ДОДАТОК А

Апробація теми роботи

ADVANTAGES OF CNC LASER SOLDERING OF ELECTRONIC COMPONENTS

Lykhoshvai I.P., student,
Olga S. Gubaryeva, PhD, Associate Professor
Kharkiv National University of Radio Electronics

CNC laser soldering stands out in modern electronics due to its exceptional precision. Unlike traditional soldering methods that apply heat broadly, CNC laser soldering employs a highly focused laser beam. This pinpoint accuracy brings forth numerous advantages for the soldering process.

At the heart of CNC laser soldering lies a high-powered laser source. This laser generates a concentrated beam of light, typically in the infrared spectrum. The CNC (Computer Numerical Control) system precisely directs and controls the laser beam, ensuring it targets the designated soldering points on the electronic components. The system also controls other parameters like laser power, pulse duration, and scan path. Additional components include a delivery system that directs the laser beam, a focusing optic to concentrate the beam further, and a machine vision system for precise positioning and process monitoring [1].

Depending on the specific application and materials involved, different types of lasers can be employed in CNC laser soldering [1]. Common choices include Nd:YAG (Neodymium-doped Yttrium Aluminum Garnet) lasers , known for their versatility and ability to deliver high peak power. Diode lasers offer another option, particularly suited for continuous wave soldering and materials with high absorption rates [2]. The selection of the optimal laser type depends on factors like wavelength, power output, and pulse characteristics.

The choice of solder material plays a crucial role in the success of the soldering process. In CNC laser soldering, special solders are often used,

formulated for compatibility with laser energy and the specific materials being joined. These solders may have lower melting points compared to traditional solders to minimize heat impact on surrounding components. Additionally, lead-free solder options are becoming increasingly popular due to environmental considerations.

The concentrated laser heat significantly reduces the thermal impact on neighboring components. This is especially critical for fragile electronic components that are vulnerable to damage caused by excessive heat. In contrast, traditional soldering methods like wave soldering may unintentionally subject nearby components to temperatures beyond their tolerance levels, resulting in malfunctions or shortened lifespan [1].

The precise control provided by the controlled laser beam is essential for shaping the solder joint accurately. This is crucial for creating well-defined solder connections, especially in high-density electronic devices where space is limited. By customizing the geometry of the solder joint, both electrical conductivity and mechanical strength are optimized, resulting in a reliable connection.

In the fast-paced world of manufacturing today, speed is a critical factor for success. CNC laser soldering is a highly efficient method that offers numerous advantages over traditional soldering techniques. Unlike manual soldering, which requires extensive setup time and skilled operators, CNC laser soldering is a fully automated process. With pre-programmed parameters, setup time is minimized, ensuring consistent outcomes. This automation leads to quicker production cycles and the ability to easily increase production volume.

The precise laser beam focuses on delivering concentrated heat, which leads to rapid melting of the solder material. This results in quicker soldering times compared to methods such as wave soldering, where the entire board must be heated. The speed advantage is further enhanced by the automated process of CNC laser soldering, removing the need for manual handling and associated delays.

By combining short setup time, fast soldering speeds, and automation, CNC laser soldering significantly boosts production throughput. This results in a higher

production output per unit time, enabling manufacturers to meet tight deadlines and efficiently fulfill larger orders. The increased speed also allows for more flexibility in production scheduling and better responsiveness to market demands.

Defect prevention is a continuous challenge in electronics manufacturing. Flawed solder joints can lead to device malfunctions, production setbacks, and increased expenses. CNC laser soldering emerges as a frontrunner in this battle, offering a notably lower defect rate compared to conventional methods.

Traditional soldering techniques, such as hand soldering or wave soldering, can easily cause component overheating due to the broad application of heat. This overheating can harm delicate electronic components, resulting in immediate failure or reduced lifespan. CNC laser soldering, with its precisely controlled laser beam, minimizes the risk of overheating by targeting only the designated soldering area, protecting surrounding components from thermal stress.

CNC laser soldering operates on a pre-programmed and automated system, eliminating the variability present in manual soldering, where operator skill and technique can impact joint quality. The consistent and repeatable process of CNC laser soldering ensures uniform solder joint formation across all components, reducing the likelihood of defects caused by human error.

The reduced defect rate associated with CNC laser soldering directly contributes to higher product quality and reliability.

The miniaturization and increasing complexity of electronic components present a significant obstacle for conventional soldering techniques. CNC laser soldering emerges as a triumphant solution in this field, offering a unique ability to tackle intricate and demanding soldering tasks.

