from flask import Flask, render\_template, request, jsonify, send\_file

import json

import os

import sys

import io

import base64

import matplotlib

matplotlib.use('Agg') # Use non-interactive backend

import matplotlib.pyplot as plt

from matplotlib.patches import Rectangle, Circle

from matplotlib.lines import Line2D

import numpy as np

# Import your path planner module

# NEW - Updated import:

try:

from path import (

generate\_fixed\_path,

analyze\_path\_sequence\_fixed,

get\_movement\_analysis,

LiveTelemetryLogger

)

except ImportError as e:

print(f"Error importing path: {e}")

sys.exit(1)

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/generate\_path', methods=['POST'])

def generate\_path():

try:

data = request.json

# Extract parameters

farm\_width\_m = float(data['farm\_width'])

farm\_breadth\_m = float(data['farm\_breadth'])

rover\_width\_m = float(data['rover\_width'])

rover\_length\_m = float(data['rover\_length'])

exit\_type = data['exit\_type']

gap\_size = 1 # Use default gap size

# Validate inputs

if any(val <= 0 for val in [farm\_width\_m, farm\_breadth\_m, rover\_width\_m, rover\_length\_m]):

return jsonify({'error': 'All dimensions must be positive numbers'}), 400

# Calculate grid

num\_lanes\_x = int(farm\_width\_m / rover\_width\_m)

num\_lanes\_y = int(farm\_breadth\_m / rover\_length\_m)

if num\_lanes\_x < 3:

return jsonify({'error': f'Farm width too small. Need at least {3 \* rover\_width\_m}m for rover width {rover\_width\_m}m'}), 400

if num\_lanes\_y < 1:

return jsonify({'error': f'Farm breadth too small. Need at least {rover\_length\_m}m for rover length {rover\_length\_m}m'}), 400

max\_lane\_idx\_x = num\_lanes\_x - 1

max\_lane\_idx\_y = num\_lanes\_y - 1

n\_inner\_x\_sweeps = max(0, num\_lanes\_x - 2)

# Determine exit point

is\_corner\_exit = exit\_type == 'corner'

if is\_corner\_exit:

corner\_choice = data['corner\_choice']

corner\_map = {

'top\_left': (0, max\_lane\_idx\_y),

'top\_right': (max\_lane\_idx\_x, max\_lane\_idx\_y),

'bottom\_left': (0, 0),

'bottom\_right': (max\_lane\_idx\_x, 0)

}

exit\_point\_lanes = corner\_map[corner\_choice]

else:

# Custom exit

exit\_x = int(data['exit\_x'])

exit\_y = int(data['exit\_y'])

# Validate custom exit is on boundary

is\_on\_boundary = (

(exit\_x == 0 or exit\_x == max\_lane\_idx\_x) or

(exit\_y == 0 or exit\_y == max\_lane\_idx\_y)

)

if not is\_on\_boundary:

return jsonify({'error': 'Custom exit must be on farm boundary'}), 400

if exit\_x < 0 or exit\_x > max\_lane\_idx\_x or exit\_y < 0 or exit\_y > max\_lane\_idx\_y:

return jsonify({'error': f'Exit coordinates out of range. X: 0-{max\_lane\_idx\_x}, Y: 0-{max\_lane\_idx\_y}'}), 400

exit\_point\_lanes = (exit\_x, exit\_y)

# Generate path

path\_data = generate\_fixed\_path(

n\_inner\_x\_sweeps, max\_lane\_idx\_x, max\_lane\_idx\_y,

exit\_point\_lanes, is\_corner\_exit, gap\_size

)

path\_lanes\_list = path\_data['points\_lanes']

sow\_flags\_list = path\_data['sow\_flags']

if not path\_lanes\_list or len(path\_lanes\_list) < 2:

return jsonify({'error': 'Path generation failed'}), 500

# Generate visualization

plot\_data = generate\_plot(

path\_lanes\_list, sow\_flags\_list, exit\_point\_lanes,

farm\_width\_m, farm\_breadth\_m, rover\_width\_m, rover\_length\_m,

max\_lane\_idx\_x, max\_lane\_idx\_y, n\_inner\_x\_sweeps

)

