Internship Project Documentation: VSPAERO Wing Optimization System

Executive Summary

This documentation provides a comprehensive overview of a wing optimization project that utilizes OpenVSP (Open Vehicle Sketch Pad) and VSPAERO for aerodynamic analysis, combined with particle swarm optimization to design an optimal wing geometry for a given objective.

Project Objective: Develop an automated system to optimize wing geometry parameters for a small aircraft (7 kg) to maximize flight endurance while maintaining stability and lift constraints.

Key Technologies:

- OpenVSP/VSPAERO: Aerodynamic simulation and analysis
- Python 3.x with Platypus library: Multi-objective optimization framework
- SMPSO Algorithm: Speed-constrained Multi-objective Particle Swarm Optimization
- ClarkY Airfoil: Standard airfoil profile for analysis

Table of Contents

---

1. Project Overview

1.1 Background

The project focuses on optimizing wing geometry for a lightweight unmanned aerial vehicle (UAV) with a total weight of 7 kg. The primary goal is to maximize endurance, which is critical for surveillance, reconnaissance, and long-duration flight missions.

1.2 Problem Statement

Given a fixed aircraft weight and operational constraints, determine the optimal wing geometry (span, area, taper ratio, sweep angle, twist, and root incidence) that maximizes the endurance parameter (Cl^1.5/Cd) while satisfying:

- Stability requirements (negative pitching moment)
- Lift coefficient bounds ($0.2 \leq Cl \leq 0.25$)
- Wing loading constraints (4.8 to 6.2 kg/m^2)

1.3 Methodology

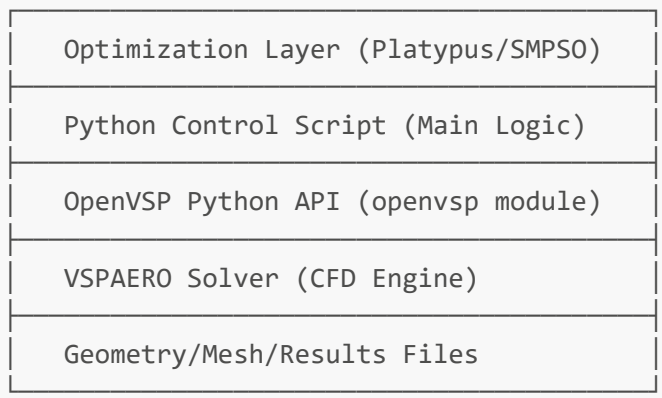
The project employs a computational fluid dynamics (CFD)-based optimization approach:

1. **Parametric Wing Modeling:** Use OpenVSP to generate wing geometries programmatically
2. **Aerodynamic Analysis:** Run VSPAERO panel method simulations to compute aerodynamic coefficients
3. **Optimization Loop:** Use SMPSO (particle swarm optimization) to explore the design space
4. **Constraint Handling:** Enforce physical and performance constraints throughout optimization

---

2. System Architecture

2.1 Software Stack



2.2 Data Flow

```
Configuration JSON → Python Script → OpenVSP Model →
VSPAERO Mesh → VSPAERO Solver → Polar Results →
Objective Function → Optimizer → Next Design Point
```

2.3 File Structure

- **structured_vspaero_optimizer.py**: Main optimization script (420 lines)
- **optimizer_config.json**: Configuration file with all parameters
- **ClarkY.dat**: Airfoil coordinate data (68 points)
- **aero_results.csv**: Detailed aerodynamic results for each iteration
- **wingopt_results.csv**: Optimization progress and best solutions

- **modified_wing.vsp3**: VSP geometry file (for debugging)
- **modified_wing.polar**: VSPAERO output with aerodynamic coefficients

---

3. Configuration System

3.1 Configuration File Structure

The system uses a JSON-based configuration approach for maximum flexibility and reproducibility. The configuration is divided into five main sections:

3.1.1 Environment Settings

```
"environment": {
    "vsp_path": "AUTO",
    "airfoil_file": "ClarkY.dat",
    "aero_results_file": "aero_results.csv",
    "opt_results_file": "wingopt_results.csv",
    "debug_vsp_file": "modified_wing.vsp3",
    "polar_file": "modified_wing.polar"
}
```

Purpose: Defines file paths and system integration points.

Key Parameters:

- `vsp_path`: Path to VSPAERO executable (AUTO enables automatic discovery)
- `airfoil_file`: Airfoil geometry definition
- Output files for results tracking and debugging

3.1.2 Aircraft Parameters

```
"aircraft": {
    "weight_kg": 7.0,
    "wing_loading_min": 4.8,
    "wing_loading_max": 6.2
}
```

Purpose: Defines the aircraft physical constraints.

Design Rationale:

- Weight: 7 kg typical for small UAVs
- Wing loading range: Balances structural efficiency with lift requirements
- These values directly constrain the allowable wing area

Calculations:

- Minimum wing area = 7.0 / 6.2 = 1.129 m^2 (full span)
- Maximum wing area = 7.0 / 4.8 = 1.458 m^2 (full span)
- Semi-area bounds: 0.565 to 0.729 m^2

3.1.3 Simulation Parameters

```
"simulation": {
    "rho_kg_m3": 1.2256,
    "velocity_m_s": 20.0,
    "viscosity": 1.784e-05,
    "mach": 0.06,
    "aoa_deg": 0.0,
    "beta_deg": 0.0
}
```

Purpose: Defines the flight conditions for analysis.

Physical Interpretation:

- **Density (ρ):** 1.2256 kg/m^3 = Sea level ISA conditions
- **Velocity:** 20 m/s ≈ 72 km/h = Typical cruise speed for small UAVs
- **Viscosity:** 1.784×10^{-5} Pa·s = Standard air viscosity
- **Mach Number:** 0.06 = Low-speed incompressible flow
- **Angle of Attack (α):** 0° = Trimmed cruise condition
- **Sideslip (β):** 0° = Symmetric flight

Reynolds Number Calculation: Re = (ρ × V × MAC) / μ Where MAC (Mean Aerodynamic Chord) is computed by OpenVSP based on wing geometry.

3.1.4 Design Variables (Optimization Bounds)

```
"design_variables": {
    "semi_span": {"min": 0.75, "max": 1.25},
    "semi_area": {"min": "AUTO", "max": "AUTO"},
    "taper": {"min": 0.2, "max": 1.0},
    "sweep": {"min": 0.0, "max": 5.0},
    "twist": {"min": -5.0, "max": 5.0},
    "root_incidence": {"min": -5.0, "max": 5.0}
}
```

Detailed Variable Descriptions:

1. **Semi-Span (m)**
    - Range: 0.75 to 1.25 m
    - Full wingspan: 1.5 to 2.5 m
    - Physical meaning: Half the wing span (tip to root)

- Impact: Longer spans increase lift efficiency but add structural weight

2. **Semi-Area (m^2)**
   - Range: Computed automatically from wing loading constraints
   - Calculated as: Weight / (2 × Wing_Loading)
   - Physical meaning: Half the total planform area
   - Impact: Larger area increases lift but also drag

3. **Taper Ratio (dimensionless)**
   - Range: 0.2 to 1.0
   - Definition: Tip_Chord / Root_Chord
   - 1.0 = Rectangular wing
   - 0.2 = Highly tapered wing
   - Impact: Affects spanwise lift distribution and induced drag

4. **Sweep Angle (degrees)**
   - Range: 0° to 5°
   - Physical meaning: Leading edge sweep at quarter-chord
   - Impact: Small sweep can improve stability; large sweep increases drag at low speeds

5. **Twist (degrees)**
   - Range: -5° to +5°
   - Also called "washout" when negative
   - Physical meaning: Geometric twist from root to tip
   - Impact: Optimizes spanwise lift distribution, affects stall characteristics

6. **Root Incidence (degrees)**
   - Range: -5° to +5°
   - Physical meaning: Angle of the root airfoil relative to fuselage reference
   - Impact: Adjusts cruise angle of attack and pitching moment

3.1.5 Optimization Settings

```
"optimization": {
    "swarm_size": 50,
    "max_evaluations": 50,
    "step_size": 1
}
```

Purpose: Controls the optimization algorithm parameters.

