Mississippi State Formula SAE

# Radiator and Sidepod System Design

Cooling Capacity, CFD Porous Media, and Sidepod Geometry

Jacob Bosarge
September 1, 2025

| | |
|---|---|
| Document Version: | v2.0 |
| Contact: | jmb2165@msstate.edu |

# Introduction

This report consolidates the cooling and packaging workflow for the Formula SAE car into three tightly linked parts. Together they convert track-derived operating data into a defensible heat-rejection assessment, translate radiator geometry into CFD-ready porous-media coefficients, and size the sidepod around a prescribed radiator center section. The result is a consistent methodology to take measured vehicle data and radiator inputs. Calculations are ran to produce information to feed design decisions that can be take advantage of completely characterized CFD, and high fidelity heat transfer calculations.

I. **Cooling-capacity pass/fail.** Evaluates radiator heat-rejection capacity across measured operating points and flags pass/fail with margin.

II. **CFD porous-media coefficients.** Derives Darcy–Forchheimer porous-media coefficients for use in CFD – mapping geometry to pressure-loss models.

III. **Sidepod geometry and taper angles.** Generates sidepod geometry around a predetermined radiator cross section – reports inlet/outlet frames, taper lengths, and taper half-angles. Outputs are used to generate a SolidWorks model of the sidepod.

## 0.1 Workflow

The idealized workflow should go as follows:

1. Generate sidepod geometry.

2. Define sidepod and radiator in CFD with correct porous media characteristics.

3. Collect velocity gradient at the face of the radiator – done in enough segments to get high fidelity heat transfer simulation.

4. Input velocity gradient into heat transfer script and analyze `Pass/Fail` conditions.

Part I

# Cooling Capacity Pass/Fail Analysis

## Purpose and Scope

The script evaluates whether a radiator at predefined dimensions and an externally simulated airflow can reject the required engine heat at each measured operating point (binned by intake manifold air pressure, MAP, and engine speed measured in RPM). For each point, it compares the radiator's calculated heat rejection $Q_{\text{total}}$, as a function of the radiator and airflow, against a required coolant heat load $Q_{\text{req}}$ derived from the engines fuel consumption at competition. Outputs include:

- A per-point `Pass/Fail` flag and a percent margin $(Q_{\text{total}} - Q_{\text{req}})/Q_{\text{req}}$.
- Gear-dependent vehicle speeds computed from engine RPM and drivetrian gear ratios.
- Console tables, a CSV of results, and multiple 3D plots (to have the data graphically represented).

## High-Level Data and Flow

1. **Data Ingest & Validation.** The script reads a CSV containing manipulated endurance and autocross data from the 2025 Competition.
2. **User Inputs.** Geometric parameters, material properties, and fluid properties values of the system (radiator core geometry, fin parameters, air and coolant properties, sidepod areas, mechanical/thermal factors, etc.) are all inputted. Airflow measurements are inputted via a velocity gradient gathered from CFD ran on the system externally. CFD methodology is discussed later in the document. The velocity gradient is inputted as a number of segments ($N_{\text{seg}} = N$), each of these segments is divided into an equal length section of the radiator core vertically. Each segment is computed as its own heat exchanger using the Effectiveness NTU method to determine the outlet temperature, which is used at the next segments inlet temperature. The radiator is assumed to not vary laterally.
3. **Geometry & Area Calculations.** Internal coolant-side area, external air-side area (with fin efficiency), hydraulic diameters, and per-segment partitions are computed based on physically measured radiator dimensions.
4. **Thermophysical Properties & Nondimensional Groups.** Properties are treated as constants (coolant: $\rho_c = 59.8$ lbm/ft$^3$, $\mu_c = 2.2 \times 10^{-4}$ lbm/(ft $\cdot$ s), $k_c = 0.37$ Btu/(hr $\cdot$ ft $\cdot ^\circ$ F), $c_{p,c} = 1.00$ Btu/(lbm $\cdot ^\circ$ F); air: $\rho_a = 0.0749$ lbm/ft$^3$, $\mu_a = 1.2 \times 10^{-5}$ lbm/(ft $\cdot$ s), $k_a = 0.0137$ Btu/(hr $\cdot$ ft $\cdot ^\circ$ F), $c_{p,a} = 0.24$ Btu/(lbm $\cdot ^\circ$ F)). Prandtl numbers are formed in English units as $\text{Pr} = c_p(\mu \cdot 3600)/k$ to be consistent with hour-based conductance. Reynolds numbers use hydraulic diameters $D_{h,c}$ (tube interior) and $D_{h,a} = h_{\text{tube}} + 2t_{\text{wall}}$ (outer height on the air side), with velocities from the measured coolant CFM and the prescribed air profile. Film coefficients follow standard correlations: coolant-side Dittus–Boelter $\text{Nu}_c = 0.023\,\text{Re}_c^{0.8}\text{Pr}_c^{0.4}$ and $h_c = \text{Nu}_c k_c / D_{h,c}$; air-side a louvered-fin Manglik–Bergles-style $j$-factor with $j = 0.6522\,\text{Re}_a^{-0.5403}\left(1 + 5.269 \times 10^{-5}\text{Re}_a^{1.340}\right)\text{Pr}_a^{1/3}\left(1 + 0.504\,\text{Pr}_a^{-2/3}\right)$, $\text{Nu}_a = j\,\text{Re}_a\,\text{Pr}_a^{1/3}$, and $h_a = \text{Nu}_a k_a / D_{h,a}$.

