**PDF Text Processing: Manual Step-by-Step Example Run**

**Input Data**

**Textract JSON Data (for one text element):**

{

"BlockType": "LINE",

"Text": "Invoice",

"Page": 1,

"Geometry": {

"BoundingBox": {

"Width": 0.18,

"Height": 0.03,

"Left": 0.4,

"Top": 0.15

},

"Polygon": [

{"X": 0.40, "Y": 0.15},

{"X": 0.58, "Y": 0.15},

{"X": 0.58, "Y": 0.18},

{"X": 0.40, "Y": 0.18}

]

}

}

**Step 1: Main Function Startup**

def fill\_textract\_boxes(input\_pdf\_path, textract\_json\_path, output\_pdf\_path):

doc = fitz.open(input\_pdf\_path) # Opens the PDF

with open(textract\_json\_path, "r") as f:

textract\_data = json.load(f) # Loads the JSON with OCR data

**Step 2: Organizing Data by Page**

# Organize words by page

page\_words = {}

for block in textract\_data['Blocks']:

if block['BlockType'] == 'LINE' and 'Page' in block:

page\_num = block['Page'] - 1 # Convert to 0-based indexing

page\_words.setdefault(page\_num, []).append(block)

For our example:

* page\_num = 1 - 1 = 0 (first page)
* The block is added to page\_words[0]

**Step 3: Processing Page 0**

# Get the page and its dimensions

page = doc[0]

page\_width = page.rect.width # 612 points

page\_height = page.rect.height # 792 points

# Sort words by reading order (top-to-bottom, left-to-right)

words.sort(key=lambda x: (

x['Geometry']['BoundingBox']['Top'],

x['Geometry']['BoundingBox']['Left']

))

Our example word "Invoice" has Top: 0.15 and Left: 0.4, so it will be positioned in the reading order.

**Step 4: Processing the "Invoice" Word**

We extract the polygon from the JSON:

word = {

"Text": "Invoice",

"Geometry": {

"Polygon": [

{"X": 0.40, "Y": 0.15},

{"X": 0.58, "Y": 0.15},

{"X": 0.58, "Y": 0.18},

{"X": 0.40, "Y": 0.18}

]

}

}

polygon = word['Geometry']['Polygon']

**Step 5: Call get\_polygon\_dimensions**

Let's manually execute this function:

def get\_polygon\_dimensions(polygon, page\_width, page\_height):

points = [(p['X'] \* page\_width, p['Y'] \* page\_height) for p in polygon]

x\_coords = [p[0] for p in points]

y\_coords = [p[1] for p in points]

width = max(x\_coords) - min(x\_coords)

height = max(y\_coords) - min(y\_coords)

return width, height

With our values:

points = [

(0.40 \* 612, 0.15 \* 792),

(0.58 \* 612, 0.15 \* 792),

(0.58 \* 612, 0.18 \* 792),

(0.40 \* 612, 0.18 \* 792)

]

= [(244.8, 118.8), (354.96, 118.8), (354.96, 142.56), (244.8, 142.56)]

x\_coords = [244.8, 354.96, 354.96, 244.8]

y\_coords = [118.8, 118.8, 142.56, 142.56]

width = max(244.8, 354.96, 354.96, 244.8) - min(244.8, 354.96, 354.96, 244.8)

width = 354.96 - 244.8 = 110.16 points

height = max(118.8, 118.8, 142.56, 142.56) - min(118.8, 118.8, 142.56, 142.56)

height = 142.56 - 118.8 = 23.76 points

Return: (110.16, 23.76)

So our text box dimensions are: width = 110.16 points, height = 23.76 points.

**Step 6: Call find\_max\_fontsize**

Let's trace through this function with our values:

def find\_max\_fontsize(page, text, box\_width, box\_height, min\_size=1, max\_size=100):

left = min\_size # 1

right = max\_size # 100

optimal\_size = min\_size # 1

while left <= right:

mid = (left + right) // 2

text\_width, text\_height = get\_text\_dimensions(page, text, mid)

if text\_width <= box\_width \* 1 and text\_height <= box\_height \* 1:

optimal\_size = mid

left = mid + 1

else:

right = mid - 1

prob\_presence = False

for ch in ['g', 'j', 'p', 'q', 'y']:

if ch in text:

prob\_presence = True

break

if prob\_presence == False:

optimal\_size \*= 1.175

return optimal\_size

Tracing through the binary search:

**Iteration 1:**

* left = 1, right = 100
* mid = (1 + 100) // 2 = 50
* Let's assume get\_text\_dimensions(page, "Invoice", 50) returns (170, 50)
* 170 > 110.16, so text doesn't fit
* right = 50 - 1 = 49

**Iteration 2:**

* left = 1, right = 49
* mid = (1 + 49) // 2 = 25
* Let's assume get\_text\_dimensions(page, "Invoice", 25) returns (85, 25)
* 85 < 110.16 and 25 > 23.76, so text doesn't fit height-wise
* right = 25 - 1 = 24

**Iteration 3:**

* left = 1, right = 24
* mid = (1 + 24) // 2 = 12
* Let's assume get\_text\_dimensions(page, "Invoice", 12) returns (40.8, 12)
* 40.8 < 110.16 and 12 < 23.76, so text fits!
* optimal\_size = 12
* left = 12 + 1 = 13

**Iterations continue...**

**Final iteration:**

* Let's say we end up with optimal\_size = 22

Now check for descending characters:

* "Invoice" has no 'g', 'j', 'p', 'q', or 'y'
* prob\_presence = False
* optimal\_size = 22 \* 1.175 = 25.85
* Round to 26

Return: 26 (font size)

**Step 7: Calculate Text Position**

# Calculate text position (center of the box)

points = [

(244.8, 118.8), (354.96, 118.8),

(354.96, 142.56), (244.8, 142.56)

]

center\_x = (244.8 + 354.96 + 354.96 + 244.8) / 4 = 1199.52 / 4 = 299.88

center\_y = (118.8 + 118.8 + 142.56 + 142.56) / 4 = 522.72 / 4 = 130.68

# Get text dimensions at size 26

text\_width, text\_height = get\_text\_dimensions(page, "Invoice", 26)

# Let's assume this returns (88.4, 26)

# Calculate final position

text\_x = center\_x - text\_width / 2 = 299.88 - 88.4/2 = 299.88 - 44.2 = 255.68

text\_y = center\_y + text\_height / 4 = 130.68 + 26/4 = 130.68 + 6.5 = 137.18

# Check for descender characters

prob\_presence = False # No descenders in "Invoice"

# Additional adjustment for text without descenders

text\_y = text\_y + text\_height / 8 = 137.18 + 26/8 = 137.18 + 3.25 = 140.43

**Step 8: Insert the Text**

page.insert\_text(

point=(255.68, 140.43),

text="Invoice",

fontsize=26,

fontname="helv",

fill\_opacity=0,

color=(0, 0, 1) # Blue color

)

This places "Invoice" on the page:

* At position (255.68, 140.43)
* With font size 26 points
* Using Helvetica font
* With transparent background
* In blue color

**Summary of Steps and Calculations**

1. **Extract polygon coordinates from Textract data**
   * Normalized coordinates → [(0.40, 0.15), (0.58, 0.15), (0.58, 0.18), (0.40, 0.18)]
2. **Convert to absolute PDF coordinates**
   * Multiply by page dimensions → [(244.8, 118.8), (354.96, 118.8), (354.96, 142.56), (244.8, 142.56)]
3. **Calculate box dimensions**
   * Width = 354.96 - 244.8 = 110.16 points
   * Height = 142.56 - 118.8 = 23.76 points
4. **Find maximum font size through binary search**
   * Binary search between 1 and 100
   * Multiple iterations comparing text dimensions vs box dimensions
   * Determined optimal size = 22
   * Adjustment for no descenders: 22 \* 1.175 = 26
5. **Calculate text position**
   * Center point: (299.88, 130.68)
   * Text dimensions at size 26: (88.4, 26)
   * Horizontal centering: 299.88 - 88.4/2 = 255.68
   * Vertical positioning: 130.68 + 26/4 = 137.18
   * Additional adjustment: 137.18 + 26/8 = 140.43
6. **Insert text at calculated position**
   * Position: (255.68, 140.43)
   * Font size: 26 points
   * Result: Perfectly centered "Invoice" text that maximally fills its bounding box

This manual trace demonstrates exactly how the code processes one text element, from the initial JSON data to the final placement in the PDF document.