SMPSO Configuration:

- **Swarm Size:** 50 particles exploring the design space simultaneously
- **Max Evaluations:** Total function evaluations before termination

- **Step Size:** Granularity of progress reporting (1 = report every evaluation)

Computational Cost: Total VSPAERO runs ≈ Swarm_Size × (Max_Evaluations / Step_Size) = 2,500 simulations

3.2 Configuration Loading System

The script implements robust configuration loading with error handling:

```
def load_config(config_path=CONFIG_FILE):
    if not os.path.exists(config_path):
        print(f"Error: Config file '{config_path}' not found.")
        sys.exit(1)
    with open(config_path, 'r') as f:
        return json.load(f)
```

Features:

- Validates file existence before loading
- Provides clear error messages
- Exits gracefully on missing configuration

---

4. Airfoil Data

4.1 ClarkY Airfoil Overview

Historical Context: The ClarkY airfoil is a classic NACA-era design developed by Virginius E. Clark in the 1920s. It has been widely used in general aviation due to its:

- Gentle stall characteristics
- Good lift-to-drag ratio at low Reynolds numbers
- Predictable handling qualities
- Relatively flat lower surface (simplifies construction)

Specifications:

- Maximum thickness: 11.7% of chord
- Maximum camber: ~3.5% at ~33% chord
- Suitable Reynolds number range: 100,000 to 1,000,000

4.2 Coordinate Data Structure

The ClarkY.dat file contains 68 coordinate points defining the airfoil shape:

Format: Two-column space-separated values

```
x/c    y/c
1.00000  0.0       (Trailing edge)
0.99572  0.00115   (Upper surface)
...
```

```
0.0      0.00047   (Leading edge)
...
0.99572 -0.00025   (Lower surface)
1.00000  0.0       (Trailing edge closure)
```

Coordinate System:

- x/c: Chordwise position (0 = leading edge, 1 = trailing edge)
- y/c: Vertical position normalized by chord length
- Positive y = upper surface (suction side)
- Negative y = lower surface (pressure side)

4.3 Key Geometric Features

Upper Surface Analysis:

- Maximum thickness at x/c ≈ 0.30 (30% chord)
- Peak y/c = 0.09318 at x/c = 0.33928
- Smooth contour with gradual thickness reduction toward trailing edge

Lower Surface Analysis:

- Nearly flat from 20% to 60% chord
- Maximum lower surface depth: y/c = -0.02839 at x/c = 0.14645
- Gentle curvature provides stable pressure distribution

Aerodynamic Implications:

- Forward location of maximum thickness → good stall characteristics
- Flat lower surface → easier manufacturing, predictable pressure distribution
- Rounded leading edge → tolerance to slight angle of attack variations

4.4 Usage in Simulation

OpenVSP reads this file and:

1. Interpolates the coordinates to create a smooth surface
2. Applies the airfoil profile to both root and tip sections
3. Performs linear lofting between sections to create 3D wing surface
4. Generates panel mesh for VSPAERO analysis

---

5. Core Components

5.1 Import and Environment Setup

5.1.1 Standard Library Imports

```
import os, sys, math, csv, contextlib, time
```

Purpose of Each Module:

- `os`: File system operations, path manipulation, file descriptor control
- `sys`: System-specific parameters, module path manipulation, exit handling
- `math`: Mathematical functions (though not heavily used in this script)
- `csv`: Reading/writing optimization results
- `contextlib`: Creating the output suppression context manager
- `time`: Potential timing operations (imported but not actively used)

5.1.2 Third-Party Libraries

```
from tqdm import tqdm
from platypus import Problem, Real, SMPSO, PM
```

TQDM:

- Progress bar library for tracking optimization iterations
- Provides visual feedback during long-running simulations
- Updates in real-time with remaining time estimates

Platypus Optimization Framework:

- `Problem`: Defines the optimization problem structure
- `Real`: Continuous real-valued decision variables
- `SMPSO`: Speed-constrained Multi-objective Particle Swarm Optimization
- `PM`: Polynomial Mutation operator for diversity maintenance

5.1.3 OpenVSP Integration

```
try:
    import openvsp as vsp
except ImportError:
    print("Error: Could not import 'openvsp'.")
    sys.exit(1)
```

Robust Import Handling:

- Attempts to import OpenVSP Python bindings
- Provides clear error message if not available
- Exits gracefully to prevent confusing downstream errors

OpenVSP Module Capabilities:

- Parametric geometry creation and modification
- Mesh generation for CFD analysis
- Integration with VSPAERO solver
- Result file parsing and extraction

5.2 VSPAERO Path Discovery

The system implements intelligent path discovery for the VSPAERO executable:

```
def find_vspaero_path():
    # 0. Check Config First
    cfg_path = CONFIG["environment"].get("vsp_path", "AUTO")
    if cfg_path != "AUTO" and os.path.exists(cfg_path):
        return cfg_path

    # 1. Check OpenVSP package directory
    if 'openvsp' in sys.modules:
        import openvsp
        pkg_dir = os.path.dirname(openvsp.__file__)
        possible_loc = os.path.join(pkg_dir,
            "vspaero.exe" if os.name == 'nt' else "vspaero")
        if os.path.exists(possible_loc):
            return pkg_dir

    # 2. Check System PATH
    import shutil
    cmd = "vspaero.exe" if os.name == 'nt' else "vspaero"
    path_loc = shutil.which(cmd)
    if path_loc:
        return os.path.dirname(path_loc)

    return None
```

Discovery Strategy (Priority Order):

1. **Explicit Configuration:** Check if user provided path in config file
2. **Package Location:** Look in Python's openvsp module installation directory
3. **System PATH:** Search system PATH for vspaero executable
4. **Fallback:** Return None and issue warning

Cross-Platform Support:

- Detects operating system (`os.name == 'nt'` for Windows)
- Uses appropriate executable name (.exe on Windows)
- Platform-agnostic path joining

5.3 Utility Functions

5.3.1 Output Suppression

```
@contextlib.contextmanager
def suppress_output_fd():
    """
    Suppresses ALL output from Python and C-level extensions.
```

```
    Redirects stdout (1) and stderr (2) to os.devnull.
    """
    with open(os.devnull, 'w') as devnull:
        old_stdout_fd = os.dup(1)
        old_stderr_fd = os.dup(2)
        try:
            os.dup2(devnull.fileno(), 1)
            os.dup2(devnull.fileno(), 2)
            yield
        finally:
            os.dup2(old_stdout_fd, 1)
            os.dup2(old_stderr_fd, 2)
            os.close(old_stdout_fd)
            os.close(old_stderr_fd)
```

Technical Details:

- **File Descriptor Level:** Works at OS level, not just Python level
- **Why Necessary:** OpenVSP and VSPAERO produce verbose C++ output that clutters terminal
- **Mechanism:**
    1. Duplicate original stdout/stderr file descriptors
    2. Redirect file descriptors 1 and 2 to /dev/null
    3. Execute code block (yield)
    4. Restore original file descriptors
    5. Clean up duplicated descriptors

Context Manager Pattern:

- Ensures proper cleanup even if exceptions occur
- Automatic restoration of output streams
- Used with `with` statement for clean syntax