5. **Segment heat transfer.** The thermal conductance, $UA$, is calculated for each segment; crossflow $\varepsilon$ (effectiveness) is evaluated as a function of NTU and $C_r$. Segment heat $q_{\text{seg}}$ and coolant outlet temperature are calculated for each segment. As said earlier, the calculated outlet temperature per segment is used as the next segments inlet temperature. At the end of the algorithm, all of the segment heat rejections are summed $\sum q_{\text{seg}}$.

6. **Decision metric.** Summed $Q_{\text{total}} = \sum q_{\text{seg}}$ is compared to $Q_{\text{req}}$ to set `Pass/Fail` and margin.

7. **Vehicle speed mapping.** For each gear, vehicle speed is computed from RPM, total gear ratio, and tire circumference; these speeds annotate each operating point and enable speed-based visualizations compared to the free stream velocity of the vehicle.

## Assumptions and Units

- Steady, one-dimensional segment model with series thermal resistances; crossflow, both fluids unmixed, via an empirical $\varepsilon$–NTU correlation.
- **Units:** English engineering units are used consistently (e.g., ft, s, hr, Btu, lbm). The horsepower-to-heat conversion 1 hp = 2545 Btu/hr is applied.
- Air and coolant properties are fixed constants at representative conditions. There are plans to improve this in the script.
- Coolant volumetric flow vs. RPM is piecewise linear (extrapolated below/above the tabulated range, linearly interpolated within). The tabulated values were measured experimentally from the engine.

## How Autocross and Endurance Data Were Obtained

Brake horsepower (BHP) values were calculated from datalogs collected during autocross and endurance events. Each log included time-resolved engine parameters such as RPM, MAP, injector pulse width, AFR (via $O_2$ lambda sensors), fuel pressure, and air temperature.

### Brake Horsepower Estimation

BHP was estimated by computing the injected fuel mass per engine cycle, correcting the air–fuel ratio based on $O_2$ sensor time delay, and applying a thermal efficiency factor to determine mechanical output. The following equation was used:

$$\text{BHP} = \frac{\dot{m}_{\text{fuel}} \cdot \text{AFR} \cdot \eta_{\text{th}} \cdot \text{LHV} \cdot N}{2545}, \tag{1}$$

$$N = 30 \cdot \text{RPM}, \quad \text{(single-cylinder, 4-stroke cycles per minute)} \tag{2}$$

where:

- $\dot{m}_{\text{fuel}}$ is the per-cycle fuel mass derived from injector characteristics and pulse width,
- AFR is corrected by interpolating based on modeled exhaust travel time,

- $\eta_{\text{th}}$ is the thermal efficiency and is assumed to be 0.30 based on the combustion engine "rule of thirds",
- LHV is the lower heating value of the fuel (20,600 $\text{Btu}\,\text{lb}^{-1}$),
- $N$ is the number of engine cycles per hour,
- 2545 is the Btu-to-horsepower conversion factor.

