Text Extraction from Images, PDF Files, and Speech Data

Penelope Prochnow

I. Introduction

The aim of this project is to create an algorithm that surpasses the current accuracy rate for the most widely adapted and refined image, PDF, and speech data text extraction implementations. Through this adaptation there were elements devised and mutated with alternative closely relating algorithms to create a product with increased accuracy. This report will highlight the trial and error of various methods conducted, which ultimately lead to the final algorithms. All files sampled in this report and recorded as results are included in the submitted directory for replication of results.

II. PDF Interpreter

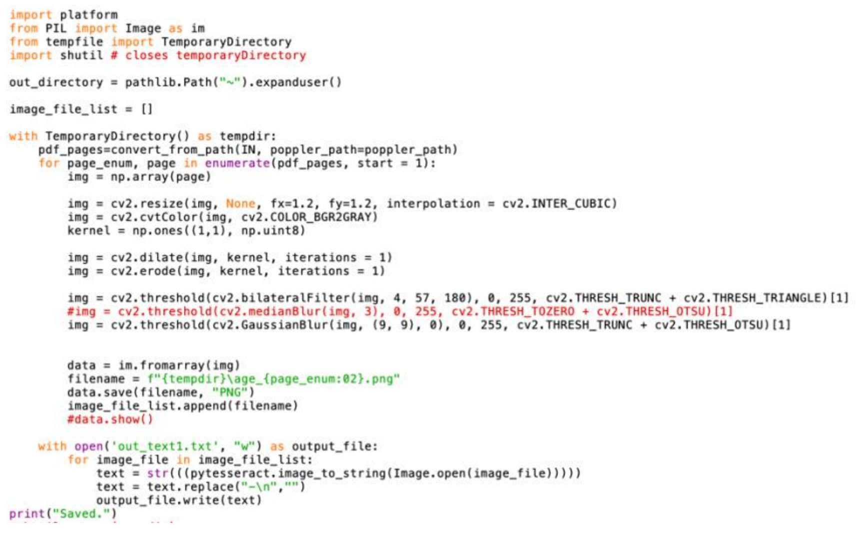
A. *Image to Text*

After installing the pre-established files included in the project description, there were minimal alterations taken to implement the image function. The image function scans images that contain text using pytesseract image to string extraction and translates to output text and has a text to speech capability using gTTS, a Google Speech package. The methods contained in this function aided the creation of the subsequent PDF interpretations containing images and the implementations of translating text to speech.

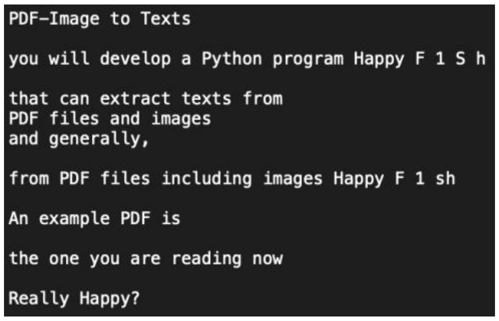
B. *Text PDFs*

As a starting point for creating the PDF Interpreter implementation, there were two pre-established packages that resulted in the highest accuracy and created an elegant .txt file interpretation of plain text pdf files. These two packages were 'fitz' and 'pdfplumber.' While these packages created promising output for pdfs that contained only text, they were altogether inefficient and inadequate when testing with the three-page 'Programming-Project-2.pdf' file. The complications from these packages arose at the images on the third page of this pdf, not only did the text not scrape properly from the images but these packages are incomprehensible for pdfs containing many columns.

Initially, one approach that was taken to overcome this challenge was to locate the images and interpret them separately through an image extraction process. This approach proved to be laborious as the images contained in the third page appeared to be encoded in the document and became corrupt when transferring to JPEG image scanning methods. Due to these setbacks, the implementation of this approach came to a halt and another approach was adapted. While adapting the usage of the 'pdfplumber' package, I came across an approach to using OpenCV as a method of reading text from images. Although using OpenCV in adaptation with 'pdfplumber' was altogether disappointing, the method gave birth to another idea. The ulterior approach was to transform each page of the pdf to an image to utilize OCR capabilities. This idea is widely adapted and used across many interfaces. Through many sessions and days of trial and error, a mediocre algorithm was created. This algorithm would take a pdf file as input and immediately scan each page to create an image using pdf2image. The images were then processed through resizing, transforming to greyscale, thresholding through various gradient cv2 methods, structuring a kernel of each page to dilate and erode images, then each image was saved to a temporary directory. From the temporary directory, images were read back in and converted to text with Pytesseract's image to string function. From there text was created. The text output from the OCRed pages was altogether disappointing. This text could be altered and improved by adjusting the parameters of the various pre-processing techniques. Inspirations for the preprocessing techniques used in this code came from a StackOverflow help page for improving tesseract OCR accuracy for image processing.



```python
import platform
from PIL import Image as im
from tempfile import TemporaryDirectory
import shutil # closes temporaryDirectory

out_directory = pathlib.Path("~").expanduser()

image_file_list = []

with TemporaryDirectory() as tempdir:
    pdf_pages=convert_from_path(IN, poppler_path=poppler_path)
    for page_enum, page in enumerate(pdf_pages, start = 1):
        img = np.array(page)

        img = cv2.resize(img, None, fx=1.2, fy=1.2, interpolation = cv2.INTER_CUBIC)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kernel = np.ones((1,1), np.uint8)

        img = cv2.dilate(img, kernel, iterations = 1)
        img = cv2.erode(img, kernel, iterations = 1)

        img = cv2.threshold(cv2.bilateralFilter(img, 4, 57, 180), 0, 255, cv2.THRESH_TRUNC + cv2.THRESH_TRIANGLE)[1]
        #img = cv2.threshold(cv2.medianBlur(img, 3), 0, 255, cv2.THRESH_TOZERO + cv2.THRESH_OTSU)[1]
        img = cv2.threshold(cv2.GaussianBlur(img, (9, 9), 0), 0, 255, cv2.THRESH_TRUNC + cv2.THRESH_OTSU)[1]


        data = im.fromarray(img)
        filename = f"{tempdir}\age_{page_enum:02}.png"
        data.save(filename, "PNG")
        image_file_list.append(filename)
        #data.show()

    with open('out_text1.txt', "w") as output_file:
        for image_file in image_file_list:
            text = str(((pytesseract.image_to_string(Image.open(image_file)))))
            text = text.replace("-\n","")
            output_file.write(text)
print("Saved.")
```