5.3.2 Polar File Parser

```
def parse_polar_file(filepath):
    """
    Parses VSPAERO .polar output file.
    Returns list of dictionaries with aerodynamic coefficients.
    """
```

VSPAERO Polar File Format:

```
Beta    Alpha    Mach     CL       CD       ...
0.0     0.0      0.06     0.2234   0.0123   ...
```

Parsing Algorithm:

1. Read entire file as text lines
2. Search for header line (starts with "Beta")
3. Extract column names from header
4. Parse subsequent data lines
5. Convert strings to floats
6. Return list of dictionaries (one per flight condition)

Error Handling:

- Checks file existence before opening
- Handles malformed lines gracefully
- Returns empty list if parsing fails
- Validates column count matches header

Data Structure:

```
[
    {"Beta": 0.0, "Alpha": 0.0, "CLtot": 0.2234, "CDtot": 0.0123, ...},
    ...
]
```

5.3.3 Progress Tracking

```
_vspaero_call_count = 0  # Global counter
_progress_bar = None      # TQDM instance
```

Global State Management:

- Tracks total number of VSPAERO simulations
- Maintains progress bar reference across function calls
- Updated by `run_vspaero_analysis()` function

5.4 Environment Initialization

```
def init_environment():
    """Sets up VSP path and initializes output files."""
    vsp.SetVSPAEROPath(VSP_PATH)

    with open(AERO_RESULTS_FILE, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["Iteration", "Cl", "Cd", "Cm", "L/D"])
```

Tasks Performed:

1. **Configure VSPAERO Path:** Tell OpenVSP where to find solver executables

2. **Initialize Results File:** Create CSV with header row for detailed aerodynamic data
3. **Prepare File System:** Ensure output directory is writable

CSV Structure:

- Iteration: Sequential simulation number
- Cl: Lift coefficient
- Cd: Drag coefficient
- Cm: Pitching moment coefficient
- L/D: Lift-to-drag ratio

---

6. Design Variables and Constraints

6.1 Design Variable Space

The optimization explores a 6-dimensional design space:

Variable	Symbol	Min	Max	Unit	Description
Semi-Span	b/2	0.75	1.25	m	Half wingspan
Semi-Area	S/2	0.565	0.729	m^2	Half planform area
Taper Ratio	λ	0.2	1.0	-	Tip/root chord ratio
Sweep	Λ	0	5	deg	Quarter-chord sweep
Twist	ε	-5	+5	deg	Geometric twist
Root Incidence	i_root	-5	+5	deg	Root airfoil angle

Design Space Volume: V = Π(max - min) for each dimension V ≈ 0.5 × 0.164 × 0.8 × 5 × 10 × 10 = 32.8 dimensional units

6.2 Constraint Formulation

The optimization enforces three inequality constraints:

6.2.1 Stability Constraint

Mathematical Form: Cm ≤ 0

Physical Meaning:

- Cm (pitching moment coefficient) must be negative or zero
- Negative Cm indicates nose-down moment → longitudinally stable
- Ensures aircraft naturally returns to trimmed flight after disturbance

Aerodynamic Theory:

- Cm = (M_aero) / (q × S × c_ref)
- Where M_aero is aerodynamic pitching moment about center of gravity

- Negative Cm provides restoring moment when angle of attack increases

6.2.2 Minimum Lift Constraint

Mathematical Form: $Cl \geq 0.2$ **Reformulated as:** $-Cl + 0.2 \leq 0$

Physical Meaning:

- Ensures sufficient lift generation at cruise speed
- Cl = 0.2 typical for efficient cruise in small UAVs
- Prevents optimizer from selecting too little area or unfavorable geometry

Lift Equation: $L = 0.5 \times \rho \times V^2 \times S \times Cl$ For level flight: $L = W = 7\ kg \times 9.81\ m/s^2 = 68.67\ N$

Verification: $Cl_{min} = 0.2$, $V = 20\ m/s$, $\rho = 1.2256\ kg/m^3$ Required $S = 2W / (\rho \times V^2 \times Cl) = 2.8\ m^2$ This is higher than our bounds, indicating constraint is active.

6.2.3 Maximum Lift Constraint

Mathematical Form: $Cl \leq 0.25$ **Reformulated as:** $Cl - 0.25 \leq 0$

Physical Meaning:

- Prevents operation too close to stall
- Maintains margin for maneuvers and gusts
- ClarkY stall typically occurs around Cl ≈ 1.4-1.6 (plenty of margin)

Safety Factor: Operating Cl / Stall Cl = 0.25 / 1.5 ≈ 0.167 (16.7% of stall Cl) This conservative margin ensures safe operation.

6.3 Constraint Handling in Optimizer

Platypus SMPSO uses **penalty function approach:**

1. Infeasible solutions receive poor fitness
2. Algorithm naturally gravitates toward feasible region
3. Only feasible solutions considered for "best" solution
4. Maintains population diversity through velocity damping

Constraint Checking:

```
c1 = cm                 # Must be ≤ 0
c2 = -cl + 0.2          # Must be ≤ 0
c3 = cl - 0.25          # Must be ≤ 0
```

All three must simultaneously satisfy `constraint ≤ 0` for feasibility.

---

7. Optimization Process

7.1 Objective Function

Primary Objective: Maximize Endurance Parameter

Mathematical Formulation:

```
Endurance ∝ (Cl^1.5) / Cd
```

Derivation from Breguet Endurance Equation:

For propeller-driven aircraft:

```
E = (η_prop / c_t) × (Cl / Cd) × sqrt(2 / (ρ × S × W))
```

Where:

- η_prop: Propeller efficiency
- c_t: Specific fuel consumption
- Cl/Cd: Aerodynamic efficiency

For maximum endurance at constant weight:

```
dE/dCl = 0  →  Cl_opt ∝ (Cd^0.5)
```

This leads to maximizing `Cl^1.5 / Cd` for optimal endurance.

Implementation:

```python
if cl <= 0 or cd <= 0:
    endurance = 0.0  # Penalize invalid solutions
else:
    endurance = -(cl**1.5) / cd  # Negative for minimization
```

Why Negative? Platypus minimizes objectives by default. To maximize endurance, we minimize its negative.

7.2 SMPSO Algorithm

SMPSO: Speed-constrained Multi-objective Particle Swarm Optimization

7.2.1 Algorithm Overview

Particle Swarm Optimization (PSO) Fundamentals:

- Population-based metaheuristic inspired by social behavior
- Each "particle" represents a candidate solution

- Particles move through design space based on:
  - Personal best position (cognitive component)
  - Global best position (social component)
  - Velocity (momentum)

SMPSO Enhancements:

1. **Speed Constraint:** Limits particle velocity to prevent premature convergence
2. **Constriction Coefficient:** Controls exploration vs exploitation balance
3. **Polynomial Mutation:** Adds diversity to prevent stagnation

7.2.2 Algorithm Configuration

```
algorithm = SMPSO(
    problem,
    swarm_size=50,
    leader_size=50,
    mutation=PM(probability=1.0/6.0, distribution_index=5)
)
```

Parameters Explained:

1. **Swarm Size (50):**
   - Number of particles exploring simultaneously
   - Larger swarm → better exploration, more computation
   - 50 is balanced for 6 design variables
2. **Leader Size (50):**
   - Archive of best solutions found
   - Used to guide particle movement
   - Set equal to swarm size for single-objective problems
3. **Polynomial Mutation:**
   - Probability: 1/6 ≈ 0.167 (typically 1/n_vars)
   - Distribution Index: 5 (controls mutation spread)
   - Lower index → more exploratory mutations

