np.sqrt(((tl[0] - bl[0]) \*\* 2) + ((tl[1] - bl[1]) \*\* 2))

maxHeight ← max(STRING\_TO\_INT(heightA), STRING\_TO\_INT(heightB))

dst ← np.array([

[0, 0],

[maxWidth - 1, 0],

[maxWidth - 1, maxHeight - 1],

[0, maxHeight - 1]], dtype←"float32")

M ← cv2.getPerspectiveTransform(rect, dst)

ENDFOR

warped ← cv2.warpPerspective(image, M, (maxWidth, maxHeight))

RETURN warped