Numerous modern electronic components are highly susceptible to heat. Conventional soldering methods, which apply heat more broadly, can easily harm these fragile components. In contrast, CNC laser soldering delivers heat with precise control and localization, enabling the safe and dependable soldering of temperature-sensitive components. By focusing the laser beam solely on the

designated solder joint, thermal impact on surrounding areas is minimized, effectively safeguarding delicate components.

Conventional soldering techniques struggle to achieve precise connections on uneven or delicate surfaces due to the challenge of controlling heat distribution. However, CNC laser soldering overcomes this limitation. The focused laser beam allows for pinpoint accuracy, facilitating precise soldering on uneven or delicate component features. This level of control ensures reliable connections on complex geometries and intricate components, which is crucial for high-performance electronics.

The trend of miniaturization in electronics has led to components with intricate geometries and tight spacing. Conventional soldering methods often lack the necessary precision to handle these challenges. On the other hand, CNC laser soldering excels in such scenarios. The narrow laser beam can be precisely directed to create well-defined solder joints, even in confined spaces. This capability makes CNC laser soldering the ideal solution for soldering complex components found in advanced electronic devices.

The world of electronics manufacturing demands a soldering solution that can adapt to diverse needs. CNC laser soldering rises to the occasion, boasting impressive flexibility and versatility that surpasses traditional methods.

Unlike traditional soldering methods with limited adjustability, CNC laser soldering offers a high degree of control. The laser parameters, such as power, wavelength, and pulse duration, can be precisely adjusted to cater to the specific properties of different materials and components. This allows for fine-tuning the soldering process for optimal results with a wide range of electronic components.

CNC laser soldering isn't a one-trick pony. Its inherent flexibility allows it to be adapted to various soldering applications. Whether it's soldering delicate SMD (Surface Mount Devices) on a printed circuit board or creating robust connections for power electronics, CNC laser soldering can be configured to meet the specific requirements. This adaptability makes it a valuable asset for diverse production lines, offering a single solution for a wide range of soldering tasks.

The future of electronics often involves the integration of different materials. CNC laser soldering demonstrates promise in this domain. With its adjustable parameters, the laser beam can be tailored to effectively solder various material combinations. This opens doors for innovative designs and functionalities in future electronic devices. The potential for multi-material soldering positions CNC laser soldering as a future-proof technology adaptable to the ever-evolving landscape of electronics manufacturing.

References

1. Kiessling, R., & Steffen, P. (2015). Laser soldering: Processes, materials, and applications. Springer International Publishing.

2. Mallia, D. B., & Milojevic, D. (2015). Laser applications in microelectronic and photonic packaging. Woodhead Publishing Limited.

3. American Welding Society. American Welding Society (AWS): https://www .aws.org/

4. International Standards Organization. (2009). ISO 21952:2009 Lasers and laser-related equipment - Vocabulary and symbols.