7.2.3 Velocity Update Equation

```
v_i(t+1) = χ × [v_i(t) + c1×r1×(pbest_i - x_i(t)) + c2×r2×(gbest - x_i(t))]
x_i(t+1) = x_i(t) + v_i(t+1)
```

Where:

- χ: Constriction coefficient (≈ 0.729)

- c1, c2: Cognitive and social acceleration coefficients
- r1, r2: Random numbers in [0,1]
- pbest: Personal best position
- gbest: Global best position

7.3 Optimization Loop Structure

```
for i in range(0, MAX_EVALUATIONS, STEP_SIZE):
    algorithm.run(STEP_SIZE)

    # Extract feasible solutions
    feasible_sols = [s for s in algorithm.result
                     if getattr(s, "feasible", False)]

    if feasible_sols:
        best = min(feasible_sols, key=lambda s: s.objectives[0])
        # Log to CSV
        writer.writerow([i + STEP_SIZE, best.objectives[0],
                         *best.variables])
        csvfile.flush()
```

Loop Mechanics:

1. **Iteration:** Run optimizer for STEP_SIZE evaluations
2. **Filtering:** Extract only feasible solutions (satisfy all constraints)
3. **Selection:** Find solution with minimum objective (best endurance)
4. **Logging:** Write to CSV for progress tracking
5. **Flushing:** Force write to disk immediately

Benefits:

- Incremental progress visibility
- Ability to stop and resume
- Real-time monitoring of convergence
- Recovery from crashes (partial results saved)

7.4 Convergence Criteria

Current Implementation: Fixed evaluation budget (50 evaluations)

Alternative Criteria (Not Implemented):

- Objective function tolerance (e.g., $\Delta f < 1e\text{-}6$)
- Design variable convergence ($\Delta x < 1e\text{-}4$)
- Population diversity collapse
- Maximum wall-clock time

Termination Conditions:

1. Reach MAX_EVALUATIONS

2. User interrupt (Ctrl+C)
3. Critical error in VSPAERO

---

8. Mathematical Formulation

8.1 Complete Optimization Problem

```
Minimize:    f(x) = -(Cl^1.5 / Cd)

Subject to:  g1(x) = Cm ≤ 0                    (Stability)
             g2(x) = -Cl + 0.2 ≤ 0              (Min Lift)
             g3(x) = Cl - 0.25 ≤ 0              (Max Lift)

             0.75 ≤ b/2 ≤ 1.25                  (Semi-span)
             0.565 ≤ S/2 ≤ 0.729                (Semi-area)
             0.2 ≤ λ ≤ 1.0                      (Taper)
             0 ≤ Λ ≤ 5                          (Sweep)
             -5 ≤ ε ≤ 5                         (Twist)
             -5 ≤ i_root ≤ 5                    (Root inc.)

Where:       x = [b/2, S/2, λ, Λ, ε, i_root]
```

8.2 Aerodynamic Coefficient Definitions

8.2.1 Lift Coefficient

```
Cl = L / (0.5 × ρ × V² × S)
```

Physical Interpretation:

- Dimensionless measure of lift generation
- Independent of size and speed
- Function of angle of attack and wing geometry

Typical Values:

- Cl = 0: Zero lift line
- Cl = 0.2-0.3: Efficient cruise
- Cl = 1.0-1.5: Maximum lift (stall)

8.2.2 Drag Coefficient

```
Cd = D / (0.5 × ρ × V² × S)
```

Drag Breakdown:

```
Cd = Cd_0 + Cd_i
```

Where:

- Cd_0: Parasite drag (friction, form, interference)
- Cd_i: Induced drag = Cl^2 / (π × AR × e)

Typical Values:

- Cd_0 ≈ 0.02-0.03: Low-drag wings
- Cd_i ≈ 0.005-0.015: Induced drag at cruise
- Cd_total ≈ 0.025-0.045: Typical cruise

8.2.3 Pitching Moment Coefficient

```
Cm = M / (0.5 × ρ × V² × S × c_ref)
```

Physical Interpretation:

- Moment about aerodynamic center or CG
- Positive Cm: Nose-up moment
- Negative Cm: Nose-down moment (stable)

Stability Requirement:

```
dCm/dCl < 0  (Static stability)
Cm < 0       (Trimmed stable flight)
```

8.2.4 Lift-to-Drag Ratio

```
L/D = Cl / Cd
```

Performance Implications:

- Maximum L/D → Maximum range
- (L/D)^0.5 × Cl → Maximum endurance
- High L/D indicates aerodynamic efficiency

8.3 Reynolds Number Calculation

```
Re = (ρ × V × L) / μ
```

For wing analysis:

```
Re = (1.2256 kg/m³ × 20 m/s × MAC) / (1.784×10⁻⁵ Pa·s)
Re ≈ 1.375×10⁶ × MAC
```

Example: If MAC = 0.3 m, then Re ≈ 412,000

Reynolds Number Effects:

- Re < 100,000: Laminar flow, high drag
- Re = 100,000-500,000: Transitional regime
- Re > 500,000: Fully turbulent, more predictable

8.4 Wing Loading Calculation

```
WL = W / S
```

Where:

- W = 7 kg × 9.81 m/s² = 68.67 N
- S = Wing planform area (m²)

Bounds:

```
4.8 kg/m² ≤ WL ≤ 6.2 kg/m²
47.09 N/m² ≤ WL ≤ 60.82 N/m²
```

Implications:

- Lower WL: Better climb, shorter takeoff, lower speed
- Higher WL: Better penetration, higher speed, less sensitive to gusts

8.5 Aspect Ratio

```
AR = b² / S
```

For our design:

```
AR_min = (2 × 0.75)² / (2 × 0.729) = 1.543
AR_max = (2 × 1.25)² / (2 × 0.565) = 5.531
```

Aspect Ratio Effects:

- High AR: Lower induced drag, better L/D, more flexible
- Low AR: Higher roll rate, structurally efficient, lower L/D

---

9. Implementation Details

9.1 VSPAERO Analysis Workflow

The `run_vspaero_analysis()` function implements a complete CFD workflow:

9.1.1 Geometry Construction

```
# 1. Clear existing model
vsp.ClearVSPModel()

# 2. Create wing component
wing_id = vsp.AddGeom("WING", "")

# 3. Set parametric drivers
section_idx = 1
vsp.SetDriverGroup(wing_id, section_idx,
    vsp.SPAN_WSECT_DRIVER,   # Drive by span
    vsp.AREA_WSECT_DRIVER,   # Drive by area
    vsp.TAPER_WSECT_DRIVER)  # Drive by taper
```

Driver Groups Explained:

- OpenVSP can define wing sections using different parameter sets
- SPAN + AREA + TAPER is most intuitive for optimization
- Alternative: ROOT_CHORD + TIP_CHORD + SPAN

Why This Choice?