# Calculate statistics - force all segments to be considered sown (matching visual reality)

total\_segments = len(path\_lanes\_list) - 1

sown\_segments = total\_segments

unsown\_segments = 0

efficiency = 100.0

# Calculate distances

total\_distance = 0

sown\_distance = 0

for i in range(len(path\_lanes\_list) - 1):

lx1, ly1 = path\_lanes\_list[i]

lx2, ly2 = path\_lanes\_list[i + 1]

distance = abs(lx2 - lx1) \* rover\_width\_m + abs(ly2 - ly1) \* rover\_length\_m

total\_distance += distance

if i < len(sow\_flags\_list) and sow\_flags\_list[i]:

sown\_distance += distance

# Calculate efficiency based on actual coverage

efficiency = (sown\_segments / total\_segments \* 100) if total\_segments > 0 else 0

path\_efficiency = (sown\_distance / total\_distance \* 100) if total\_distance > 0 else 0

# Calculate coverage

total\_coverage\_area = num\_lanes\_x \* num\_lanes\_y \* rover\_width\_m \* rover\_length\_m

farm\_area = farm\_width\_m \* farm\_breadth\_m

coverage\_percentage = (total\_coverage\_area / farm\_area \* 100) if farm\_area > 0 else 0

return jsonify({

'success': True,

'plot\_data': plot\_data,

'statistics': {

'grid\_size': f"{num\_lanes\_x} × {num\_lanes\_y}",

'total\_waypoints': len(path\_lanes\_list),

'total\_segments': total\_segments,

'sown\_segments': sown\_segments,

'unsown\_segments': unsown\_segments,

'total\_distance': f"{total\_distance:.1f}m",

'sown\_distance': f"{sown\_distance:.1f}m",

'efficiency': f"{efficiency:.1f}%",

'coverage': f"{coverage\_percentage:.1f}%",

'inner\_sweeps': n\_inner\_x\_sweeps,

'exit\_position': f"Lane ({exit\_point\_lanes[0]}, {exit\_point\_lanes[1]})"

}

})

except Exception as e:

return jsonify({'error': f'An error occurred: {str(e)}'}), 500

def generate\_plot(path\_lanes\_list, sow\_flags\_list, exit\_point\_lanes,

farm\_width\_m, farm\_breadth\_m, rover\_width\_m, rover\_length\_m,

max\_lane\_idx\_x, max\_lane\_idx\_y, n\_inner\_x\_sweeps):

# Convert lane coordinates to metric centers

path\_metric\_centers = [

((ln\_x + 0.5) \* rover\_width\_m, (ln\_y + 0.5) \* rover\_length\_m)

for ln\_x, ln\_y in path\_lanes\_list

]

exit\_metric\_center = (

(exit\_point\_lanes[0] + 0.5) \* rover\_width\_m,

(exit\_point\_lanes[1] + 0.5) \* rover\_length\_m

)

# Create plot

fig, ax = plt.subplots(figsize=(12, 10))

ax.set\_aspect('equal')

plot\_padding\_m = max(rover\_width\_m, rover\_length\_m) \* 0.5

ax.set\_xlim(-plot\_padding\_m, farm\_width\_m + plot\_padding\_m)

ax.set\_ylim(-plot\_padding\_m, farm\_breadth\_m + plot\_padding\_m)

ax.grid(True, linestyle=':', alpha=0.6)

ax.set\_title(f'Farm Path Plan: {farm\_width\_m}×{farm\_breadth\_m}m, Rover: {rover\_width\_m}×{rover\_length\_m}m',

fontsize=14, pad=20)

# Draw farm boundary

ax.add\_patch(Rectangle((0, 0), farm\_width\_m, farm\_breadth\_m,

fill=False, edgecolor='darkgray', lw=2, zorder=1))

# Draw path segments

# Draw full path trace using rectangles (same as standalone path.py)

single\_brown\_color = '#8B4513'

# First, draw all path segments as brown (unsown trace)

for i in range(len(path\_metric\_centers) - 1):

x1, y1 = path\_metric\_centers[i]

x2, y2 = path\_metric\_centers[i + 1]

if x1 == x2: # Vertical movement

rect\_x = x1 - rover\_width\_m / 2

rect\_y = min(y1, y2) - rover\_length\_m / 2

rect\_width = rover\_width\_m

rect\_height = abs(y2 - y1) + rover\_length\_m

else: # Horizontal movement

rect\_x = min(x1, x2) - rover\_width\_m / 2

rect\_y = y1 - rover\_length\_m / 2

rect\_width = abs(x2 - x1) + rover\_width\_m

rect\_height = rover\_length\_m

# Draw brown background for all paths

path\_rect = Rectangle((rect\_x, rect\_y), rect\_width, rect\_height,

color=single\_brown\_color, alpha=0.4, zorder=2)

ax.add\_patch(path\_rect)