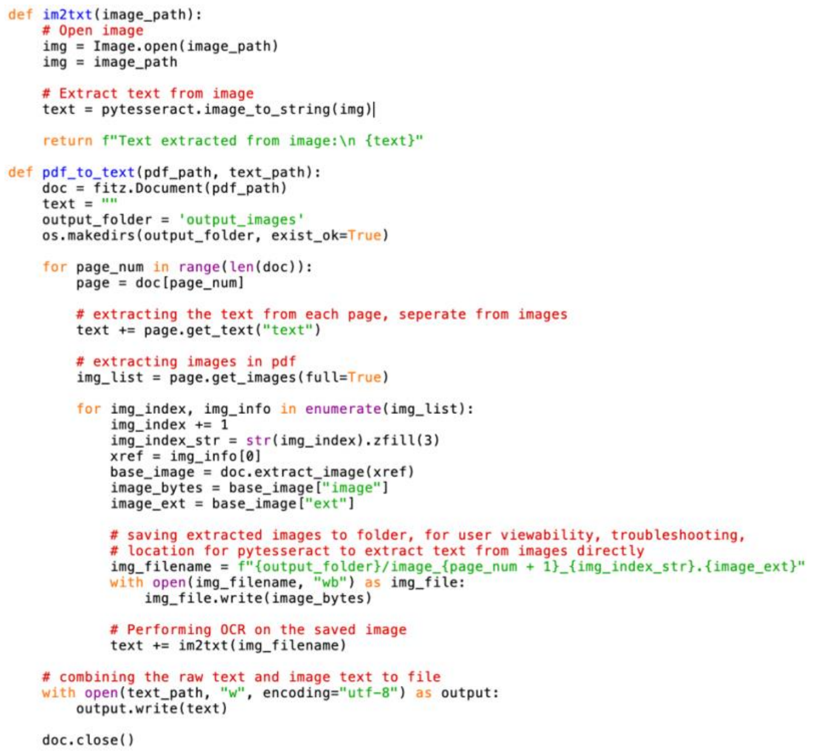
```
PDF-Image to Texts

you will develop a Python program Happy F 1 S h

that can extract texts from
PDF files and images
and generally,

from PDF files including images Happy F 1 sh

An example PDF is

the one you are reading now

Really Happy?
```

Figure 1. OCR implementations to translate PDF file through pdf2image.

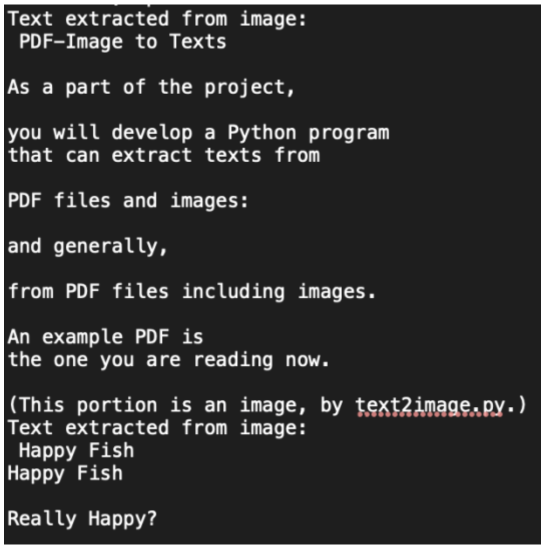
In the end, as shown in Figure 1, even the most fine-tuned pre-processing techniques resulted in a poor text translation. Ultimately, the most complex lines included the second "Happy Fish" and the "Really Happy?" lines. It would seem as one line got closer to being perfected the other line would stray further from a proper translation. This translation is a sound method for most applications and results in an improvement over basic pdf interpretation packages as all other elements of the pdf were properly translated and the text adheres to spacing that properly coordinates with the original pdf file.

C. Adaptations taken for the 'Programming-Project-2.pdf' File

While compiling the three aforementioned methods the results for this use case were not optimal. After reiterating through each of the attempts, troubleshooting the issues of each implementation was segmented into a finer revision. Troubleshooting the implementation of the 'fitz' package method led to an alternative discovery after having worked with and learning the basis of image processing in python through the trial and error shown in Figure 1. Segmenting the code into fractions that worked well, plain text interpretation, and segments that did not work well, interpretation of images, and thoroughly examining routes that could be taken to better each limitation led to a furthered development of image extraction. At which point, a development was made as it was discovered that in the PyMuPDF 'fitz' package there is a function to get images from pdf files. After getting the image information into a list, each image could then be extracted individually, and text could be taken from the images using the pytesseract image to string function as used in the image to text function. This final development is shown in Figure 2.



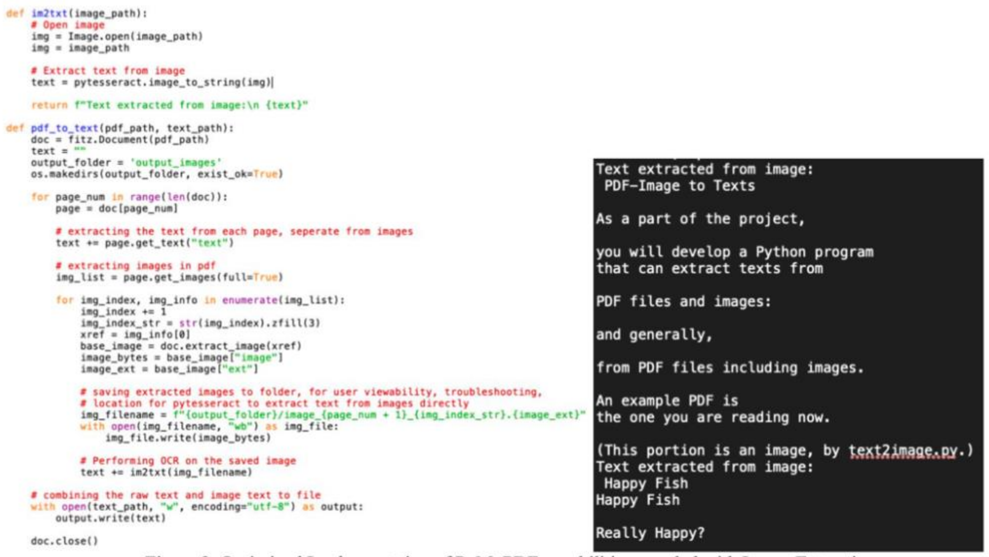
```python
def im2txt(image_path):
    # Open image
    img = Image.open(image_path)
    img = image_path

    # Extract text from image
    text = pytesseract.image_to_string(img)|

    return f"Text extracted from image:\n {text}"

def pdf_to_text(pdf_path, text_path):
    doc = fitz.Document(pdf_path)
    text = ""
    output_folder = 'output_images'
    os.makedirs(output_folder, exist_ok=True)

    for page_num in range(len(doc)):
        page = doc[page_num]

        # extracting the text from each page, seperate from images
        text += page.get_text("text")

        # extracting images in pdf
        img_list = page.get_images(full=True)

        for img_index, img_info in enumerate(img_list):
            img_index += 1
            img_index_str = str(img_index).zfill(3)
            xref = img_info[0]
            base_image = doc.extract_image(xref)
            image_bytes = base_image["image"]
            image_ext = base_image["ext"]

            # saving extracted images to folder, for user viewability, troubleshooting,
            # location for pytesseract to extract text from images directly
            img_filename = f"{output_folder}/image_{page_num + 1}_{img_index_str}.{image_ext}"
            with open(img_filename, "wb") as img_file:
                img_file.write(image_bytes)

            # Performing OCR on the saved image
            text += im2txt(img_filename)

    # combining the raw text and image text to file
    with open(text_path, "w", encoding="utf-8") as output:
        output.write(text)

    doc.close()
```



```
Text extracted from image:
 PDF—Image to Texts

As a part of the project,

you will develop a Python program
that can extract texts from

PDF files and images:

and generally,

from PDF files including images.

An example PDF is
the one you are reading now.

(This portion is an image, by text2image.py.)
Text extracted from image:
 Happy Fish
Happy Fish

Really Happy?
```