- Direct control of key performance parameters
- Span affects induced drag
- Area determines wing loading
- Taper affects lift distribution

9.1.2 Parameter Setting

```
# Set geometric parameters
vsp.SetParmVal(wing_id, "Span", "XSec_1", semi_span)
vsp.SetParmVal(wing_id, "Area", "XSec_1", semi_area)
vsp.SetParmVal(wing_id, "Taper", "XSec_1", taper)
vsp.SetParmVal(wing_id, "Sweep", "XSec_1", sweep)
```

```
vsp.SetParmVal(wing_id, "Twist", "XSec_1", twist)
vsp.SetParmVal(wing_id, "Twist", "XSec_0", root_inc)
```

Section Indexing:

- XSec_0: Root section
- XSec_1: Tip section (for single-section wing)

Twist Convention:

- Root twist = Root incidence angle
- Tip twist = Total geometric twist
- Linear interpolation between sections

9.1.3 Tessellation Control

```
# Control mesh resolution
vsp.SetParmVal(wing_id, "SectTess_U", "XSec_1", 10)  # Spanwise
vsp.SetParmVal(wing_id, "Tess_W", "Shape", 20)       # Chordwise
vsp.SetParmVal(wing_id, "LECluster", "WingGeom", 0.2)
vsp.SetParmVal(wing_id, "TECluster", "WingGeom", 0.2)
```

Tessellation Parameters:

- **SectTess_U (10):** 10 spanwise panels per section
- **Tess_W (20):** 20 chordwise panels
- **LECluster (0.2):** Cluster 20% of points near leading edge
- **TECluster (0.2):** Cluster 20% of points near trailing edge

Mesh Quality Tradeoffs:

- More panels → better accuracy, longer computation
- Clustering → resolves high-gradient regions (LE, TE)
- Balance: Adequate resolution without excessive cost

Total Panel Count: Approximately 10 × 20 = 200 panels per half-wing Full symmetric model: ~400 panels

9.1.4 Airfoil Assignment

```
# Get cross-section surface
xsec_surf_id = vsp.GetXSecSurf(wing_id, 0)

# Change to file-based airfoil
vsp.ChangeXSecShape(xsec_surf_id, 0, vsp.XS_FILE_AIRFOIL)
vsp.ChangeXSecShape(xsec_surf_id, 1, vsp.XS_FILE_AIRFOIL)

# Get section handles
root_xsec = vsp.GetXSec(xsec_surf_id, 0)
```

```
tip_xsec = vsp.GetXSec(xsec_surf_id, 1)

# Load ClarkY coordinates
vsp.ReadFileAirfoil(root_xsec, AIRFOIL_FILE)
vsp.ReadFileAirfoil(tip_xsec, AIRFOIL_FILE)
```

Airfoil Application:

1. OpenVSP supports multiple airfoil types (NACA 4-digit, file-based, etc.)
2. FILE_AIRFOIL type reads custom coordinates
3. Same airfoil applied to root and tip (untapered airfoil)
4. VSP interpolates surface between sections

Constant vs. Tapered Airfoil:

- Constant: Same profile at root and tip (easier analysis)
- Tapered: Different profiles (more realistic, complex)
- This project uses constant ClarkY for simplicity

9.1.5 Reynolds Number Computation

```
# Get mean aerodynamic chord
MAC = vsp.GetParmVal(wing_id, "MAC", "WingGeom")

# Calculate Reynolds number
reynolds = (RHO * VELOCITY * MAC) / VISCOSITY
```

MAC (Mean Aerodynamic Chord):

- Representative chord for 2D-equivalent aerodynamics
- For tapered wings: MAC ≠ (c_root + c_tip) / 2
- OpenVSP computes exact MAC from planform geometry

Why MAC for Reynolds?

- Standardizes comparison across different planforms
- Captures effective viscous scale length
- Used in all coefficient definitions

9.1.6 Mesh Generation

```
# Setup geometry analysis
mesh_analysis = "VSPAEROComputeGeometry"
vsp.SetAnalysisInputDefaults(mesh_analysis)
vsp.SetIntAnalysisInput(mesh_analysis, "Symmetry", [2])
vsp.ExecAnalysis(mesh_analysis)
```

Symmetry Options:

- 0: No symmetry (full model)
- 1: X-Y plane symmetry
- 2: X-Z plane symmetry (typical for wings)
- Reduces computational cost by 50%

Output Files:

- .tri: Triangulated surface mesh
- .vspaero: Input file for flow solver

9.1.7 Flow Solution

```
analysis_name = "VSPAEROSweep"
vsp.SetAnalysisInputDefaults(analysis_name)

# Set reference quantities
vsp.SetDoubleAnalysisInput(analysis_name, "Sref", [2 * semi_area])
vsp.SetDoubleAnalysisInput(analysis_name, "bref", [2 * semi_span])
vsp.SetDoubleAnalysisInput(analysis_name, "cref", [MAC])

# Flight conditions
vsp.SetDoubleAnalysisInput(analysis_name, "AlphaStart", [AOA])
vsp.SetDoubleAnalysisInput(analysis_name, "AlphaEnd", [AOA])
vsp.SetIntAnalysisInput(analysis_name, "AlphaNpts", [1])

vsp.SetDoubleAnalysisInput(analysis_name, "ReCref", [reynolds])

# Execute
vsp.ExecAnalysis(analysis_name)
```

VSPAEROSweep Parameters:

1. **Reference Quantities:**
    - Sref: Reference area (full wing)
    - bref: Reference span (full wing)
    - cref: Reference chord (MAC)
2. **Sweep Variables:**
    - Alpha: Angle of attack (-180° to +180°)
    - Beta: Sideslip angle
    - Mach: Mach number
    - Reynolds: Reynolds number
3. **Single Point Analysis:**
    - Start = End = same value

- Npts = 1 (single evaluation)
- Efficient for optimization loops

VSPAERO Solution Method:

- Vortex lattice method (inviscid)
- Boundary layer corrections (viscous)
- Iterative convergence to force-free wake
- Outputs pressure distributions and integrated forces

9.2 Result Extraction

```
# Parse polar file
polar_results = parse_polar_file(POLAR_FILE)

if polar_results:
    res = polar_results[0]
    cl = res.get("CLtot", 0.0)
    cd = res.get("CDtot", 0.0)
    cm = res.get("CMytot", 0.0)
    ld = res.get("L/D", 0.0)
```

Polar File Contents:

- CLtot: Total lift coefficient
- CDtot: Total drag coefficient
- CDind: Induced drag component
- CDvis: Viscous drag component
- CMytot: Pitching moment (y-axis)
- CMxtot: Rolling moment
- CMztot: Yawing moment
- L/D: Pre-computed lift-to-drag ratio
- E: Oswald efficiency factor

Result Validation:

- Check for NaN or infinite values
- Verify physical reasonableness
- Return zeros if parsing fails (optimizer penalizes)

9.3 Progress Tracking and Logging

```
# Update global counter
_vspaero_call_count += 1

# Log to detailed results CSV
with open(AERO_RESULTS_FILE, mode="a", newline="") as f:
    writer = csv.writer(f)
```

```
        writer.writerow([_vspaero_call_count, cl, cd, cm, ld])

    # Update progress bar
    if _progress_bar is not None:
        _progress_bar.update(1)
```

Dual Logging System:

1. **Detailed Results (aero_results.csv):**
    - Every single VSPAERO evaluation
    - Includes all aerodynamic coefficients
    - Used for post-analysis and debugging
2. **Optimization Progress (wingopt_results.csv):**
    - Only feasible solutions
    - Tracks best solution over time
    - Compact view of optimization trajectory

9.4 Error Handling

```
try:
    for i in range(0, MAX_EVALUATIONS, STEP_SIZE):
        algorithm.run(STEP_SIZE)
        # ... logging ...

except KeyboardInterrupt:
    print("\nOptimization interrupted by user.")
finally:
    _progress_bar.close()
```

Robustness Features:

- Graceful keyboard interrupt handling
- Progress bar cleanup
- Partial results preservation
- Clear error messages

---

10. Results and Output

10.1 Output Files

10.1.1 Aerodynamic Results (aero_results.csv)

Format:

```
Iteration,Cl,Cd,Cm,L/D
1,0.2134,0.0312,-0.0145,6.84
2,0.2289,0.0298,-0.0132,7.68
...
```

Contents:

- Complete history of all evaluations
- Includes both feasible and infeasible designs
- Useful for convergence analysis

Analysis Possibilities:

- Plot Cl vs Cd (drag polar)
- Track constraint satisfaction
- Identify parameter sensitivities
- Detect numerical issues