ДОДАТОК Б

Фрагменти коду реалізації програмної частини

Головний програмний модуль, що викликає інші підсистеми:

```
import extractor  # local .py script
import image_processing  # local .py script
from path_planner import generate_path  # local .py script
from image_view import tk_root  # local .py script
from gcoder import generate_g_code  # local .py script

import os

DEFAULT_DOTS_PER_MM = 50

# Rountine to pick gerber files
def pick_gbr_files(output_dir, dpmm) -> str:
    from tkinter import filedialog

    board_file = filedialog.askopenfilename(
        initialdir='.',
        title="Select a Board edge .gbr file",
        filetypes=[("Gerber Files", "*.gbr"), ("All Files", "*.*")]
    )

    if not board_file:
        print("No file selected.")
        return

    solder_file = filedialog.askopenfilename(
        initialdir='.',
```

```
        title="Select a Solder Mask .gbr file",
        filetypes=[("Gerber Files", "*.gbr"), ("All Files", "*.*")]
    )

    if not solder_file:
        print("No file selected.")
        return

    svg_file = extractor.parse2svg(board_edge_file_path=board_file,
solder_paste_file_path=solder_file, dpmm=dpmm, output_dir=output_dir)

    return svg_file

def display_results(image_path, centroids, shape_outlines, paths,
need_annotations=False):
    from PIL import Image, ImageDraw
    """
    Overlap the results (centroids, shape outlines, and paths) on the given image.

    :param image_path: Path to the original image
    :param centroids: List of centroids [(x1, y1), (x2, y2), ...]
    :param shape_outlines: List of shape outlines, each as a list of (x, y) coordinates
    :param paths: List of paths, each as a list of (x, y) points covering the shape area
    :param scale: Scaling factor to enlarge the coordinates for better visualization
    :param need_annotations: Boolean to decide if the visualization should include
annotations
    :return: None (displays the image)
    """
    # Load the image
    img = Image.open(image_path)
```

```
    draw = ImageDraw.Draw(img)

    # Draw each shape, centroid, and path
    for idx, (centroid, outline, path) in enumerate(zip(centroids, shape_outlines,
paths)):
        # Draw the shape outline
        draw.polygon(outline, outline="blue", width=2)

        # Draw the centroid
        draw.ellipse([centroid[0] - 4, centroid[1] - 4, centroid[0] + 4, centroid[1] + 4],
fill="red")

        # Draw the laser path
        if len(path) > 1:
            path_coords = [(path[i][0], path[i][1]) for i in range(len(path))]
            draw.line(path_coords, fill="green", width=2)

        # Annotate the centroid (optional)
        if need_annotations:
            draw.text((centroid[0] + 10, centroid[1]), f"#{idx+1}", fill="red")

    # Show the image with the overlapped results
    img.show()  # This will display the image in the default viewer

if __name__ == '__main__':
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument("-v", "--verbose", help='Verbose images output',
action="store_true", required=False)
```

```
    parser.add_argument("-i", "--image", type=str, help='Input image',
required=False)
    parser.add_argument("-dpmm", "--dots_per_mm", type=int, help='Input image
dots/mm', required=False)
    args = parser.parse_args()

    need_annotations = bool(args.verbose)
    image_path = args.image
    dpmm = args.dots_per_mm
    outdir = ""

    if dpmm is None:
        dpmm = int(DEFAULT_DOTS_PER_MM)

    if image_path is None:
        outdir = "out"
        image_path = pick_gbr_files(outdir, dpmm)

    image_path = str(image_path)

    centroids, shape_outlines = image_processing.process(image_path, dpmm,
annotate=need_annotations)

    laser_w = dpmm // 5
    paths = generate_path(centroids, shape_outlines, laser_width = laser_w, margin
= laser_w / 2, annotate=need_annotations)

    laser_gcode_lines = generate_g_code(paths, dpmm)

    file_path_list = image_path.split(os.sep)
```

```
    dir_path = os.sep.join(file_path_list[:-1])
    output_file = f"{os.path.join(dir_path,
file_path_list[-1].removesuffix(".png"))}.gcode"
    print(output_file)
    with open(output_file, "w") as file:
        file.write(laser_gcode_lines)

    print(f"G-code saved to {output_file}")

    display_results(image_path, centroids, shape_outlines, paths)

    if need_annotations:
        if tk_root is not None:
            tk_root.mainloop()
```

Програмний модуль, що виконує парсинг та рендеринг Gerber файлів у растрові зображення:

```
import os, time
from pygerber.gerberx3.api.v2 import Project, GerberFile, FileTypeEnum,
ImageFormatEnum, PixelFormatEnum

def get_timestamp() -> str:
    # Get the current timestamp (number of seconds since the epoch)
    current_timestamp = time.time()
    # Convert timestamp to a readable format (YYYY-MM-DD HH:MM:SS)
    time_struct = time.localtime(current_timestamp)
    formatted_time = time.strftime("%Y_%m_%d_%H_%M_%S", time_struct)
    return formatted_time
```

```
def parse2svg(board_edge_file_path : str, solder_paste_file_path : str, dpmm : int =
250, output_dir = "out") -> str:
    if not os.path.exists(board_edge_file_path):
        exit(f"Error: Gerber file {board_edge_file_path} not found.")
    if not os.path.exists(solder_paste_file_path):
        exit(f"Error: Gerber file {solder_paste_file_path} not found.")



    out_file = Project( [
        GerberFile.from_file(
            file_path = board_edge_file_path,
            file_type = FileTypeEnum.EDGE,
        ),
        GerberFile.from_file(
            file_path = solder_paste_file_path,
            file_type = FileTypeEnum.PASTE,
        ),
    ] ).parse()

    export_dir = f"{output_dir}_{get_timestamp()}"
    if not os.path.exists(export_dir):
        # Create the directory (and intermediate directories, if needed)
        os.makedirs(export_dir)

    output_file_name = os.path.join(export_dir, "output.png")
    out_file.render_raster(destination=output_file_name, dpmm=dpmm,
image_format = ImageFormatEnum.PNG, pixel_format =
PixelFormatEnum.RGBA)

    return output_file_name
```

```
if __name__ == "__main__":
    test_board_paste_file = "../sample/pcba/gerber/sample-pcb-Edge_Cuts.gbr"
    test_solder_paste_file = "../sample/pcba/gerber/sample-pcb-F_Paste.gbr"

    parse2svg(test_board_paste_file, test_solder_paste_file)
```