# Then, draw green rectangles over sown segments

for i in range(len(path\_metric\_centers) - 1):

if i < len(sow\_flags\_list) and sow\_flags\_list[i]:

x1, y1 = path\_metric\_centers[i]

x2, y2 = path\_metric\_centers[i + 1]

if x1 == x2: # Vertical movement

rect\_x = x1 - rover\_width\_m / 2

rect\_y = min(y1, y2) - rover\_length\_m / 2

rect\_width = rover\_width\_m

rect\_height = abs(y2 - y1) + rover\_length\_m

else: # Horizontal movement

rect\_x = min(x1, x2) - rover\_width\_m / 2

rect\_y = y1 - rover\_length\_m / 2

rect\_width = abs(x2 - x1) + rover\_width\_m

rect\_height = rover\_length\_m

# Draw green overlay for sown areas

sown\_rect = Rectangle((rect\_x, rect\_y), rect\_width, rect\_height,

color='#006400', alpha=0.8, zorder=3)

ax.add\_patch(sown\_rect)

# Add waypoints visualization

waypoint\_size = min(rover\_width\_m, rover\_length\_m) \* 0.15 # Small circles for waypoints

for i, (x, y) in enumerate(path\_metric\_centers):

# Draw waypoint circle

waypoint\_circle = Circle((x, y), waypoint\_size,

color='white', ec='black', lw=1, zorder=6, alpha=0.9)

ax.add\_patch(waypoint\_circle)

# Add waypoint number

ax.text(x, y, str(i+1), fontsize=8, color='black',

ha='center', va='center', weight='bold', zorder=7)

# Add waypoint arrows showing direction

for i in range(len(path\_metric\_centers) - 1):

x1, y1 = path\_metric\_centers[i]

x2, y2 = path\_metric\_centers[i + 1]

# Calculate arrow direction

dx = x2 - x1

dy = y2 - y1

# Draw arrow

ax.annotate('', xy=(x2, y2), xytext=(x1, y1),

arrowprops=dict(arrowstyle='->', color='red', lw=2, alpha=0.7),

zorder=5)

# Add row labels

add\_row\_labels(ax, path\_lanes\_list, sow\_flags\_list, rover\_width\_m, rover\_length\_m,

plot\_padding\_m, max\_lane\_idx\_y)

# Add start marker

# Add start marker (rectangular rover shape)

start\_center = path\_metric\_centers[0]

rover\_rect\_width = rover\_width\_m

rover\_rect\_height = rover\_length\_m

rover\_start\_x = start\_center[0] - rover\_rect\_width/2

rover\_start\_y = start\_center[1] - rover\_rect\_height/2

ax.add\_patch(Rectangle((rover\_start\_x, rover\_start\_y), rover\_rect\_width, rover\_rect\_height,

color='orange', fill=True, lw=1.5, zorder=4,

edgecolor='black'))

ax.add\_patch(Rectangle((rover\_start\_x, rover\_start\_y), rover\_rect\_width, rover\_rect\_height,

color='orange', fill=True, lw=1.5, zorder=4,

edgecolor='black'))

# Add exit gate

add\_exit\_gate(ax, exit\_point\_lanes, exit\_metric\_center, rover\_width\_m, rover\_length\_m,

farm\_width\_m, farm\_breadth\_m, max\_lane\_idx\_x, max\_lane\_idx\_y)

# Add legend

legend\_handles = [

Line2D([0], [0], c=single\_brown\_color, lw=10, alpha=0.4,

label=f'Unsown Path'),

Line2D([0], [0], c='#006400', lw=10, alpha=0.8,

label=f'Sown Area'),

Rectangle((0, 0), 1, 1, fc='orange', ec='black',

label=f'🤖 Rover ({start\_center[0]:.1f}, {start\_center[1]:.1f})m'),

Line2D([0], [0], marker='o', mfc='white', mec='black', ms=8, ls='None',

label=f'Waypoints (1-{len(path\_metric\_centers)})'),

Line2D([0], [0], color='red', lw=2, alpha=0.7,

label='Path Direction →'),

Rectangle((0, 0), 1, 1, fc='red', ec='darkred',

label=f'EXIT ({exit\_metric\_center[0]:.1f}, {exit\_metric\_center[1]:.1f})m')

]

ax.legend(handles=legend\_handles, loc='upper right', bbox\_to\_anchor=(1.28, 1.02), fontsize=8)

plt.subplots\_adjust(right=0.75)

# Convert plot to base64 string

img\_buffer = io.BytesIO()

plt.savefig(img\_buffer, format='png', dpi=150, bbox\_inches='tight')

img\_buffer.seek(0)

img\_base64 = base64.b64encode(img\_buffer.getvalue()).decode()

plt.close()

return img\_base64

def add\_row\_labels(ax, path\_lanes\_list, sow\_flags\_list, rover\_width\_m, rover\_length\_m,

plot\_padding\_m, max\_lane\_idx\_y):