Figure 2. Optimized Implementation of PyMuPDF capabilities coupled with Image Extraction

As presented in the Figure 2 code extract, the images are individually saved to a “output_folder” directory, the purpose suffices as a troubleshooting method as well as a user clarity check to assure that each image was properly extracted, and they can view exactly which elements are appearing in the text extraction. Although this code is highly efficient to translate all elements of any pdf, it does not place the text in the corresponding image location from the pdf in the text file. Alternative tests are included below with original pdf images and output text.



```python
def im2txt(image_path):
    # Open image
    img = Image.open(image_path)
    img = image_path

    # Extract text from image
    text = pytesseract.image_to_string(img)|

    return f"Text extracted from image:\n {text}"

def pdf_to_text(pdf_path, text_path):
    doc = fitz.Document(pdf_path)
    text = ""
    output_folder = 'output_images'
    os.makedirs(output_folder, exist_ok=True)

    for page_num in range(len(doc)):
        page = doc[page_num]

        # extracting the text from each page, seperate from images
        text += page.get_text("text")

        # extracting images in pdf
        img_list = page.get_images(full=True)

        for img_index, img_info in enumerate(img_list):
            img_index += 1
            img_index_str = str(img_index).zfill(3)
            xref = img_info[0]
            base_image = doc.extract_image(xref)
            image_bytes = base_image["image"]
            image_ext = base_image["ext"]

            # saving extracted images to folder, for user viewability, troubleshooting,
            # location for pytesseract to extract text from images directly
            img_filename = f"{output_folder}/image_{page_num + 1}_{img_index_str}.{image_ext}"
            with open(img_filename, "wb") as img_file:
                img_file.write(image_bytes)

            # Performing OCR on the saved image
            text += im2txt(img_filename)

    # combining the raw text and image text to file
    with open(text_path, "w", encoding="utf-8") as output:
        output.write(text)

    doc.close()
```

```
Text extracted from image:
 PDF-Image to Texts

As a part of the project,

you will develop a Python program
that can extract texts from

PDF files and images:

and generally,

from PDF files including images.

An example PDF is
the one you are reading now.

(This portion is an image, by text2image.py.)
Text extracted from image:
 Happy Fish
Happy Fish

Really Happy?
```

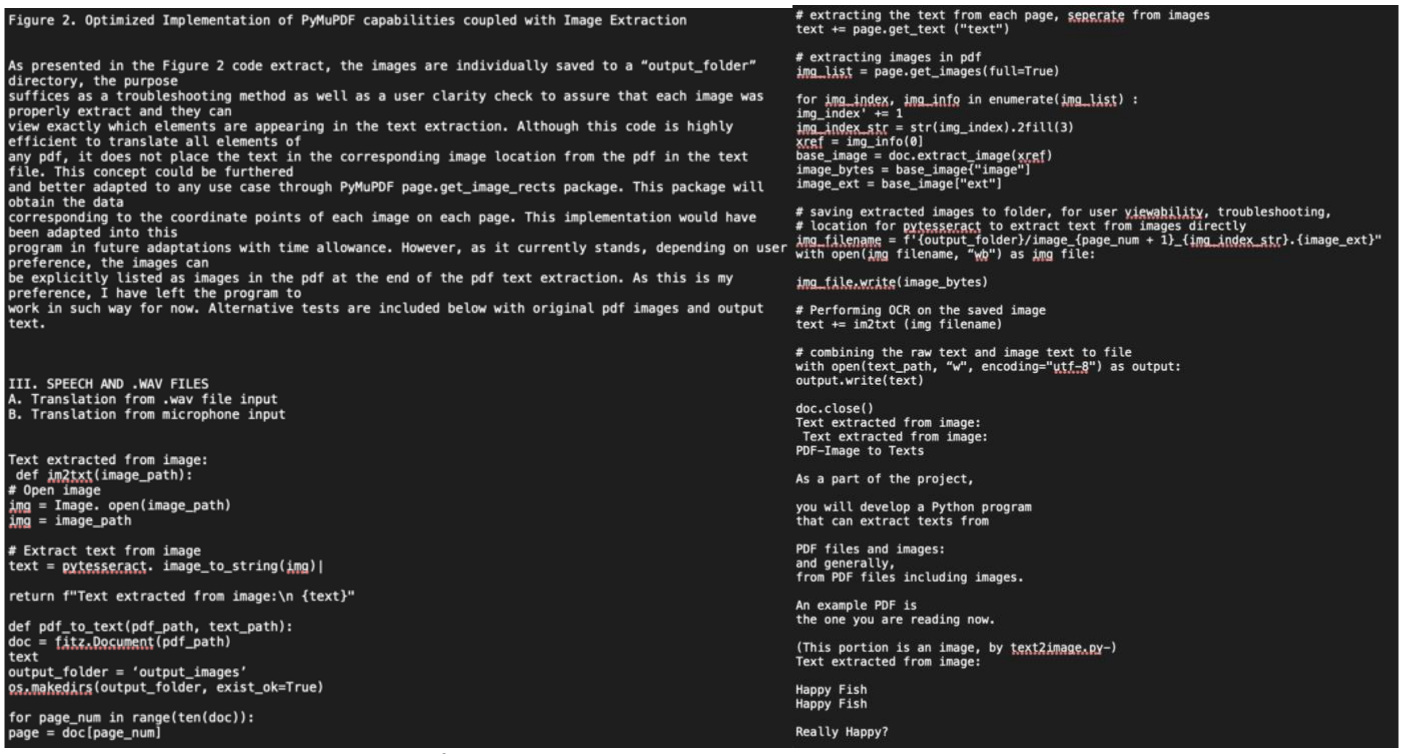
Figure 2. Optimized Implementation of PyMuPDF capabilities coupled with Image Extraction