**Notes BHP** Raw brake horsepower data was calculated for every single operating point from the datalogs initially. The following graph is a 3D plot of the raw VE data, which is directly proportional to BHP.

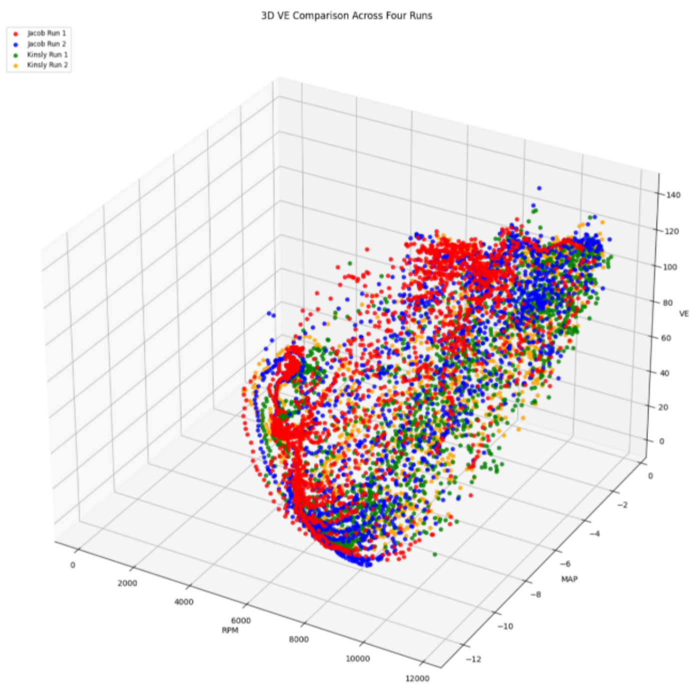

Figure 1: The Raw Unfiltered VE data from the autocross datalogs

## Data Aggregation and Interpretation

To produce a representative performance map, BHP data from five logged sessions were merged and analyzed as follows:

1. **Binning:** RPM was divided into 500 $\text{rev}\,\text{min}^{-1}$ bins (from 3000 to 12,000 RPM), and MAP was divided into 1 psi bins (from $-13$ to $+1$ psig). Each recorded data point was assigned to the corresponding bin.
2. **Averaging:** Within each RPM–MAP bin, the average BHP was calculated to estimate engine output in that operating region.
3. **Weighting by frequency:** Each data point represents a discrete time interval; bins with higher counts represent regions where the engine operated more frequently or for longer

durations. These counts were later used as statistical weights in computing aggregate metrics (e.g., mean, median). The ”count” of these data points are also displayed in the results of the `Pass/Fail` script as a way to graphically view the significance of each failure point in terms of operational likelihood.

4. **Filtering:** Bins with fewer than 10 data points were excluded to avoid sparsely sampled or non-representative regions.

5. **Interpretation:** The resulting two-dimensional dataset reflects both the magnitude of power output across engine conditions and the prevalence of those conditions. This enables weighted statistical analysis and supports performance and cooling-system optimization.

## Equations Used (step-by-step)

All symbols and variables can be referenced in Appendix E for clarification on definition and units.

### Data-driven coolant flow

Let tabulated coolant volumetric flows be ($\mathrm{RPM}_i$, $\dot{V}_{\mathrm{coolant},\,i}$). Define end slopes for extrapolation outside of dataset:

$$s_{\mathrm{lo}} = \frac{\dot{V}_{\mathrm{coolant},\,2} - \dot{V}_{\mathrm{coolant},\,1}}{\mathrm{RPM}_2 - \mathrm{RPM}_1}, \qquad s_{\mathrm{hi}} = \frac{\dot{V}_{\mathrm{coolant},\,N} - \dot{V}_{\mathrm{coolant},\,N-1}}{\mathrm{RPM}_N - \mathrm{RPM}_{N-1}}. \tag{3}$$

To calculate coolant flow at a queried RPM:

$$\dot{V}_{\mathrm{coolant}}(\mathrm{RPM}) = \begin{cases} \dot{V}_{\mathrm{coolant},\,1} + s_{\mathrm{lo}}\big(\mathrm{RPM} - \mathrm{RPM}_1\big), & \mathrm{RPM} < \mathrm{RPM}_1, \\ \text{linear interpolation between bracketing points}, & \mathrm{RPM}_1 \le \mathrm{RPM} \le \mathrm{RPM}_N, \\ \dot{V}_{\mathrm{coolant},\,N} + s_{\mathrm{hi}}\big(\mathrm{RPM} - \mathrm{RPM}_N\big), & \mathrm{RPM} > \mathrm{RPM}_N. \end{cases} \tag{4}$$