Програмний модуль, що обробляє вхідне растрове зображення плати, що містить 2 шари: паяльної пасти та країв плати та повертає точки центрів та точки фігур посадкових місць

```
import numpy as np
from scipy.ndimage import label, binary_closing
from skimage.measure import find_contours
from shapely.geometry import Polygon
from PIL import Image, ImageDraw, ImageFont
from image_view import show_image # local .py file

# Function to annotate shapes on the image
def annotate_shapes(image, centroids, shape_outlines):
    draw = ImageDraw.Draw(image)

    try:
        font = ImageFont.truetype("arial.ttf", size=20)  # Use a font file and specify the size
    except IOError:
        font = ImageFont.load_default()  # Fallback to default font if the specified one isn't found

    # Annotate centroids and outlines
```

```
    for i, (centroid, outline) in enumerate(zip(centroids, shape_outlines)):
        x, y = centroid

        # Draw a blue circle at the centroid
        draw.ellipse((x - 3, y - 3, x + 3, y + 3), fill="blue", outline="blue")

        # Draw the outline of the shape in red
        if len(outline) > 0:
            draw.line(outline + [outline[0]], fill="red", width=2)  # Close the contour
loop

        # Label the shape with its number
        draw.text((x, y + 5), f"{i + 1}", fill="blue", font=font)

    return image

# Function to detect and label connected regions
def detect_shapes(image, edge_offset):
    data = np.array(image)  # Convert image to a NumPy array
    # Get the image dimensions
    height, width = data.shape[:2]

    # Define the region to ignore the edges (apply offset)
    min_x = edge_offset
    max_x = width - edge_offset
    min_y = edge_offset
    max_y = height - edge_offset

    # Create a mask to ignore the edges, i.e., set to False for edge regions
    mask = np.ones((height, width), dtype=bool)  # True everywhere initially
```

```
    mask[:min_y, :] = False  # Top edge
    mask[max_y:, :] = False  # Bottom edge
    mask[:, :min_x] = False  # Left edge
    mask[:, max_x:] = False  # Right edge

    alpha_channel = data[:, :, 3]  # Extract alpha channel (transparency)

    # Create a binary mask (True for non-transparent pixels, False for transparent)
    binary_mask = alpha_channel > 127  # Treat anything with alpha > 127 as part
of a shape

    combined_mask = binary_mask & mask  # Only consider shapes not near the
edges

    # Optionally, apply a morphological closing operation to clean up small holes or
noise
    combined_mask = binary_closing(combined_mask, structure=np.ones((2, 2)))  #
Adjust size of structuring element as needed

    # Label connected components
    labeled_array, num_features = label(combined_mask)

    return labeled_array, num_features

# Function to find the centroid of a labeled region
def get_centroids(labeled_array, num_features):
    centroids = []
    for i in range(1, num_features + 1):  # Skip background (label 0)
        coords = np.argwhere(labeled_array == i)  # Get all pixels of this feature
        centroid = coords.mean(axis=0)  # Calculate the centroid
```

```
        centroids.append((int(centroid[1]), int(centroid[0])))  # (x, y)
    return centroids

# Function to get shape outlines
def get_shape_outlines(labeled_array, num_features):
    """
    Filter out internal contours for each labeled region.
    Internal contours are those that are fully enclosed within another contour.
    """
    filtered_outlines = []
    for i in range(1, num_features + 1):
        shape_mask = (labeled_array == i)
        contours = find_contours(shape_mask, 0.5)

        # Create Shapely Polygons for each contour, converting to integer points
        polygons = [Polygon([((int(x), int(y))) for y, x in contour]) for contour in
contours]

        # Identify external contours (not fully enclosed by another contour)
        external_contours = []
        for j, poly in enumerate(polygons):
            is_internal = False
            for k, other_poly in enumerate(polygons):
                if j != k and other_poly.contains(poly):  # Check if poly is inside
other_poly
                    is_internal = True
                    break
            if not is_internal:
                # Append the contour (converted to integer coordinates)
                external_contours.append([(int(y), int(x)) for x, y in contours[j]])
```

```
        # Append the external contours for this label
        filtered_outlines.extend(external_contours)

    return filtered_outlines

def calculate_distance(p1, p2):
    """Calculate Euclidean distance between two points (centroids)."""
    return np.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)

def find_shortest_path(centroids):
    """Find the shortest path visiting each centroid using a greedy nearest neighbor
approach."""
    visited = [False] * len(centroids)  # To track visited centroids
    path = []  # To store the sequence of visited centroids
    current_pos = centroids[0]  # Start from the first centroid
    path.append(0)  # Store the index of the starting centroid
    visited[0] = True  # Mark the first centroid as visited

    # Iterate through all centroids
    for _ in range(1, len(centroids)):
        closest_centroid = None
        min_distance = float('inf')

        # Find the closest unvisited centroid
        for i, centroid in enumerate(centroids):
            if not visited[i]:
                dist = calculate_distance(current_pos, centroid)
                if dist < min_distance:
                    min_distance = dist
```

```
                closest_centroid = i

        # Move to the closest centroid and mark it as visited
        current_pos = centroids[closest_centroid]
        path.append(closest_centroid)  # Append the index of the closest centroid
        visited[closest_centroid] = True

    return path

def reorder_centroids_and_outlines(centroids, shape_outlines, shortest_path):
    """Reorganize centroids and shape outlines based on the visit order from the
shortest path."""
    ordered_centroids = [centroids[i] for i in shortest_path]
    ordered_outlines = [shape_outlines[i] for i in shortest_path]

    return ordered_centroids, ordered_outlines

# Main code to detect, label, and retrieve shape outlines
def process(image_path, dpmm, annotate = False):
    image = Image.open(image_path).convert("RGBA")

    dots_per_inch = dpmm * 25.4  # 1 inch = 25.4 mm
    image.info['dpi'] = (dots_per_inch, dots_per_inch)

    labeled_array, num_features = detect_shapes(image, dpmm)
    centroids = get_centroids(labeled_array, num_features)
    shape_outlines = get_shape_outlines(labeled_array, num_features)

    # Find the shortest path visiting all centroids
    shortest_path = find_shortest_path(centroids)
```

```
    # Reorganize centroids and shape outlines based on the visit order
    ordered_centroids, ordered_outlines = reorder_centroids_and_outlines(centroids, shape_outlines, shortest_path)

    if annotate:
        annotated_image = annotate_shapes(image.copy(), ordered_centroids, ordered_outlines)
        show_image(annotated_image)

    return ordered_centroids, ordered_outlines
```