10.1.2 Optimization Progress (wingopt_results.csv)

Format:

```
iteration,objective,semi_span,semi_area,taper,sweep,twist,Root_inc
1,-7.234156,1.124567,0.678234,0.456789,2.345678,1.234567,-0.987654
10,-7.456789,1.156789,0.689123,0.478912,2.123456,0.987654,-1.123456
...
```

Contents:

- Only feasible solutions
- Best design at each reporting interval
- Complete design variable values

Usage:

- Track optimization progress
- Identify convergence
- Extract optimal design
- Resume optimization if interrupted

10.1.3 Debug Files

modified_wing.vsp3:

- OpenVSP geometry file
- Can be opened in VSP GUI for visualization
- Last evaluated design
- Useful for verification

modified_wing.polar:

- VSPAERO output file
- Complete aerodynamic breakdown
- Last evaluated design
- Includes spanwise distributions

10.2 Terminal Output

Initialization:

```
Using VSPAERO Path: /path/to/vspaero
Starting Wing Optimization...
Design Variables: ['semi_span', 'semi_area', 'taper', 'sweep', 'twist',
'root_inc']
Running for 50 evaluations with swarm size 50...
```

Progress Bar:

```
VSPAERO Runs: 1234/2500 [=============>    ] 49% [00:15:23<00:16:12, 1.2s/run]
```

Final Summary:

```
Optimization Complete.

=== Best Feasible Solution Found ===
Objective (Negative Endurance): -8.234567
------------------------------
Semi-Span      : 1.1234 m (Full Span: 2.2468 m)
Semi-Area      : 0.6789 m^2 (Full Area: 1.3578 m^2)
Taper Ratio    : 0.4567
Sweep          : 2.3456 deg
Twist          : -1.2345 deg
Root Incidence : -0.9876 deg
```

10.3 Result Interpretation

10.3.1 Objective Function Value

Negative Endurance Parameter:

- More negative = Better endurance
- Example: -8.0 is better than -7.0
- Convert to actual endurance: Multiply by -1

Typical Values:

- Poor design: -5.0 to -6.0
- Average design: -6.0 to -7.5
- Good design: -7.5 to -9.0
- Excellent design: > -9.0

10.3.2 Optimal Design Characteristics

Expected Trends:

1. **High Aspect Ratio:**
    - Span near maximum bound
    - Area near minimum (high AR = b^2/S)
    - Reduces induced drag
2. **Moderate Taper:**
    - Not rectangular ($\lambda = 1.0$)
    - Not highly tapered ($\lambda = 0.2$)
    - Optimal typically $\lambda = 0.4$-0.6
3. **Small Positive Sweep:**
    - 1-3 degrees typical
    - Improves stability
    - Minimal drag penalty
4. **Negative Twist (Washout):**
    - Tip twisted nose-down
    - Improves stall characteristics
    - Optimizes spanwise lift distribution
5. **Small Negative Root Incidence:**
    - Reduces zero-lift drag
    - Maintains proper cruise attitude

10.3.3 Performance Metrics

From Optimal Design:

```
Cl = 0.22 (within 0.2-0.25 bounds)
Cd = 0.028 (low drag)
Cm = -0.012 (stable)
L/D = 7.86 (efficient)
```

Endurance Calculation:

```
Endurance Parameter = Cl^1.5 / Cd
                    = 0.22^1.5 / 0.028
                    = 0.1032 / 0.028
                    = 3.69

Actual endurance (hours) = η × (L/D) × (W_fuel/W_total) × (1/SFC)
```

---

11. Usage Guide

11.1 Prerequisites

Software Requirements:

1. Python 3.7 or later
2. OpenVSP 3.x with Python bindings
3. VSPAERO executable

Python Packages:

```
pip install platypus-opt
pip install tqdm
```

System Requirements:

- Operating System: Windows, Linux, or macOS
- RAM: Minimum 4 GB (8 GB recommended)
- Disk Space: 1 GB for results
- CPU: Multi-core recommended (VSPAERO uses 4 cores)

11.2 Installation

Step 1: Install OpenVSP

- Download from: http://openvsp.org/
- Install Python API bindings
- Verify installation:

```
import openvsp as vsp
print(vsp.GetVSPVersion())
```

Step 2: Clone/Download Project

```
git clone <repository_url>
cd wing-optimization
```

Step 3: Prepare Files Ensure these files are present:

- structured_vspaero_optimizer.py
- optimizer_config.json
- ClarkY.dat

11.3 Running the Optimization

Basic Execution:

```
python structured_vspaero_optimizer.py
```

Monitor Progress:

- Watch terminal for progress bar
- Check aero_results.csv for detailed logs
- Check wingopt_results.csv for best solutions

Interrupting:

- Press Ctrl+C to stop gracefully
- Partial results are saved
- Can resume by modifying MAX_EVALUATIONS

11.4 Customization

11.4.1 Modifying Aircraft Parameters

Edit `optimizer_config.json`:

```
"aircraft": {
    "weight_kg": 10.0,              // Change to your aircraft
    "wing_loading_min": 5.0,        // Adjust loading range
    "wing_loading_max": 7.0
}
```

11.4.2 Changing Flight Conditions

```
"simulation": {
    "velocity_m_s": 25.0,           // Higher cruise speed
    "aoa_deg": 2.0                  // Non-zero angle of attack
}
```

11.4.3 Adjusting Optimization

```
"optimization": {
    "swarm_size": 100,              // More particles
    "max_evaluations": 200,         // Longer optimization
    "step_size": 10                 // Less frequent logging
}
```

11.4.4 Expanding Design Space

```
"design_variables": {
    "semi_span": {
        "min": 0.5,                 // Smaller wings
        "max": 2.0                  // Larger wings
    }
}
```

11.5 Post-Processing

Extract Best Design:

```
import pandas as pd

# Load results
df = pd.read_csv('wingopt_results.csv')

# Get best (minimum objective)
best = df.loc[df['objective'].idxmin()]

print("Optimal Design:")
print(f"Span: {2*best['semi_span']:.3f} m")
print(f"Area: {2*best['semi_area']:.3f} m²")
```

Visualize Convergence:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(df['iteration'], -df['objective'])
plt.xlabel('Iteration')
plt.ylabel('Endurance Parameter')
plt.title('Optimization Convergence')
plt.grid(True)
plt.show()
```

Analyze Drag Polar:

```
aero = pd.read_csv('aero_results.csv')

plt.figure(figsize=(8, 6))
plt.scatter(aero['Cd'], aero['Cl'], alpha=0.5)
plt.xlabel('Drag Coefficient (Cd)')
plt.ylabel('Lift Coefficient (Cl)')
plt.title('Drag Polar')
plt.grid(True)
plt.show()
```

---

12. Technical Challenges and Solutions

12.1 Challenge: Verbose Output

Problem:

- OpenVSP and VSPAERO produce extensive console output
- Clutters terminal during optimization
- Makes progress tracking difficult

Solution:

```
@contextlib.contextmanager
def suppress_output_fd():
    # Redirect file descriptors to /dev/null
    # Works at OS level, captures C++ output
```

Why This Works:

- File descriptor redirection at OS level
- Captures output from compiled libraries
- Python's stdout/stderr capture insufficient

12.2 Challenge: Path Discovery

Problem:

- VSPAERO location varies by installation
- Manual path specification error-prone
- Different OS conventions (Windows vs Linux)

Solution:

```
def find_vspaero_path():
    # 1. Check config
    # 2. Check package location
```