As presented in the Figure 2 code extract, the images are individually saved to a "output_folder" directory, the purpose suffices as a troubleshooting method as well as a user clarity check to assure that each image was properly extract and they can view exactly which elements are appearing in the text extraction. Although this code is highly efficient to translate all elements of any pdf, it does not place the text in the corresponding image location from the pdf in the text file. This concept could be furthered and better adapted to any use case through PyMuPDF page.get_image_rects package. This package will obtain the data corresponding to the coordinate points of each image on each page. This implementation would have been adapted into this program in future adaptations with time allowance. However, as it currently stands, depending on user preference, the images can be explicitly listed as images in the pdf at the end of the pdf text extraction. As this is my preference, I have left the program to work in such way for now. Alternative tests are included below with original pdf images and output text.

III. Speech and .Wav Files

A. Translation from .wav file input

B. Translation from microphone input

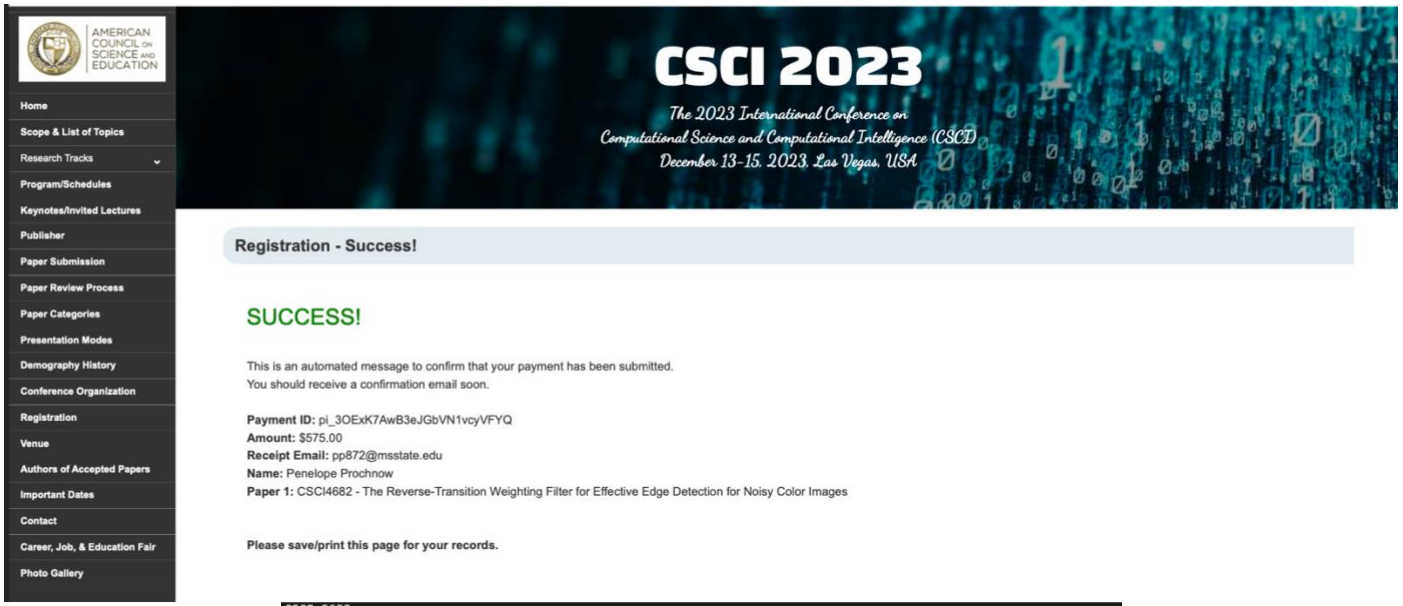


```
Figure 2. Optimized Implementation of PyMuPDF capabilities coupled with Image Extraction


As presented in the Figure 2 code extract, the images are individually saved to a "output_folder"
directory, the purpose
suffices as a troubleshooting method as well as a user clarity check to assure that each image was
properly extract and they can
view exactly which elements are appearing in the text extraction. Although this code is highly
efficient to translate all elements of
any pdf, it does not place the text in the corresponding image location from the pdf in the text
file. This concept could be furthered
and better adapted to any use case through PyMuPDF page.get_image_rects package. This package will
obtain the data
corresponding to the coordinate points of each image on each page. This implementation would have
been adapted into this
program in future adaptations with time allowance. However, as it currently stands, depending on user
preference, the images can
be explicitly listed as images in the pdf at the end of the pdf text extraction. As this is my
preference, I have left the program to
work in such way for now. Alternative tests are included below with original pdf images and output
text.



III. SPEECH AND .WAV FILES
A. Translation from .wav file input
B. Translation from microphone input


Text extracted from image:
 def im2txt(image_path):
# Open image
img = Image. open(image_path)
img = image_path

# Extract text from image
text = pytesseract. image_to_string(img)|

return f"Text extracted from image:\n {text}"

def pdf_to_text(pdf_path, text_path):
doc = fitz.Document(pdf_path)
text
output_folder = 'output_images'
os.makedirs(output_folder, exist_ok=True)

for page_num in range(ten(doc)):
page = doc[page_num]
```

```
# extracting the text from each page, seperate from images
text += page.get_text ("text")

# extracting images in pdf
img_list = page.get_images(full=True)

for img_index, img_info in enumerate(img_list) :
img_index' += 1
img_index_str = str(img_index).2fill(3)
xref = img_info(0]
base_image = doc.extract_image(xref)
image_bytes = base_image{"image"]
image_ext = base_image["ext"]

# saving extracted images to folder, for user viewability, troubleshooting,
# location for pytesseract to extract text from images directly
img_filename = f'{output_folder}/image_{page_num + 1}_{img_index_str}.{image_ext}"
with open(img filename, "wb") as img file:

img_file.write(image_bytes)

# Performing OCR on the saved image
text += im2txt (img filename)

# combining the raw text and image text to file
with open(text_path, "w", encoding="utf-8") as output:
output.write(text)

doc.close()
Text extracted from image:
 Text extracted from image:
PDF-Image to Texts

As a part of the project,

you will develop a Python program
that can extract texts from

PDF files and images:
and generally,
from PDF files including images.

An example PDF is
the one you are reading now.

(This portion is an image, by text2image.py-)
Text extracted from image:

Happy Fish
Happy Fish

Really Happy?
```

Figure 3. Interpretation of Two Images and Text, images contain text at various heights in various colors.