Програмний модуль, що створює траєкторію переміщення робочого інструменту між посадковими місцями:

```
from shapely.geometry import Point, Polygon
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon as MPolygon

def visualize_path_and_outline(centroid, outline, path, shape_polygon, shape_index):
    """
    Visualize the centroid, outline, and laser path in a plot.

    :param centroid: The (x, y) coordinates of the centroid
    :param outline: The list of (x, y) coordinates defining the shape's outline
    :param path: The list of (x, y) coordinates for the laser path
    :param shape_polygon: The Shapely Polygon object for the shape
    """
    fig, ax = plt.subplots()
```

```
# Plot the shape outline
shape_patch = MPolygon(outline, edgecolor='blue', facecolor='none', lw=2)
ax.add_patch(shape_patch)

# Plot the centroid
ax.plot(centroid[0], centroid[1], 'ro', label="Centroid", markersize=8)

# Plot the laser path
path_x, path_y = zip(*path)
ax.plot(path_x, path_y, label="Laser Path", color='green', lw=1.5)

# Plot the bounding box for clarity
minx, miny, maxx, maxy = shape_polygon.bounds
ax.plot([minx, maxx], [miny, miny], 'k--', lw=1)  # Bottom edge
ax.plot([minx, maxx], [maxy, maxy], 'k--', lw=1)  # Top edge
ax.plot([minx, minx], [miny, maxy], 'k--', lw=1)  # Left edge
ax.plot([maxx, maxx], [miny, maxy], 'k--', lw=1)  # Right edge

# Set plot limits
ax.set_xlim(minx - 10, maxx + 10)
ax.set_ylim(miny - 10, maxy + 10)

ax.set_title(f"Laser Path and Shape Outline - Shape {shape_index + 1}")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()

plt.show()
```

```
def generate_path(centroids, shape_outlines, laser_width: int, margin: int,
annotate=False):
    """
    Generate laser path by scanning the shape's area, starting from the centroid.
    Avoid paths near the edges of the shape by using a margin.

    :param centroids: List of centroids [(x1, y1), (x2, y2), ...]
    :param shape_outlines: List of shape outlines, each as a list of (x, y) coordinates
    :param laser_width: The width of the laser beam (step size)
    :param margin: The margin distance to avoid generating the path near the
shape's edge
    :param annotate: Boolean to decide if the visualization should include
annotations
    :return: List of paths, each as a list of (x, y) points covering the shape area
    """
    paths = []

    for idx, (centroid, outline) in enumerate(zip(centroids, shape_outlines)):
        # Create a polygon for the shape outline
        shape_polygon = Polygon(outline)
        if not shape_polygon.is_valid:
            shape_polygon = shape_polygon.buffer(0)  # Fix invalid geometries if
needed

        # Check if the centroid is inside the shape
        cx, cy = centroid
        centroid_point = Point(cx, cy)

        if not shape_polygon.contains(centroid_point):
```

```
            # If the centroid is outside, choose a point inside the polygon as the starting
point
            # (e.g., the centroid of the polygon)
            centroid_point = shape_polygon.representative_point()

        # Start path from centroid or adjusted point
        path = [centroid_point.coords[0]]  # Starting point (centroid or adjusted point)

        # Get the bounding box of the polygon
        minx, miny, maxx, maxy = shape_polygon.bounds

        # Apply the margin to the bounding box to avoid paths near the edges
        minx += margin
        miny += margin
        maxx -= margin
        maxy -= margin

        # Ensure the adjusted bounding box is still valid
        if minx >= maxx or miny >= maxy:
            continue  # Skip if the margin makes the bounding box invalid

        # Example of a simple square scan (modify as needed)
        step_x = laser_width
        step_y = laser_width

        for y in range(int(miny), int(maxy), step_y):
            for x in range(int(minx), int(maxx), step_x):
                point = Point(x, y)
                if shape_polygon.contains(point):  # Only add points inside the polygon
                    path.append((x, y))
```

```
        paths.append(path)

        # Visualize if needed
        if annotate:
            visualize_path_and_outline(centroid, outline, path, shape_polygon, idx)

    return paths
```