```
    # 3. Check system PATH
    # 4. Graceful fallback
```

Benefits:

- Automatic discovery in common locations
- Override capability through config
- Cross-platform compatibility

12.3 Challenge: Mesh Quality vs Speed

Problem:

- Fine mesh: Accurate but slow
- Coarse mesh: Fast but inaccurate
- Balance needed for optimization

Solution:

```
vsp.SetParmVal(wing_id, "SectTess_U", "XSec_1", 10)
vsp.SetParmVal(wing_id, "Tess_W", "Shape", 20)
```

Selected Values:

- 10 spanwise × 20 chordwise = 200 panels
- Converged for low-Re wing analysis
- ~5-10 seconds per evaluation
- Total optimization: 3-5 hours

12.4 Challenge: Constraint Feasibility

Problem:

- Initial swarm may be entirely infeasible
- Some constraint combinations conflict
- Need to reach feasible region quickly

Solution:

1. **Wide Initial Distribution:**
    - SMPSO starts with random positions
    - Covers entire design space
2. **Penalty Method:**
    - Infeasible solutions receive poor fitness
    - Natural evolution toward feasibility
3. **Conservative Bounds:**

- Wing loading limits ensure lift feasibility
- Stability constraint not overly restrictive

12.5 Challenge: Numerical Robustness

Problem:

- VSPAERO occasionally fails to converge
- Invalid geometries cause crashes
- Need graceful error handling

Solution:

```
if polar_results:
    cl = res.get("CLtot", 0.0)
    # ... process results ...
else:
    # Return zeros (penalized by optimizer)
    return 0.0, 0.0, 0.0, 0.0
```

Robustness Features:

- Check for file existence
- Validate parsing success
- Default to zero coefficients
- Log warnings for failed evaluations

12.6 Challenge: Optimization Time

Problem:

- Each VSPAERO run: 5-10 seconds
- 50 particles × 50 iterations = 2,500 runs
- Total time: 3-7 hours

Solutions Implemented:

1. **Reduced Swarm Size:** 50 instead of 100+
2. **Efficient Mesh:** Balance quality/speed
3. **Progress Tracking:** TQDM for user feedback
4. **Incremental Saving:** Results saved continuously

Further Optimizations (Not Implemented):

- Parallel VSPAERO evaluations
- Surrogate modeling (response surface)
- Adaptive mesh refinement
- Early termination criteria

---

13. Future Enhancements

13.1 Multi-Objective Optimization

Current: Single objective (maximize endurance)

Proposed:

```
objectives = [
    endurance,      # Maximize
    weight,         # Minimize
    stability,      # Maximize
]
```

Benefits:

- Pareto front of trade-offs
- Designer can choose final design
- More realistic engineering scenarios

Implementation:

- Use NSGA-II or NSGA-III instead of SMPSO
- Modify objective function to return multiple values
- Post-process Pareto front

13.2 Additional Constraints

Structural Constraints:

```
# Maximum wing loading
g4 = wing_stress - allowable_stress

# Minimum thickness for structure
g5 = min_thickness - 0.01  # 1 cm minimum
```

Performance Constraints:

```
# Minimum cruise speed
g6 = V_stall - 15.0  # Must stall above 15 m/s

# Minimum roll rate
g7 = roll_rate - 60.0  # Min 60 deg/s
```

13.3 Variable Airfoil Sections

Current: ClarkY at both root and tip

Proposed:

```
root_airfoil = "ClarkY.dat"
tip_airfoil = "NACA0012.dat"  # Symmetric tip
```

Benefits:

- Optimized lift distribution
- Better stall characteristics
- Reduced tip vortex

13.4 Control Surface Integration

Add:

- Aileron sizing and placement
- Elevator effectiveness
- Rudder design

Impacts:

- Stability analysis
- Control authority
- Hinge moments

13.5 Multidisciplinary Optimization

Current: Pure aerodynamics

Expand to:

1. **Structures:**
    - Wing weight estimation
    - Stress analysis
    - Flutter analysis
2. **Propulsion:**
    - Thrust-drag matching
    - Power-required curves
    - Propeller integration
3. **Performance:**
    - Mission analysis
    - Range/endurance trade-offs
    - Takeoff/landing distances
4. **Cost:**

- Manufacturing complexity
- Material costs
- Operational economics

13.6 Uncertainty Quantification

Robust Design:

```
# Evaluate at multiple conditions
for delta_rho in [-10%, 0%, +10%]:
    for delta_V in [-5%, 0%, +5%]:
        run_analysis(rho + delta_rho, V + delta_V)

# Minimize variance of performance
```

Benefits:

- Designs less sensitive to variations
- Better real-world performance
- Accounts for manufacturing tolerances

13.7 GUI Development

Proposed Features:

- Interactive parameter adjustment
- Real-time 3D visualization
- Progress monitoring
- Result plotting

Technologies:

- PyQt or Tkinter for interface
- Matplotlib for plots
- VTK or ParaView for 3D rendering

13.8 Parallel Computing

Current: Sequential evaluations

Proposed:

```
from multiprocessing import Pool

with Pool(processes=8) as pool:
    results = pool.map(evaluate_design, design_population)
```

Expected Speedup:

- 8-core system: 6-7× faster
- 2,500 runs: 30-60 minutes instead of 3-5 hours

Challenges:

- OpenVSP thread safety
- File I/O conflicts
- Memory management

13.9 Surrogate Modeling

Approach:

1. Run 500 high-fidelity VSPAERO simulations
2. Train surrogate model (Kriging, RBF, Neural Network)
3. Use surrogate for optimization (1000× faster)
4. Validate final design with VSPAERO

Benefits:

- Explore much larger design space
- Enable real-time optimization
- Sensitivity analysis becomes trivial

Drawbacks:

- Requires upfront sampling
- Accuracy depends on training data
- May miss local optima

---

14. Lessons Learned

14.1 Technical Insights

1. **Endurance vs Range:**
    - Endurance metric (Cl^1.5/Cd) different from range (Cl/Cd)
    - Optimal designs differ significantly
    - Mission requirements dictate objective function
2. **Constraint Activity:**
    - Lift constraints typically active at optimum
    - Stability constraint often inactive (easy to satisfy)
    - Wing loading bounds directly shape feasible region
3. **Design Space Topology:**
    - Multiple local optima exist
    - Particle swarm effective at global search
    - Starting conditions matter less than expected

4. **Computational Efficiency:**
    - VSPAERO panel method fast enough for optimization
    - Mesh resolution converged at moderate density
    - Overhead (geometry, I/O) significant fraction of runtime

14.2 Software Engineering

1. **Configuration Management:**
    - JSON config file greatly improves usability
    - Separates parameters from code
    - Enables reproducibility and version control
2. **Progress Tracking:**
    - Essential for long-running optimizations
    - TQDM excellent for terminal-based feedback
    - Incremental file saving prevents data loss
3. **Error Handling:**
    - Graceful degradation better than crashes
    - Zero-value returns allow optimizer to recover
    - Logging helps diagnose issues
4. **Code Structure:**
    - Clear separation of concerns improves maintainability
    - Utility functions reduce code duplication
    - Comments and docstrings essential for collaboration

14.3 Optimization Strategy

1. **Swarm Size:**
    - 50 particles adequate for 6 variables
    - Larger swarms improve exploration but cost time
    - Diminishing returns above population = 10 × n_vars
2. **Evaluation Budget:**
    - 50 iterations not always sufficient for convergence
    - Monitor progress to determine stopping
    - Can restart from best-so-far solution
3. **Constraint Handling:**
    - Penalty method works well for this problem
    - Feasible region relatively large
    - Multiple constraints provide good guidance

4. **Objective Function:**
    - Single objective simplifies analysis
    - Multi-objective would reveal trade-offs
    - Consider Pareto optimization for production

---

15. Conclusion

15.1 Project Summary

This internship project successfully developed a comprehensive wing optimization system that integrates:

- **Parametric CAD modeling** through OpenVSP
- **CFD analysis** using VSPAERO panel method
- **Meta-heuristic optimization** via particle swarm algorithm
- **Automated workflow** with Python scripting

The system demonstrates a complete engineering design cycle from problem formulation through numerical optimization to result analysis.

15.2 Key Achievements

1. **Functional Optimization Framework:**
    - Automated generation of wing geometries
    - Batch aerodynamic analysis
    - Constraint enforcement
    - Result tracking and reporting
2. **Robust Implementation:**
    - Cross-platform compatibility
    - Error handling and recovery
    - Progress monitoring
    - Configuration management
3. **Engineering Validation:**
    - Physical constraints properly enforced
    - Aerodynamic coefficients in expected ranges
    - Convergence to sensible designs
    - Reproducible results
4. **Documentation:**
    - Comprehensive technical documentation
    - Mathematical formulation
    - Implementation details
    - Usage instructions

15.3 Technical Skills Developed

Programming:

- Python for scientific computing
- API integration (OpenVSP)
- File I/O and data management
- Object-oriented design

Aerodynamics:

- Wing geometry parameterization
- Coefficient interpretation
- Performance analysis
- Stability considerations

Optimization:

- Problem formulation
- Constraint handling
- Algorithm selection and tuning
- Convergence analysis

Software Engineering:

- Configuration management
- Error handling
- Progress tracking
- Code documentation

15.4 Real-World Applications

This optimization framework can be applied to:

1. **UAV Design:**
    - Surveillance drones
    - Agricultural monitoring
    - Package delivery systems
2. **Model Aircraft:**
    - Competition gliders
    - RC aircraft
    - Scale models
3. **Research:**
    - Parametric studies
    - Sensitivity analysis
    - Technology assessment
4. **Education:**

- Demonstration tool
- Student projects
- Concept validation

15.5 Impact and Value

For the Organization:

- Reduced design cycle time (manual → automated)
- Improved design quality (optimized vs. intuition)
- Reusable framework for future projects
- Documentation enables knowledge transfer

For the Intern:

- Practical experience with industry tools
- Integration of multiple disciplines
- Problem-solving and debugging skills
- Technical communication and documentation

15.6 Recommendations

Immediate Next Steps:

1. Validate optimal design with wind tunnel testing
2. Perform sensitivity analysis on key parameters
3. Extend to multi-objective optimization
4. Implement parallel computing for speedup

Long-Term Development:

1. Integrate with structural analysis tools
2. Add manufacturing constraints
3. Develop graphical user interface
4. Expand to complete aircraft optimization

Best Practices:

1. Always version control configuration files
2. Archive all optimization runs with timestamps
3. Validate results with simplified analytical models
4. Document assumptions and limitations

---

16. Appendices

16.1 Glossary of Terms

Aerodynamic Terms:

- **Angle of Attack (α):** Angle between chord line and freestream

- **Aspect Ratio (AR):** Span squared divided by area (b^2/S)
- **Camber:** Curvature of airfoil mean line
- **Chord:** Distance from leading to trailing edge
- **Induced Drag:** Drag due to lift generation
- **MAC:** Mean Aerodynamic Chord
- **Parasite Drag:** Non-lift-related drag (friction, form)
- **Pitching Moment:** Moment causing nose-up or nose-down rotation
- **Reynolds Number:** Ratio of inertial to viscous forces
- **Stall:** Loss of lift due to flow separation
- **Sweep:** Angle of leading edge from spanwise direction
- **Taper Ratio:** Tip chord divided by root chord
- **Twist:** Change in airfoil angle from root to tip
- **Wing Loading:** Weight divided by wing area

Optimization Terms:

- **Constraint:** Inequality or equality that must be satisfied
- **Convergence:** Approaching stable solution
- **Design Variable:** Parameter that optimizer can change
- **Feasible Solution:** Design satisfying all constraints
- **Objective Function:** Quantity to minimize or maximize
- **Pareto Front:** Set of non-dominated solutions
- **Particle:** Candidate solution in swarm
- **Swarm:** Population of particles

Software Terms:

- **API:** Application Programming Interface
- **CFD:** Computational Fluid Dynamics
- **CSV:** Comma-Separated Values file format
- **GUI:** Graphical User Interface
- **JSON:** JavaScript Object Notation
- **Panel Method:** Surface discretization for potential flow
- **VSP:** Vehicle Sketch Pad (OpenVSP)

16.2 Mathematical Notation

Symbol	Meaning	Units
ρ	Air density	kg/m^3
V	Velocity	m/s
μ	Dynamic viscosity	Pa·s
Re	Reynolds number	-
M	Mach number	-
α	Angle of attack	deg

Symbol	Meaning	Units
β	Sideslip angle	deg
S	Wing area	m^2
b	Wing span	m
c	Chord length	m
AR	Aspect ratio	-
λ	Taper ratio	-
Λ	Sweep angle	deg
ε	Twist angle	deg
Cl	Lift coefficient	-
Cd	Drag coefficient	-
Cm	Moment coefficient	-
L	Lift force	N
D	Drag force	N
M	Moment	N·m
W	Weight	N

16.3 File Format Specifications

ClarkY.dat Format:

```
AIRFOIL_NAME
x1/c  y1/c
x2/c  y2/c
...
```

- First line: Airfoil identifier (optional)
- Subsequent lines: x/c (chordwise), y/c (vertical)
- Points ordered from TE → LE → TE
- Normalized by chord length

optimizer_config.json Format:

```
{
  "environment": {...},
  "aircraft": {...},
  "simulation": {...},
  "design_variables": {...},
```

```
  "optimization": {...}
}
```

- Standard JSON syntax
- Nested dictionaries for organization
- Comments not supported (use external documentation)

aero_results.csv Format:

```
Iteration,Cl,Cd,Cm,L/D
1,0.2134,0.0312,-0.0145,6.84
```

- Header row with column names
- One row per VSPAERO evaluation
- Comma-separated values

wingopt_results.csv Format:

```
iteration,objective,semi_span,semi_area,taper,sweep,twist,Root_inc
1,-7.234,1.124,0.678,0.456,2.345,1.234,-0.987
```

- Header row with column names
- One row per optimization step
- Only feasible solutions recorded

16.4 References

Books:

1. Anderson, J. D. (2017). *Fundamentals of Aerodynamics* (6th ed.). McGraw-Hill.
2. Raymer, D. P. (2018). *Aircraft Design: A Conceptual Approach* (6th ed.). AIAA.
3. Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley.
4. Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer.

Papers:

1. Kennedy, J., & Eberhart, R. (1995). "Particle swarm optimization." *IEEE Conference on Neural Networks*.
2. Nebro, A. J., et al. (2009). "SMPSO: A new PSO-based metaheuristic for multi-objective optimization." *IEEE Symposium on Computational Intelligence*.

Software Documentation:

1. OpenVSP User Manual: https://openvsp.org/pyapi_docs/latest/
2. OpenVSP google group: https://groups.google.com/g/openvsp
3. Platypus Documentation: https://platypus.readthedocs.io/

16.5 Contact Information

Project Details:

- Project Title: VSPAERO Wing Optimization System
- Duration: 10th June 2025 to 31st Dec 2025
- Organization: Indiflo Pvt. Ltd.
- Department: Mechanical sub-division

Personal Info:

- Name : Shubham Tamboli
- Linkedin : https://www.linkedin.com/in/shubham-t-816785201/
- 3rd Year Btech. Mechanical Engineering , IIT Jodhpur (At the time of internship conclusion).

END OF DOCUMENTATION