### Core geometry and areas

Calculate frontal area, air flow area, coolant side hydraulic diameter, and coolant flow area:

$$A_{\mathrm{front}} = A_{\mathrm{core,face}}\,\sin(\theta_{\mathrm{rad}}), \tag{5}$$

$$A_{\mathrm{flow}} = n_{\mathrm{tube}}\,w_{\mathrm{tube}}\,h_{\mathrm{tube}}, \tag{6}$$

$$D_{h,c} = \frac{4(w_{\mathrm{tube}}h_{\mathrm{tube}})}{2(h_{\mathrm{tube}} + w_{\mathrm{tube}})}, \tag{7}$$

$$A_{c,\mathrm{tot}} = n_{\mathrm{tube}}\,n_{\mathrm{rows}}\,\big[2(h_{\mathrm{tube}} + w_{\mathrm{tube}})\,L_{\mathrm{tube}}\big]. \tag{8}$$

Air-side augmented area with fin efficiency $\eta_{\mathrm{fin}}$:

$$\mathrm{FPI} = \frac{1}{12\,d_{\mathrm{fin}}}, \qquad N_{\mathrm{fins}} \approx \mathrm{round}\big(\mathrm{FPI}\cdot 12\,W_{\mathrm{core}}\big), \tag{9}$$

$$W_{\mathrm{fin}} = n_{\mathrm{rows}}(h_{\mathrm{tube}} + 2t_{\mathrm{wall}} + t_{\mathrm{fin}})\,, \tag{10}$$

$$A_{\mathrm{fin,tot}} = 2\,N_{\mathrm{fins}}\,H_{\mathrm{core}}\,W_{\mathrm{fin}}, \tag{11}$$

$$A_{\mathrm{bare}} = n_{\mathrm{tube}}\,n_{\mathrm{rows}}\,\big[2(h_{\mathrm{tube,o}} + w_{\mathrm{tube,o}})\,L_{\mathrm{tube}}\big], \tag{12}$$

$$A_{a,\mathrm{tot}} = \eta_{\mathrm{fin}}A_{\mathrm{fin,tot}} + A_{\mathrm{bare}}. \tag{13}$$

Per-segment area division (with $N_{\text{seg}}$ segments):

$$A_{\text{front,seg}} = \frac{A_{\text{front}}}{N_{\text{seg}}}, \quad A_{a,\text{seg}} = \frac{A_{a,\text{tot}}}{N_{\text{seg}}}, \quad A_{c,\text{seg}} = \frac{A_{c,\text{tot}}}{N_{\text{seg}}}. \tag{14}$$

### Fluid properties and nondimensional groups

$$\mu_{\text{hr}} = 3600\,\mu \quad (\text{convert [s]} \to \text{[hr]}), \tag{15}$$

$$\text{Pr} = \frac{c_p\,\mu_{\text{hr}}}{k}, \tag{16}$$

$$\text{Re} = \frac{\rho\,v\,D_h}{\mu}, \tag{17}$$

$$\text{Nu} = \frac{h\,D_h}{k}. \tag{18}$$

### Coolant-side convection (Dittus–Boelter form)

Due to turbulent flows and smooth internal surfaces the Dittus–Boelter correlation was chosen:

$$\text{Nu}_c = 0.023\,\text{Re}_c^{0.8}\,\text{Pr}_c^{0.4}, \tag{19}$$

$$h_c = \frac{\text{Nu}_c\,k_c}{D_{h,c}}. \tag{20}$$

Coolant internal velocity and mass flow:

$$v_c = \frac{\dot{\text{V}}_{\text{coolant}}/60}{A_{\text{flow}}}, \tag{21}$$

$$\dot{m}_c = \rho_c\,v_c\,A_{\text{flow}} \cdot 3600, \tag{22}$$

$$C_c = \dot{m}_c\,c_{p,c}. \tag{23}$$

### Air-side convection via $j$-factor correlation (louvered/fin surfaces)

The Manglik–Bergles $j$-factor coefficient that uses an empirically defined dataset to account for the fact that the heat transfer on the air side is primarily a function of the louver fin geometry. Colburn $j$-factor correlations of the Manglik–Bergles equation collapse the geometric effects into an empirical $j(\text{Re}, \text{Pr})$ that maps directly to $\text{Nu}_a = j\,\text{Re}_a\,\text{Pr}_a^{1/3}$ and thus to $h_a$: It is a common method used in many automotive radiator research papers:

$$j = 0.6522\,\text{Re}_a^{-0.5403}\left(1 + 5.269 \times 10^{-5}\,\text{Re}_a^{1.340}\right)\text{Pr}_a^{1/3}\left(1 + 0.504\,\text{Pr}_a^{-2/3}\right), \tag{24}$$

$$\text{Nu}_a = j\,\text{Re}_a\,\text{Pr}_a^{1/3}, \qquad h_a = \frac{\text{Nu}_a\,k_a}{D_{h,\text{air}}}. \tag{25}$$

Air mass flow and heat capacity rate per segment:

$$\dot{m}_a = \rho_a\,v_a\,A_{\text{front,seg}} \cdot 3600, \tag{26}$$

$$C_a = \dot{m}_a\,c_{p,a}. \tag{27}$$

## Thermal Resistances and UA per segment

Find segment UA with thermal resistance terms:

$$R_{\text{air}} = \frac{1}{h_a\, A_{a,\text{seg}}}, \qquad R_{\text{wall}} = \frac{t_{\text{wall}}}{k_w\, A_{w,\text{seg}}}, \qquad R_{\text{cool}} = \frac{1}{h_c\, A_{c,\text{seg}}}, \tag{28}$$

$$UA_{\text{seg}} = \frac{1}{R_{\text{air}} + R_{\text{wall}} + R_{\text{cool}}}. \tag{29}$$

## Effectiveness–NTU (crossflow, both fluids unmixed)

Define $C_{\min} = \min(C_c, C_a)$, $C_{\max} = \max(C_c, C_a)$, $C_r = C_{\min}/C_{\max}$,

$$\text{NTU} = \frac{UA_{\text{seg}}}{C_{\min}}. \tag{30}$$

Effectiveness correlation:

$$\varepsilon = \begin{cases} 1 - e^{-\text{NTU}}, & C_r = 0, \\ 1 - \exp\!\Big(-\dfrac{1}{C_r}\big(1 - e^{-C_r\,\text{NTU}^{0.78}}\big)\,\text{NTU}^{0.22}\Big), & C_r > 0. \end{cases} \tag{31}$$

Segment heat transfer and coolant temperature calculation:

$$q_{\text{seg}} = \varepsilon\, C_{\min}\,\big(T_{c,\text{in}} - T_a\big), \tag{32}$$

$$T_{c,\text{out}} = T_{c,\text{in}} - \frac{q_{\text{seg}}}{C_c}. \tag{33}$$

Total across segments:

$$Q_{\text{total}} = \sum_{\text{segments}} q_{\text{seg}}. \tag{34}$$

## Required heat rejection & pass/fail margin

$$Q_{\text{req}} = \left(\frac{\text{BHP}}{\eta_{\text{mech}}}\right) f_{\text{thermal}} \times 2545 \quad [\text{Btu hr}^{-1}], \tag{35}$$

$$\text{Margin}\,[\%] = 100 \times \left(\frac{Q_{\text{total}} - Q_{\text{req}}}{Q_{\text{req}}}\right). \tag{36}$$

## Vehicle speed mapping (per gear)

Let $r_{\text{tot}} = r_{\text{gear}} \cdot r_{\text{primary}} \cdot r_{\text{final}}$ and tire circumference $C_{\text{tire}} = \pi D_{\text{tire}}$.

$$\text{wheel rps} = \frac{\text{RPM}/60}{r_{\text{tot}}}, \tag{37}$$

$$v\ [\text{ft/s}] = \text{wheel rps} \cdot C_{\text{tire}}. \tag{38}$$

Rearrangement vehicle speed reference plane (RPM at target free stream $V_\infty$):

$$\text{RPM}_* = \frac{V_\infty\, r_{\text{tot}}\, 60}{C_{\text{tire}}}. \tag{39}$$

## Algorithmic Steps

For each row of the input table $(\text{MAP\_bin}, \text{RPM\_bin}, \text{average\_BHP}, \text{count})$:

1. Compute $\dot{\text{V}}_{\text{coolant}}(\text{RPM})$ by interpolation/extrapolation; form coolant velocity $v_c$, $\text{Re}_c$, $\text{Nu}_c$, $h_c$; then $\dot{m}_c$ and $C_c$.