Figure 3 contains the text extraction of the third page from this report, these images were taken before the completion of the report. The top image is the pdf excerpt that is translated in the two subsequent text extractions. As shown here, the handling of multiple images containing an assortment of different colored text at different heights on different colored background images is interpreted well due to the interpretation of each image as a separate segment. Through this practice, the readability of the extraction is optimized as the lines are separated clearly and do not blend across the interface which would create a mutated translation.



CSCI 2023

The 2023 International Conference on Computational Science and Computational Intelligence (CSCI)
December 13-15, 2023, Las Vegas, USA

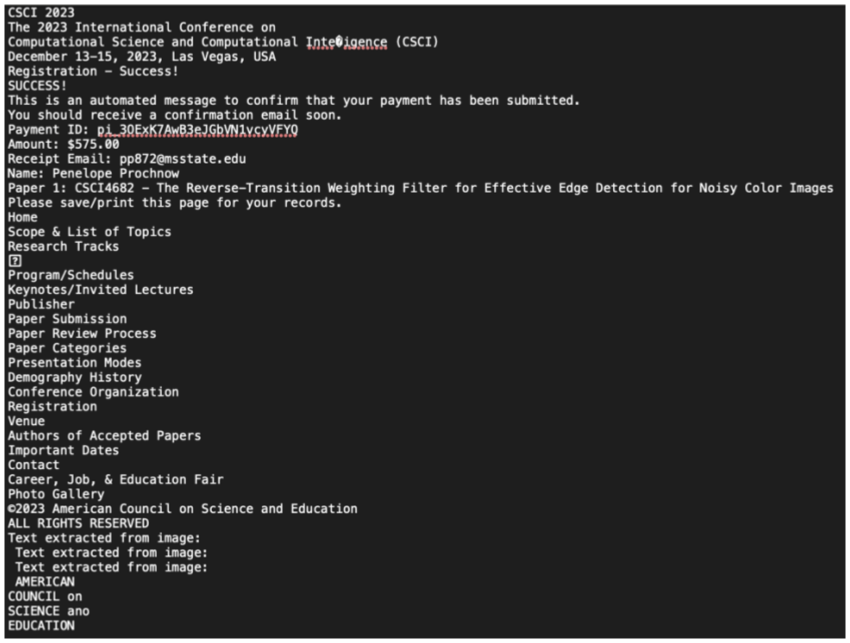
Registration - Success!

SUCCESS!

This is an automated message to confirm that your payment has been submitted.
You should receive a confirmation email soon.

Payment ID: pi_3OExK7AwB3eJGbVN1vcyVFYQ
Amount: $575.00
Receipt Email: pp872@msstate.edu
Name: Penelope Prochnow
Paper 1: CSCI4682 - The Reverse-Transition Weighting Filter for Effective Edge Detection for Noisy Color Images

Please save/print this page for your records.



```
CSCI 2023
The 2023 International Conference on
Computational Science and Computational Inte�igence (CSCI)
December 13-15, 2023, Las Vegas, USA
Registration - Success!
SUCCESS!
This is an automated message to confirm that your payment has been submitted.
You should receive a confirmation email soon.
Payment ID: pi_3OExK7AwB3eJGbVN1vcyVFYQ
Amount: $575.00
Receipt Email: pp872@msstate.edu
Name: Penelope Prochnow
Paper 1: CSCI4682 - The Reverse-Transition Weighting Filter for Effective Edge Detection for Noisy Color Images
Please save/print this page for your records.
Home
Scope & List of Topics
Research Tracks
⍰
Program/Schedules
Keynotes/Invited Lectures
Publisher
Paper Submission
Paper Review Process
Paper Categories
Presentation Modes
Demography History
Conference Organization
Registration
Venue
Authors of Accepted Papers
Important Dates
Contact
Career, Job, & Education Fair
Photo Gallery
©2023 American Council on Science and Education
ALL RIGHTS RESERVED
Text extracted from image:
 Text extracted from image:
 Text extracted from image:
 AMERICAN
COUNCIL on
SCIENCE ano
EDUCATION
```

Figure 4. Interpretation of A PDF with complex image presentation.

The Figure 4 extraction is the interpretation of a PDF that does not contain plain text, this pdf was created from the downloading of a webpage resulting in complex text interpretation. This pdf also contains images that do not contain text. The images extracted are shown in Figure 5 for clarity of empty "Text extracted from image:" lines. Additionally, the usage of testing the program with this pdf goes to show that complex cursive fonts can be miss interpreted as symbols at a low error rate, as this confusion only occurred for the two 'L's in 'Intelligence' that are connected with this font. There also appears to be an error interpreting the downward arrow for the 'Research Tracks' line.