# Add VRow labels

vrow\_columns = set()

for i in range(len(path\_lanes\_list) - 1):

lx1, ly1 = path\_lanes\_list[i]

lx2, ly2 = path\_lanes\_list[i + 1]

is\_sown = i < len(sow\_flags\_list) and sow\_flags\_list[i]

if lx1 == lx2 and ly1 != ly2 and is\_sown:

vrow\_columns.add(lx1)

sorted\_vrow\_columns = sorted(vrow\_columns)

for i, column\_x in enumerate(sorted\_vrow\_columns):

tx\_m = (column\_x + 0.5) \* rover\_width\_m

ty\_m = -plot\_padding\_m \* 0.7

label = f'VRow{i+1}'

ax.text(tx\_m, ty\_m, label, fontsize=10, color='navy',

ha='center', va='center', weight='bold',

bbox=dict(boxstyle="round,pad=0.3", fc='lightblue',

alpha=0.9, ec='navy', lw=2))

# Add HRow labels

horizontal\_y\_positions = set()

for i in range(len(path\_lanes\_list) - 1):

x1, y1 = path\_lanes\_list[i]

x2, y2 = path\_lanes\_list[i + 1]

is\_sown = i < len(sow\_flags\_list) and sow\_flags\_list[i]

if y1 == y2 and x1 != x2 and is\_sown:

horizontal\_y\_positions.add(y1)

sorted\_horizontal\_positions = sorted(horizontal\_y\_positions)

for i, y\_pos in enumerate(sorted\_horizontal\_positions):

tx\_m = -plot\_padding\_m \* 0.7

ty\_m = (y\_pos + 0.5) \* rover\_length\_m

label = f'HRow{i+1}'

ax.text(tx\_m, ty\_m, label, fontsize=10, color='darkgreen', ha='center', va='center', weight='bold', rotation=0,

bbox=dict(boxstyle="round,pad=0.3", fc='lightgreen',

alpha=0.9, ec='darkgreen', lw=2))

def add\_exit\_gate(ax, exit\_point\_lanes, exit\_metric\_center, rover\_width\_m, rover\_length\_m,

farm\_width\_m, farm\_breadth\_m, max\_lane\_idx\_x, max\_lane\_idx\_y):

gate\_width = rover\_width\_m \* 0.8

gate\_height = rover\_length\_m \* 0.3

exit\_x, exit\_y = exit\_point\_lanes

if exit\_x == 0: # Left border

gate\_x = -gate\_height/2

gate\_y = exit\_metric\_center[1] - gate\_width/2

gate\_w, gate\_h = gate\_height, gate\_width

elif exit\_x == max\_lane\_idx\_x: # Right border

gate\_x = farm\_width\_m - gate\_height/2

gate\_y = exit\_metric\_center[1] - gate\_width/2

gate\_w, gate\_h = gate\_height, gate\_width

elif exit\_y == 0: # Bottom border

gate\_x = exit\_metric\_center[0] - gate\_width/2

gate\_y = -gate\_height/2

gate\_w, gate\_h = gate\_width, gate\_height

else: # Top border

gate\_x = exit\_metric\_center[0] - gate\_width/2

gate\_y = farm\_breadth\_m - gate\_height/2

gate\_w, gate\_h = gate\_width, gate\_height

exit\_gate = Rectangle((gate\_x, gate\_y), gate\_w, gate\_h,

color='red', alpha=0.8, zorder=5,

edgecolor='darkred', linewidth=2)

ax.add\_patch(exit\_gate)

@app.route('/download\_csv')

def download\_csv():

csv\_filename = "navigation\_log.csv"

if os.path.exists(csv\_filename):

return send\_file(csv\_filename, as\_attachment=True)

else:

return jsonify({'error': 'CSV file not found'}), 404

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, host='0.0.0.0', port=5000)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>🤖 Farm Robot Path Planner</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

min-height: 100vh;

padding: 20px;

}

.container {

max-width: 1400px;

margin: 0 auto;

background: white;

border-radius: 15px;

box-shadow: 0 20px 40px rgba(0,0,0,0.1);

overflow: hidden;

}

.header {

background: linear-gradient(135deg, #2c3e50 0%, #34495e 100%);

color: white;

padding: 30px;

text-align: center;

}

.header h1 {

font-size: 2.5em;

margin-bottom: 10px;

text-shadow: 2px 2px 4px rgba(0,0,0,0.3);