2. For each segment: with prescribed $v_a$, compute air-side $\text{Re}_a$, $j$, $\text{Nu}_a$, $h_a$; assemble series resistances and $UA_{\text{seg}}$.

3. Form $C_{\min}$, $C_r$, NTU, effectiveness $\varepsilon$; compute $q_{\text{seg}}$ and update $T_{c,\text{in}}$.

4. Sum $Q_{\text{total}}$ and compute $Q_{\text{req}}$ from BHP; evaluate margin and `Pass/Fail`.

5. For each gear, compute vehicle speed and attach as additional columns.

## Interpretation of Example Results

### Graphical Representation: 3D Scatter Plots

The following 3D scatter plots display the system's pass and fail conditions. Points are colored by cooling pass/fail (green/red); if the point fails, it means the cooling system is not rejecting enough heat meaning the engine will rise in temperature. For the charts with a count axis, it is used to determine the amount of time the vehicle spends operating in that zone. The charts do not have to be all green to mean the system is effective, just any failure cases need to be sparing. The last two plots account for every operating point the vehicle will encounter independent of gear, the margin, RPM, and MAP plot shows where the system fails on based on engine load, and the density, RPM, and margin plot shows where the system fails based on operational frequency. Past this, the gear plots can be used to see if these failure points occur in driving conditions that are actually possible. The gear plots display {RPM, MAP, Count} for bins whose *estimated vehicle speed* in a given gear falls within 20% of the entered free stream air velocity – this margin of error is to account for the acceleration and deceleration of air into the sidepods. This is useful for considering that there may be points that do fail, but considering the speed the vehicle is actually traveling, those failure conditions will never be reached, even if at other speeds the car spend lots of time in those conditions. This roundabout method is mainly due to the lack of measured vehicle speed and gear at competition, so it has to be accounted by using engineering judgement for these failure cases.

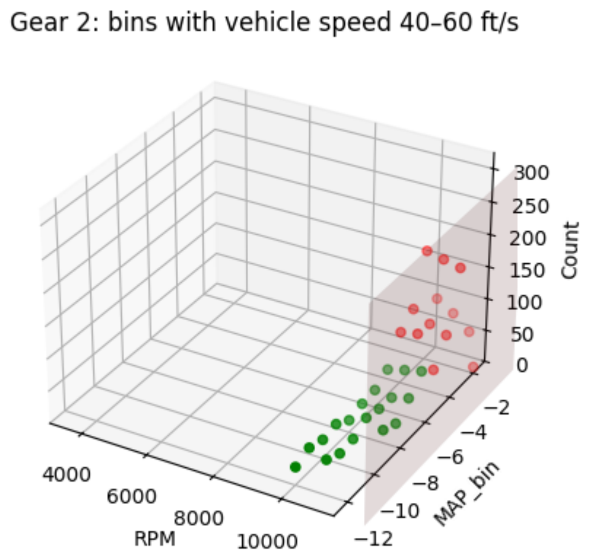

Figure 2: Gear 2: bins with vehicle speed 40–60 ft/s (example).

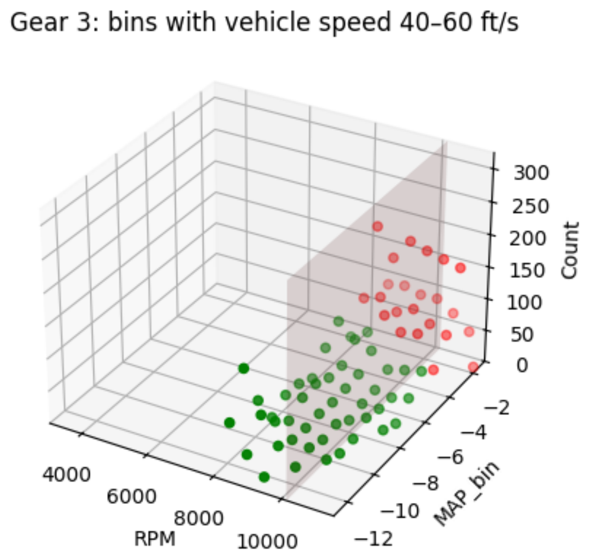

Figure 3: Gear 3: bins with vehicle speed 40–60 ft/s (example).