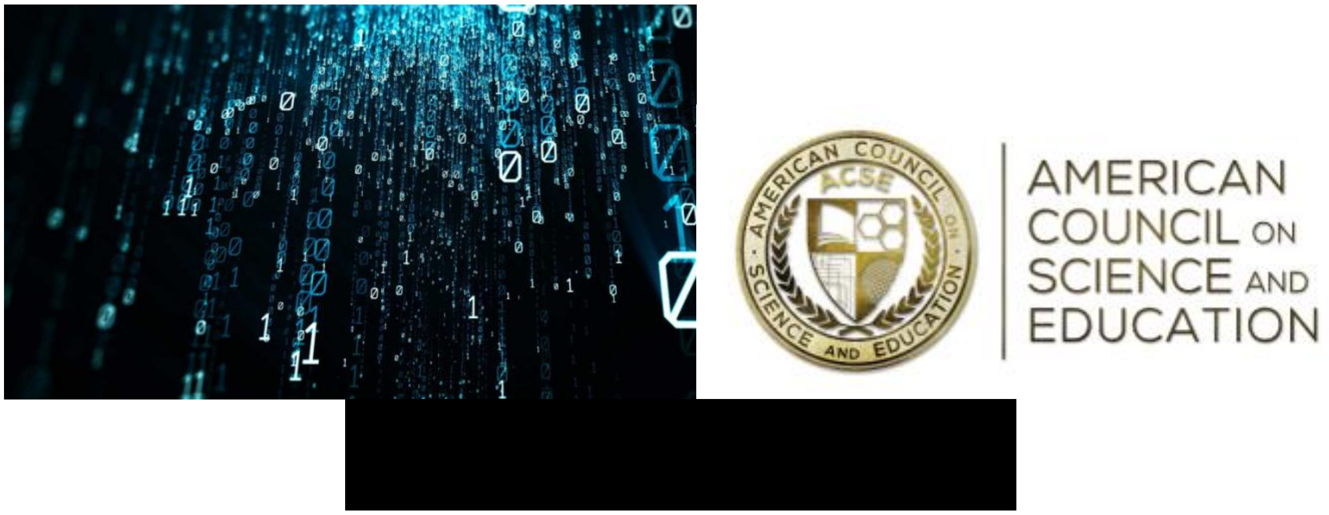
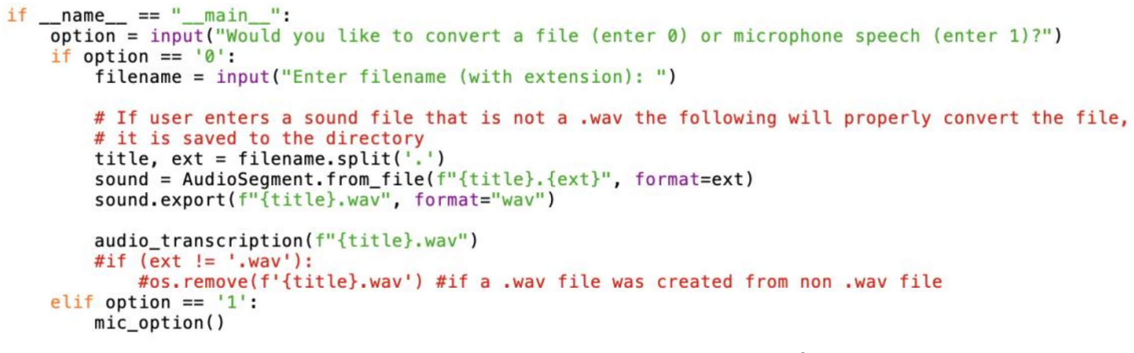

AMERICAN COUNCIL ON SCIENCE AND EDUCATION
ACSE
AMERICAN COUNCIL ON SCIENCE AND EDUCATION

Figure 5. Images Extracted from the Figure 4 pdf, from the output folder.

Viewing the images that were extracted from the pdf goes to show that through this method, all images are extracted even images that do not contain textual elements. Through a conditioning factor these images could be removed from the final output.

III. Speech and .Wav Files

Similar to the pmode functionality of the pdf interpreter, this speech interpreter initializes and routes itself based on user input to whether they would like to enter a sound file or create sound data through their microphone. The coding function is adapted to translate any sound file to a .wav file for interpretation. The main compiler basis that is used to call all functions is included in Figure 6 for understanding of the calling basis for each of the subsequent functions discussed.



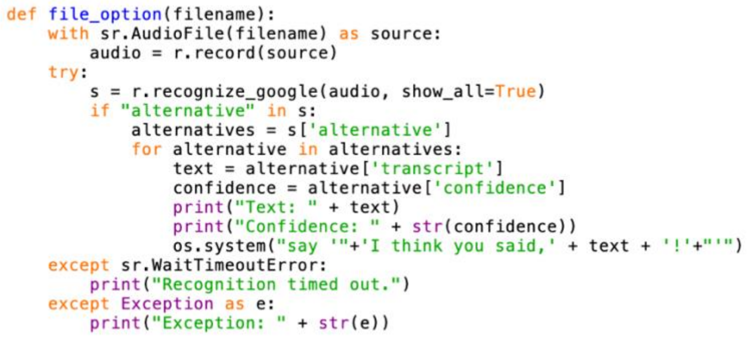
```
if __name__ == "__main__":
    option = input("Would you like to convert a file (enter 0) or microphone speech (enter 1)?")
    if option == '0':
        filename = input("Enter filename (with extension): ")

        # If user enters a sound file that is not a .wav the following will properly convert the file,
        # it is saved to the directory
        title, ext = filename.split('.')
        sound = AudioSegment.from_file(f"{title}.{ext}", format=ext)
        sound.export(f"{title}.wav", format="wav")

        audio_transcription(f"{title}.wav")
        #if (ext != '.wav'):
            #os.remove(f'{title}.wav') #if a .wav file was created from non .wav file
    elif option == '1':
        mic_option()
```

Figure 6. The main compiler showing user decision basis and file conversion.

For user preference, there is a condition commented out that will remove the translated .wav file from the original sound data if the user wishes to maintain a compact program that will eradicate the created file and clean their directory.

A. *Translation from .wav file input*

To glean an understanding of sound data capabilities in Python, the sound file interpretation was curated first. For initial testing of the recognize_google speech recognition package, a function was created to output all possible alternatives of interpretations of whichever sound file that is loaded. Through this exercise it was determinable that the standalone recognize_google package does not optimize the interpretation of speech files. An excerpt of this coding function is included below.