}

.header p {

font-size: 1.2em;

opacity: 0.9;

}

.main-content {

display: grid;

grid-template-columns: 1fr 2fr;

gap: 30px;

padding: 30px;

}

.controls-panel {

background: #f8f9fa;

border-radius: 10px;

padding: 25px;

height: fit-content;

}

.form-group {

margin-bottom: 20px;

}

.form-group label {

display: block;

margin-bottom: 8px;

font-weight: 600;

color: #2c3e50;

font-size: 14px;

}

.form-group input, .form-group select {

width: 100%;

padding: 12px;

border: 2px solid #e9ecef;

border-radius: 8px;

font-size: 16px;

transition: all 0.3s ease;

}

.form-group input:focus, .form-group select:focus {

outline: none;

border-color: #667eea;

box-shadow: 0 0 0 3px rgba(102, 126, 234, 0.1);

}

.section-title {

font-size: 18px;

font-weight: 700;

color: #2c3e50;

margin: 25px 0 15px 0;

padding-bottom: 8px;

border-bottom: 2px solid #667eea;

}

.exit-options {

display: none;

margin-top: 15px;

}

.exit-options.active {

display: block;

}

.custom-exit-inputs {

display: grid;

grid-template-columns: 1fr 1fr;

gap: 10px;

margin-top: 10px;

}

.btn {

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

color: white;

border: none;

padding: 15px 30px;

border-radius: 8px;

font-size: 16px;

font-weight: 600;

cursor: pointer;

transition: all 0.3s ease;

width: 100%;

margin-top: 20px;

}

.btn:hover {

transform: translateY(-2px);

box-shadow: 0 10px 20px rgba(102, 126, 234, 0.3);

}

.btn:disabled {

background: #95a5a6;

cursor: not-allowed;

transform: none;

box-shadow: none;

}

.results-panel {

background: white;

border-radius: 10px;

padding: 25px;

border: 1px solid #e9ecef;

}

.loading {

text-align: center;

padding: 50px;

color: #667eea;

}

.spinner {

border: 4px solid #f3f3f3;

border-top: 4px solid #667eea;

border-radius: 50%;

width: 50px;

height: 50px;

animation: spin 1s linear infinite;

margin: 0 auto 20px;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

.error {

background: #fee;

border: 1px solid #fcc;

color: #c33;

padding: 15px;

border-radius: 8px;

margin-bottom: 20px;

}

.success {

background: #efe;

border: 1px solid #cfc;

color: #3c3;

padding: 15px;

border-radius: 8px;

margin-bottom: 20px;

}

.plot-container {

text-align: center;

margin-bottom: 30px;

}

.plot-container img {

max-width: 100%;

height: auto;

border-radius: 8px;

box-shadow: 0 10px 30px rgba(0,0,0,0.1);

}

.statistics {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

gap: 15px;

margin-top: 20px;

}

.stat-card {

background: linear-gradient(135deg, #f8f9fa 0%, #e9ecef 100%);

padding: 20px;

border-radius: 8px;

text-align: center;

border: 1px solid #dee2e6;

}

.stat-value {

font-size: 24px;

font-weight: 700;

color: #667eea;

margin-bottom: 5px;

}

.stat-label {

font-size: 14px;

color: #6c757d;

font-weight: 500;

}

.download-btn {

background: linear-gradient(135deg, #28a745 0%, #20c997 100%);

margin-top: 15px;

}

.download-btn:hover {

box-shadow: 0 10px 20px rgba(40, 167, 69, 0.3);

}

@media (max-width: 768px) {

.main-content {

grid-template-columns: 1fr;

}

.header h1 {

font-size: 2em;

}

.statistics {

grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

}

}

.input-hint {

font-size: 12px;

color: #6c757d;

margin-top: 5px;

font-style: italic;

}

.validation-error {

color: #dc3545;

font-size: 12px;

margin-top: 5px;

}

</style>

</head>

<body>

<div class="container">

<div class="header">

<h1>🤖 Farm Robot Path Planner</h1>

<p>Advanced Agricultural Automation System v12.9</p>

</div>

<div class="main-content">

<div class="controls-panel">

<form id="pathForm">

<div class="section-title">🚜 Farm Dimensions</div>

<div class="form-group">

<label for="farm\_width">Farm Width (X-axis) - meters</label>

<input type="number" id="farm\_width" name="farm\_width" step="0.1" min="0.1" required>

<div class="input-hint">Total width of the farming area</div>

</div>

<div class="form-group">

<label for="farm\_breadth">Farm Breadth (Y-axis) - meters</label>

<input type="number" id="farm\_breadth" name="farm\_breadth" step="0.1" min="0.1" required>

