Downloads to pi

```
pip install opencv-python matplotlib scikit-image numpy
```

Wednesday 3/26 --- Andrew Scerbo

In this code I was able to test clipart so that the image file is read and then convert it to general coordinates. I used skeletonize so the thickness of the lines can be accounted for and also shapes that are inside other shapes. These coordinates will translate to our systems coordinates.

```
import cv2
import numpy as np
from skimage.morphology import skeletonize
import matplotlib.pyplot as plt

# === 1. Load and preprocess image ===
img = cv2.imread('test.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# === 2. Threshold and invert (black lines become white) ===
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)

# === 3. Skeletonize to thin the lines ===
binary_bool = binary > 0
skeleton = skeletonize(binary_bool).astype(np.uint8) * 255

# === 4. Find all contours, including shapes inside shapes ===
contours, hierarchy = cv2.findContours(skeleton, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

# === 5. Extract and print coordinates ===
all_coords = []
for i, contour in enumerate(contours):
    coords = contour.reshape(-1, 2)
    all_coords.append(coords)

    #print(f"\nContour {i}: {len(coords)} points")
    #print(coords)

# === 6. Plot all contours ===
plt.figure(figsize=(8, 8))
```

```
for coords in all_coords:
    xs, ys = zip(*coords)
    plt.plot(xs, ys, marker='.', linestyle='-', linewidth=0.5)

plt.title("All Contour Coordinates (Including Nested Shapes)")
plt.gca().invert_yaxis()
plt.axis('equal')
plt.grid(True)
plt.show()
```

Monday 3/31 -- Andrew Scerbo

On this day I got my code to use the camera and take a snapshot from the camera and use it to find the coordinates on a piece of paper. There is a consistent view of the cameras output and it is uploaded on the website. You are able to take a snapchat and it will upload the coordinate image of the snapshot if the background is white and the writing is black. Started 3D print of the linear actuator to be used with the servo motor.

```
import cv2
import numpy as np
from flask import Flask, Response, render_template, send_file
from skimage.morphology import skeletonize
import io
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure

app = Flask(__name__)
cap = cv2.VideoCapture(1)  # Use 0 for built-in webcam, 1 for external

# Initialize a dummy frame to avoid None errors
last_frame = np.zeros((480, 640, 3), dtype=np.uint8)

@app.route('/')
def index():
    return render_template('index.html')

def generate_frames():
    global last_frame
    while True:
        success, frame = cap.read()
        if not success:
            continue
        last_frame = frame.copy()

        _, buffer = cv2.imencode('.jpg', frame)
```

```
        frame_bytes = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/snapshot')
def snapshot():
    global last_frame

    # Convert to grayscale and binary
    gray = cv2.cvtColor(last_frame, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
    binary_bool = binary > 0
    skeleton = skeletonize(binary_bool).astype(np.uint8) * 255

    # Find contours
    contours, _ = cv2.findContours(skeleton, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

    # Plot using object-oriented matplotlib
    fig = Figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1)

    for contour in contours:
        coords = contour.reshape(-1, 2)
        if len(coords) > 1:
            xs, ys = zip(*coords)
            ax.plot(xs, ys, marker='.', linestyle='-', linewidth=0.5)

    ax.set_title("Skeleton Contours from Snapshot")
    ax.invert_yaxis()
    ax.axis('equal')
    ax.grid(True)

    buf = io.BytesIO()
    canvas = FigureCanvas(fig)
    canvas.print_png(buf)
    buf.seek(0)

    return send_file(buf, mimetype='image/png', download_name='snapshot.png')
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

This is the html format for the video processing and the button for the capture.

```
<!DOCTYPE html>
<html>
<head>
    <title>Live Feed with Snapshot</title>
    <script>
        function takeSnapshot() {
            const snapshotImg = document.getElementById("snapshotImage");
            const timestamp = new Date().getTime();
            snapshotImg.src = "/snapshot?" + timestamp;
        }
    </script>
</head>
<body>
    <h1>Webcam Live Feed</h1>
    <img src="{{ url_for('video_feed') }}" width="640" height="480" />

    <h2>Take Snapshot</h2>
    <button onclick="takeSnapshot()">Capture Snapshot</button>

    <h2>Processed Snapshot</h2>
    <img id="snapshotImage" src="" width="640" height="480" alt="Snapshot will
appear here after capture" />
</body>
</html>
```