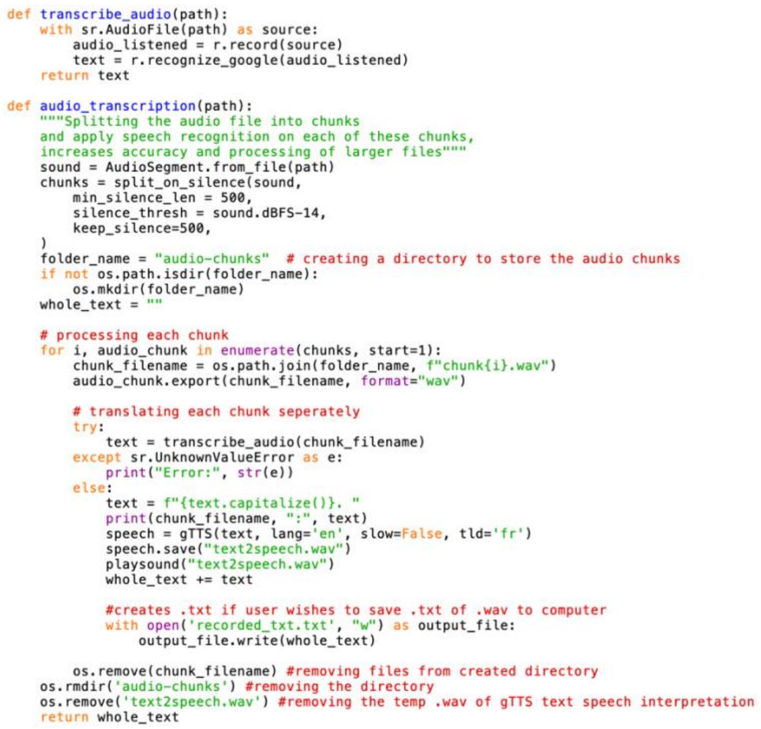


```python
def file_option(filename):
    with sr.AudioFile(filename) as source:
        audio = r.record(source)
    try:
        s = r.recognize_google(audio, show_all=True)
        if "alternative" in s:
            alternatives = s['alternative']
            for alternative in alternatives:
                text = alternative['transcript']
                confidence = alternative['confidence']
                print("Text: " + text)
                print("Confidence: " + str(confidence))
                os.system("say '"+'I think you said,' + text + '!'+"'")
    except sr.WaitTimeoutError:
        print("Recognition timed out.")
    except Exception as e:
        print("Exception: " + str(e))
```

Figure 7. Initial configuration of sound file to text and interpreted to speech.

This initial attempt led to many groundbreaking findings and an increased understanding of sound data capabilities with Python. Out of this attempt four devices became clear, the simplicity of reading in sound files to become audio data, the recognize_google package, the operating system speech translation, and the virtue of try/except functions. When entering a variety of different sound files, it became clear that an alternative approach would have to be taken to translate larger sound files and increase confidence of output.

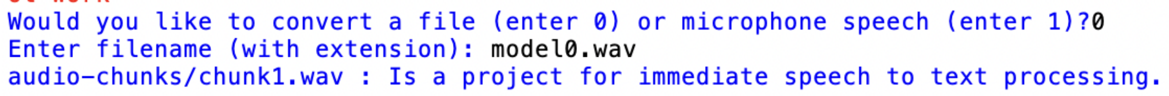
The adaptation that came from this attempt was similar in the importing of the sound file and reading of the data. Alternative speech recognition packages were considered however rejected due their complex initialization processes. Ulterior packages would either require an API key (Microsoft Bing Voice Recognition and IBM Watson Speech to Text), operated on cloud-based systems (CMU Sphinx), or required a subscription plan (Houndify). While a similar approach was taken, the leading issue addressed was the interpretation of larger sound files. To accommodate the interpretation of these larger files an approach taken was divide the sound file into smaller files that would split on large spaces of silence that could be interpreted as the end of a sentence. While testing this method, I came to realize that the interpretation of the files had an increased accuracy.



```python
def transcribe_audio(path):
    with sr.AudioFile(path) as source:
        audio_listened = r.record(source)
        text = r.recognize_google(audio_listened)
    return text

def audio_transcription(path):
    """Splitting the audio file into chunks
    and apply speech recognition on each of these chunks,
    increases accuracy and processing of larger files"""
    sound = AudioSegment.from_file(path)
    chunks = split_on_silence(sound,
        min_silence_len = 500,
        silence_thresh = sound.dBFS-14,
        keep_silence=500,
    )
    folder_name = "audio-chunks"  # creating a directory to store the audio chunks
    if not os.path.isdir(folder_name):
        os.mkdir(folder_name)
    whole_text = ""

    # processing each chunk
    for i, audio_chunk in enumerate(chunks, start=1):
        chunk_filename = os.path.join(folder_name, f"chunk{i}.wav")
        audio_chunk.export(chunk_filename, format="wav")

        # translating each chunk seperately
        try:
            text = transcribe_audio(chunk_filename)
        except sr.UnknownValueError as e:
            print("Error:", str(e))
        else:
            text = f"{text.capitalize()}. "
            print(chunk_filename, ":", text)
            speech = gTTS(text, lang='en', slow=False, tld='fr')
            speech.save("text2speech.wav")
            playsound("text2speech.wav")
            whole_text += text

            #creates .txt if user wishes to save .txt of .wav to computer
            with open('recorded_txt.txt', "w") as output_file:
                output_file.write(whole_text)

        os.remove(chunk_filename) #removing files from created directory
    os.rmdir('audio-chunks') #removing the directory
    os.remove('text2speech.wav') #removing the temp .wav of gTTS text speech interpretation
    return whole_text
```

Figure 8. Final Sound File Interpreter

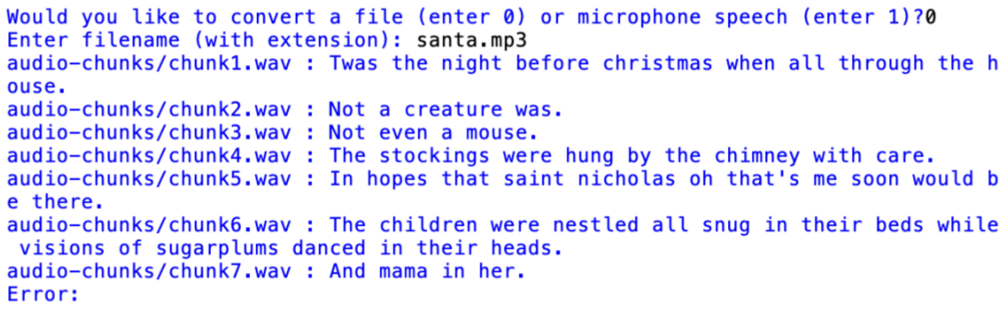
The coding excerpt above includes the exacted function that was used for the interpretation of sound files. This function works through an incrementing process of taking each step in a clear cohesive fashion, which does not result in the most concise implementation but provides elegant output. The audio is read in, split into respective chunks based on silence and decibel frequencies, then saved to separate files in an automated directory, read back in to be interpreted, and spoken back to the user through the Google Text to Speech interpreter. In the final lines of this code the created files are removed. During the development process troubleshooting the program occurred through checking each of the files for the expected contents before removing them. Through this implementation, the final 'text2speech.wav' file removal will throw an error if the data recorded to translate into speech is too incoherent to be interpreted. A conditional statement could be used to eradicate this issue in impeding the removal of the file if it does not exist. However, in this use case the error provides context to which elements need to be adjusted from input.



```
Would you like to convert a file (enter 0) or microphone speech (enter 1)?0
Enter filename (with extension): model0.wav
audio-chunks/chunk1.wav : Is a project for immediate speech to text processing.
```

Figure 9. Output from model0.wav file

The output shown in Figure 9 is the detected speech from the model0.wav file that was included in the original files that were downloaded from the project directory. This output segment shows the lines for user input to designate their choice in uploading a file or microphone speech, showing the choice made and then the input call for the user to enter their desired sound file.