Програмний модуль, що створює G-code для робочої каретки машини:

```
import numpy as np

def generate_g_code(paths, dots_per_mm, feed_rate=1000, delay_ms=1000):
    """
    Generate GRBL-compatible G-code from given paths.

    Args:
        paths (list of list of tuples): A list of paths, where each path is a list of (x, y) points.
        dots_per_mm (float): Conversion factor from pixels to millimeters.
        feed_rate (int): Feed rate for the G1 moves in mm/min.

    Returns:
        str: The generated G-code as a single string.
    """
    g_code = []

    # Initialize G-code
    g_code.append("G21 ; Set units to millimeters")
    g_code.append("G90 ; Absolute positioning")
```

```
g_code.append("G17 ; XY Plane")
g_code.append("G94 ; units per minute feed rate mode")
g_code.append("M3 S1000 ; Laser power on")

for path in paths:
    real_path = np.array(path) / dots_per_mm  # Scale the path to mm
    if len(real_path) == 0:
        continue  # Skip empty paths

    # Move to the start point without cutting (rapid move)
    start_x, start_y = real_path[0]
    g_code.append(f"G0 X{start_x:.3f} Y{start_y:.3f}")

    # Trace the path with cutting
    for line, (x, y) in enumerate(real_path[1:]):
        if line == 0:
            g_code.append(f"G1 F{feed_rate} ; Activate laser")
            g_code.append(f"G4 P{round(delay_ms/1000, 3)} ; Stall for 1s")
        g_code.append(f"G1 X{x:.3f} Y{y:.3f} F{feed_rate}")

# Finalize G-code
g_code.append("M5 ; Turning off laser")

return "\n".join(g_code)
```

ДОДАТОК В

Демонстраційний графічний матеріал

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮИК. 501600.012 ПЗ	Пояснювальна записка	110 с., ф. А4
		Фрагменти коду	18 с., ф. А4
		реалізації програмної	
		частини	
		Графічні документи	
		Презентація	у форматі
			*.pptx

Змін.	Арк.	№ документа	Підп.	Дата	ГЮИК. 501600.012 ВД			
Розроб.		Лихошвай І.П.			Розроблення системи управління технологічним процесом лазерної пайки з використанням ЧПК	Літ.	Аркуш	Аркушів
Перевір.		Омаров Ш.А.						
Н. контр.		Демська Н.П				ХНУРЕ Кафедра КІТАР		
Затвер.		Невлюдов І.Ш						