<div class="input-hint">Total height/depth of the farming area</div>

</div>

<div class="section-title">🤖 Rover Specifications</div>

<div class="form-group">

<label for="rover\_width">Rover Width (Lane Spacing) - meters</label>

<input type="number" id="rover\_width" name="rover\_width" step="0.1" min="0.1" required>

<div class="input-hint">Width of rover for vertical lane spacing</div>

</div>

<div class="form-group">

<label for="rover\_length">Rover Length (Coverage Depth) - meters</label>

<input type="number" id="rover\_length" name="rover\_length" step="0.1" min="0.1" required>

<div class="input-hint">Length of rover for horizontal coverage</div>

</div>

<div class="section-title">🎯 Exit Configuration</div>

<div class="form-group">

<label for="exit\_type">Exit Type</label>

<select id="exit\_type" name="exit\_type" required>

<option value="">Select Exit Type</option>

<option value="corner">Corner Exit</option>

<option value="custom">Custom Boundary Exit</option>

</select>

</div>

<div id="corner\_options" class="exit-options">

<div class="form-group">

<label for="corner\_choice">Corner Selection</label>

<select id="corner\_choice" name="corner\_choice">

<option value="">Select Corner</option>

<option value="top\_left">Top-Left Corner</option>

<option value="top\_right">Top-Right Corner</option>

<option value="bottom\_left">Bottom-Left Corner</option>

<option value="bottom\_right">Bottom-Right Corner</option>

</select>

</div>

</div>

<div id="custom\_options" class="exit-options">

<div class="form-group">

<label>Custom Exit Coordinates (Lane Indices)</label>

<div class="custom-exit-inputs">

<input type="number" id="exit\_x" name="exit\_x" placeholder="X Lane" min="0">

<input type="number" id="exit\_y" name="exit\_y" placeholder="Y Lane" min="0">

</div>

<div class="input-hint">Exit must be on farm boundary</div>

</div>

</div>

<div class="section-title">⚙️ Advanced Settings</div>

<button type="submit" class="btn" id="generateBtn">

🚀 Generate Path Plan

</button>

</form>

</div>

<div class="results-panel">

<div id="results">

<div style="text-align: center; padding: 50px; color: #6c757d;">

<h3>🎯 Ready to Plan Your Farm Route</h3>

<p>Fill in the parameters on the left and click "Generate Path Plan" to see your optimized farming route.</p>

<div style="margin-top: 30px; text-align: left; max-width: 400px; margin-left: auto; margin-right: auto;">

<h4>📋 Quick Start Guide:</h4>

<ol style="margin-top: 15px; line-height: 1.6;">

<li>Enter your farm dimensions</li>

<li>Specify rover specifications</li>

<li>Choose exit point location</li>

<li>Generate your optimized path</li>

</ol>

</div>

</div>

</div>

</div>

</div>

</div>

<script>

// Form handling and validation

document.getElementById('exit\_type').addEventListener('change', function() {

const exitType = this.value;

const cornerOptions = document.getElementById('corner\_options');

const customOptions = document.getElementById('custom\_options');

// Hide all options first

cornerOptions.classList.remove('active');

customOptions.classList.remove('active');

// Show relevant options

if (exitType === 'corner') {

cornerOptions.classList.add('active');

} else if (exitType === 'custom') {

customOptions.classList.add('active');

}

});

// Real-time validation for farm dimensions

function validateDimensions() {

const farmWidth = parseFloat(document.getElementById('farm\_width').value) || 0;

const farmBreadth = parseFloat(document.getElementById('farm\_breadth').value) || 0;

const roverWidth = parseFloat(document.getElementById('rover\_width').value) || 0;

const roverLength = parseFloat(document.getElementById('rover\_length').value) || 0;

// Clear previous validation messages

document.querySelectorAll('.validation-error').forEach(el => el.remove());

let isValid = true;

// Check minimum farm width (need at least 3 lanes)

if (farmWidth > 0 && roverWidth > 0) {

const numLanesX = Math.floor(farmWidth / roverWidth);

if (numLanesX < 3) {

showValidationError('farm\_width', `Need at least ${(3 \* roverWidth).toFixed(1)}m width for rover width ${roverWidth}m`);

isValid = false;

}

}

// Check minimum farm breadth

if (farmBreadth > 0 && roverLength > 0) {

const numLanesY = Math.floor(farmBreadth / roverLength);

if (numLanesY < 1) {

showValidationError('farm\_breadth', `Need at least ${roverLength}m breadth for rover length ${roverLength}m`);

isValid = false;