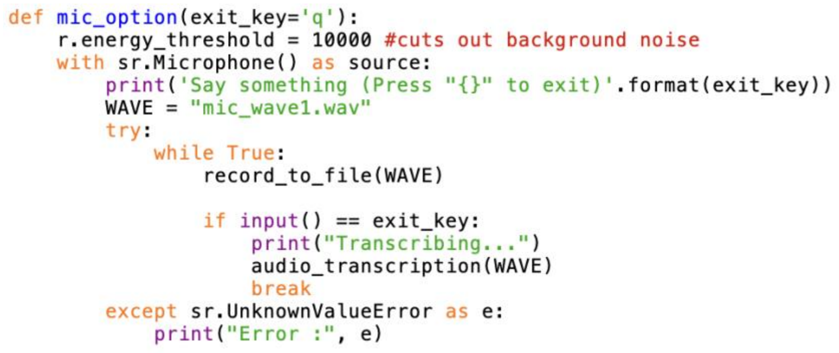
```
Would you like to convert a file (enter 0) or microphone speech (enter 1)?0
Enter filename (with extension): santa.mp3
audio-chunks/chunk1.wav : Twas the night before christmas when all through the h
ouse.
audio-chunks/chunk2.wav : Not a creature was.
audio-chunks/chunk3.wav : Not even a mouse.
audio-chunks/chunk4.wav : The stockings were hung by the chimney with care.
audio-chunks/chunk5.wav : In hopes that saint nicholas oh that's me soon would b
e there.
audio-chunks/chunk6.wav : The children were nestled all snug in their beds while
 visions of sugarplums danced in their heads.
audio-chunks/chunk7.wav : And mama in her.
Error:
```

Figure 10. Output using santa.mp3 sound, file contained in submitted directory.

The above excerpt from the sound file contains flaws as the sound file contains muffled sounds. Additionally, the final "chunk" returns an error as the sound file cuts off without completing the final sentence. The .mp3 file is translated accurately for all distinct words and segments at pauses in speech to distinguish sentences.

B. *Translation from microphone input*

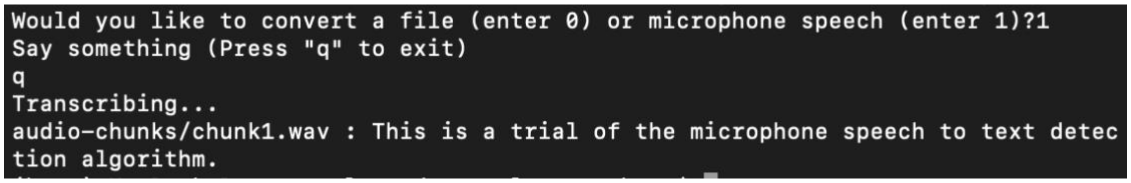
Once the initial program for translating sound files to text output and then into machine speech of the outputted text was completed, the microphone input function was developed. Using the elements from the speech to text function, the only difference between these two functions would be the input option. To reduce redundancy and increase compiling times, the audio_transcription function (shown in Figure 8) was implemented into the microphone coding segment yet was missing the most vital piece of obtaining coherent microphone input. While perusing the files provided in the downloaded project directory the 'speech_mic2wave.py' script proved to provide usable elements that would work apprehensively with the created audio_transcription function while resolving the missing microphone recording element. Through the coupled functions the algorithm shown below was created.



```
def mic_option(exit_key='q'):
    r.energy_threshold = 10000 #cuts out background noise
    with sr.Microphone() as source:
        print('Say something (Press "{}" to exit)'.format(exit_key))
        WAVE = "mic_wave1.wav"
        try:
            while True:
                record_to_file(WAVE)

                if input() == exit_key:
                    print("Transcribing...")
                    audio_transcription(WAVE)
                    break
        except sr.UnknownValueError as e:
            print("Error :", e)
```

Figure 11. Microphone Sound Data Interpreter

The implementation of the algorithm in Figure 11 takes programming beyond simple coding ideology and transforms programming into a sense of thinking. As shown in the above code, using the pre-established functions creates a cohesive and concise algorithm to take user microphone input until the augmented exit key is pressed. At which point the data recorded and processed will be sent to the audio_transcription function which will process the sound file adhering with the aforementioned methods.



```
Would you like to convert a file (enter 0) or microphone speech (enter 1)?1
Say something (Press "q" to exit)
q
Transcribing...
audio-chunks/chunk1.wav : This is a trial of the microphone speech to text detec
tion algorithm.
```

Figure 12. Output from mic_option

The output above in Figure 12 comes from running the code and selecting 1 as user input to record speech through the microphone until the exit key 'q' is pressed or 30 seconds of silence has elapsed as delineated in the record_to_file function. The microphone input is then transferred to the audio_transcription function where the speech is translated to text then spoken back to the user with Google Text to Speech capabilities.

IV. FUTURE WORK

The finalized pdf interpreter concept could be furthered and better adapted to any use case through PyMuPDF page.get_image_rects package. This package will obtain the data corresponding to the coordinate points of each image on each page. This implementation will be adapted into this program in future adaptations with time allowance. However, as it currently stands, the images can be explicitly listed as images in the pdf at the end of the page in which the images were displayed. As the current implementation is my preference, I have left the program to work in such way for now. Additionally, the pdf interpreter would benefit from a conditional statement to avoid outputting any images that are interpreted yet do not contain text. In future applications, the microphone to speech algorithm would benefit from an adaptation with more diverse testing and different background noises or various trials with different ethnic users testing and adapting the algorithm. Through all algorithms introduced and mutated in this project, there is vast room for improvement from fine tuning to additional advancements in clarity of interpretations.