}

}

return isValid;

}

function showValidationError(fieldId, message) {

const field = document.getElementById(fieldId);

const errorDiv = document.createElement('div');

errorDiv.className = 'validation-error';

errorDiv.textContent = message;

field.parentNode.appendChild(errorDiv);

}

// Add event listeners for real-time validation

['farm\_width', 'farm\_breadth', 'rover\_width', 'rover\_length'].forEach(id => {

document.getElementById(id).addEventListener('input', validateDimensions);

});

// Update custom exit coordinate limits based on farm dimensions

function updateExitLimits() {

const farmWidth = parseFloat(document.getElementById('farm\_width').value) || 0;

const farmBreadth = parseFloat(document.getElementById('farm\_breadth').value) || 0;

const roverWidth = parseFloat(document.getElementById('rover\_width').value) || 0;

const roverLength = parseFloat(document.getElementById('rover\_length').value) || 0;

if (farmWidth > 0 && farmBreadth > 0 && roverWidth > 0 && roverLength > 0) {

const maxLaneX = Math.floor(farmWidth / roverWidth) - 1;

const maxLaneY = Math.floor(farmBreadth / roverLength) - 1;

document.getElementById('exit\_x').max = maxLaneX;

document.getElementById('exit\_y').max = maxLaneY;

document.getElementById('exit\_x').placeholder = `X Lane (0-${maxLaneX})`;

document.getElementById('exit\_y').placeholder = `Y Lane (0-${maxLaneY})`;

}

}

['farm\_width', 'farm\_breadth', 'rover\_width', 'rover\_length'].forEach(id => {

document.getElementById(id).addEventListener('input', updateExitLimits);

});

// Form submission

document.getElementById('pathForm').addEventListener('submit', async function(e) {

e.preventDefault();

if (!validateDimensions()) {

return;

}

const formData = new FormData(this);

const data = Object.fromEntries(formData.entries());

// Validate exit configuration

if (data.exit\_type === 'corner' && !data.corner\_choice) {

alert('Please select a corner for corner exit');

return;

}

if (data.exit\_type === 'custom' && (!data.exit\_x || !data.exit\_y)) {

alert('Please enter both X and Y coordinates for custom exit');

return;

}

const generateBtn = document.getElementById('generateBtn');

const resultsDiv = document.getElementById('results');

// Show loading state

generateBtn.disabled = true;

generateBtn.textContent = '⏳ Generating Path...';

resultsDiv.innerHTML = `

<div class="loading">

<div class="spinner"></div>

<h3>🤖 Calculating Optimal Path...</h3>

<p>Please wait while we generate your farm route</p>

</div>

`;

try {

const response = await fetch('/generate\_path', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(data)

});

const result = await response.json();

if (result.success) {

displayResults(result);

} else {

displayError(result.error);

}

} catch (error) {

displayError('Network error: ' + error.message);

} finally {

generateBtn.disabled = false;

generateBtn.textContent = '🚀 Generate Path Plan';

}

});

function displayResults(result) {

const resultsDiv = document.getElementById('results');

resultsDiv.innerHTML = `

<div class="success">

✅ Path generation completed successfully!

</div>

<div class="plot-container">

<img src="data:image/png;base64,${result.plot\_data}" alt="Farm Path Plan" />

</div>

<div class="section-title">📊 Mission Statistics</div>

<div class="statistics">

<div class="stat-card">

<div class="stat-value">${result.statistics.grid\_size}</div>

<div class="stat-label">Grid Layout</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.total\_waypoints}</div>

<div class="stat-label">Total Waypoints</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.sown\_segments}</div>

<div class="stat-label">Sown Segments</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.unsown\_segments}</div>

<div class="stat-label">Unsown Segments</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.total\_distance}</div>

<div class="stat-label">Total Distance</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.sown\_distance}</div>

<div class="stat-label">Sown Distance</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.efficiency}</div>

<div class="stat-label">Path Efficiency</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.coverage}</div>

<div class="stat-label">Farm Coverage</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.inner\_sweeps}</div>

<div class="stat-label">Inner Sweeps</div>

</div>

<div class="stat-card">

<div class="stat-value">${result.statistics.exit\_position}</div>

<div class="stat-label">Exit Position</div>

</div>

</div>

<button class="btn download-btn" onclick="downloadCSV()">

📥 Download Telemetry CSV

</button>

<div style="margin-top: 20px; padding: 15px; background: #f8f9fa; border-radius: 8px; font-size: 14px; color: #6c757d;">

<strong>📋 Path Details:</strong><br>

• Grid: ${result.statistics.grid\_size} lanes (vertical × horizontal)<br>

• Route: ${result.statistics.total\_waypoints} waypoints with ${result.statistics.total\_segments} segments<br>

• Productivity: ${result.statistics.sown\_segments} productive segments, ${result.statistics.unsown\_segments} positioning moves<br>

• Distance: ${result.statistics.total\_distance} total (${result.statistics.sown\_distance} productive)<br>

• Exit: ${result.statistics.exit\_position}

</div>

`;

}

function displayError(error) {

const resultsDiv = document.getElementById('results');

resultsDiv.innerHTML = `

<div class="error">

❌ <strong>Error:</strong> ${error}

</div>

<div style="text-align: center; padding: 30px; color: #6c757d;">

<h3>🔧 Troubleshooting Tips:</h3>

<ul style="text-align: left; max-width: 400px; margin: 20px auto; line-height: 1.6;">

<li>Ensure farm width allows at least 3 vertical lanes</li>

<li>Check that rover dimensions are reasonable</li>

<li>Verify exit coordinates are on farm boundary</li>

<li>Make sure all required fields are filled</li>

</ul>

</div>

`;

}

async function downloadCSV() {

try {

const response = await fetch('/download\_csv');

if (response.ok) {

const blob = await response.blob();

const url = window.URL.createObjectURL(blob);

const a = document.createElement('a');

a.style.display = 'none';

a.href = url;

a.download = 'navigation\_log.csv';

document.body.appendChild(a);

a.click();

window.URL.revokeObjectURL(url);

} else {

alert('CSV file not available for download');

}

} catch (error) {

alert('Error downloading CSV: ' + error.message);

}

}

// Auto-fill example values for demo

function loadExampleValues() {

document.getElementById('farm\_width').value = '50';

document.getElementById('farm\_breadth').value = '40';

document.getElementById('rover\_width').value = '10';

document.getElementById('rover\_length').value = '8';

document.getElementById('exit\_type').value = 'corner';

document.getElementById('exit\_type').dispatchEvent(new Event('change'));

document.getElementById('corner\_choice').value = 'top\_right';

validateDimensions();

updateExitLimits();

}

// Add example button

window.addEventListener('load', function() {

const exampleBtn = document.createElement('button');

exampleBtn.type = 'button';

exampleBtn.className = 'btn';

exampleBtn.style.background = 'linear-gradient(135deg, #17a2b8 0%, #138496 100%)';

exampleBtn.style.marginTop = '10px';

exampleBtn.innerHTML = '📝 Load Example Values';

exampleBtn.onclick = loadExampleValues;

const form = document.getElementById('pathForm');

form.appendChild(exampleBtn);

});

// Keyboard shortcuts

document.addEventListener('keydown', function(e) {

if (e.ctrlKey && e.key === 'Enter') {

document.getElementById('pathForm').dispatchEvent(new Event('submit'));

}

});

// Auto-save form data to localStorage

function saveFormData() {

const formData = new FormData(document.getElementById('pathForm'));

const data = Object.fromEntries(formData.entries());

localStorage.setItem('farmPlannerData', JSON.stringify(data));

}

function loadFormData() {

const saved = localStorage.getItem('farmPlannerData');

if (saved) {

try {

const data = JSON.parse(saved);

Object.keys(data).forEach(key => {

const element = document.getElementById(key);

if (element) {

element.value = data[key];

if (element.type === 'select-one') {

element.dispatchEvent(new Event('change'));

}

}

});

validateDimensions();

updateExitLimits();

} catch (e) {

console.log('Error loading saved data:', e);

}

}

}

// Save form data on input changes

document.getElementById('pathForm').addEventListener('input', saveFormData);

document.getElementById('pathForm').addEventListener('change', saveFormData);

// Load saved data on page load

window.addEventListener('load', loadFormData);

// Add tooltips for better UX

const tooltips = {

'farm\_width': 'The total width of your farming area in meters. This determines how many vertical passes the rover will make.',

'farm\_breadth': 'The total height/depth of your farming area in meters. This determines the length of each pass.',

'rover\_width': 'The effective working width of your rover. This sets the spacing between vertical lanes.',

'rover\_length': 'The effective working length of your rover. This determines horizontal coverage depth.',

'exit\_type': 'Choose whether to exit from a corner or a custom boundary position.',

'gap\_size': 'Size of gaps left around the exit point to ensure safe rover exit.'

};

Object.keys(tooltips).forEach(id => {

const element = document.getElementById(id);

if (element) {

element.title = tooltips[id];

}

});

</script>

</body>

</html>