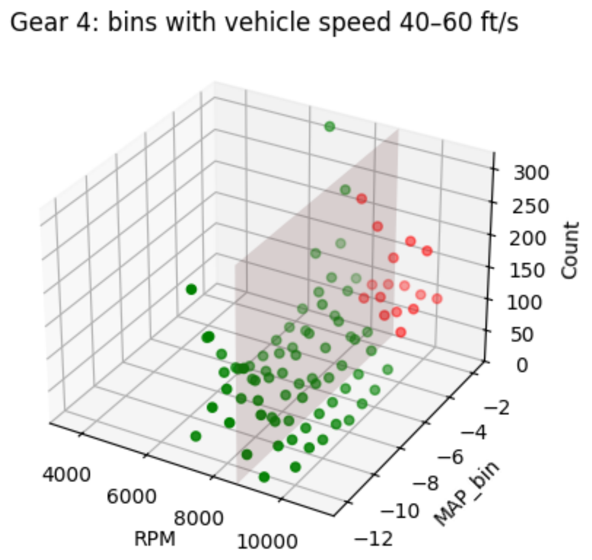

Figure 4: Gear 4: bins with vehicle speed 40–60 ft/s (example).

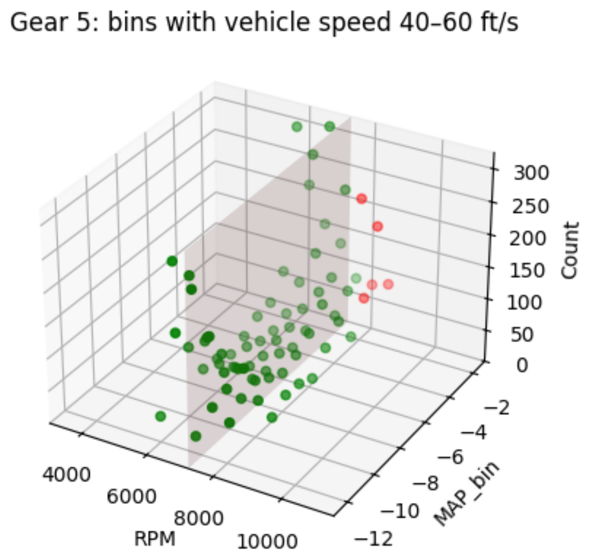

Figure 5: Gear 5: bins with vehicle speed 40–60 ft/s (example).

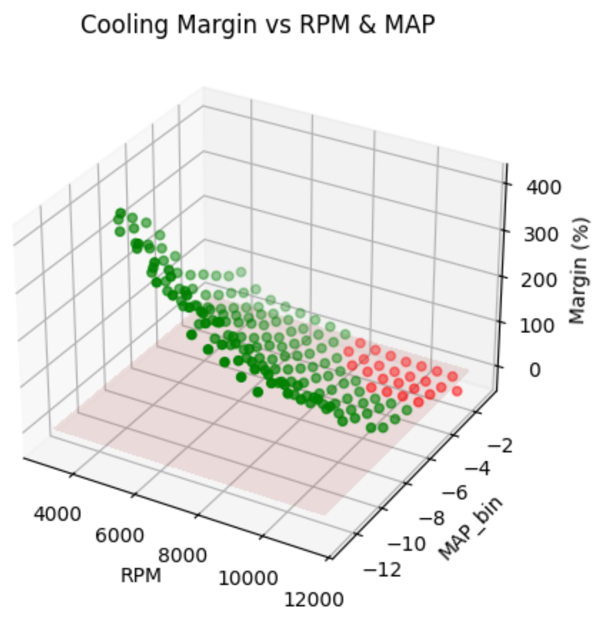

Figure 6: Cooling margin vs. RPM and MAP (example).

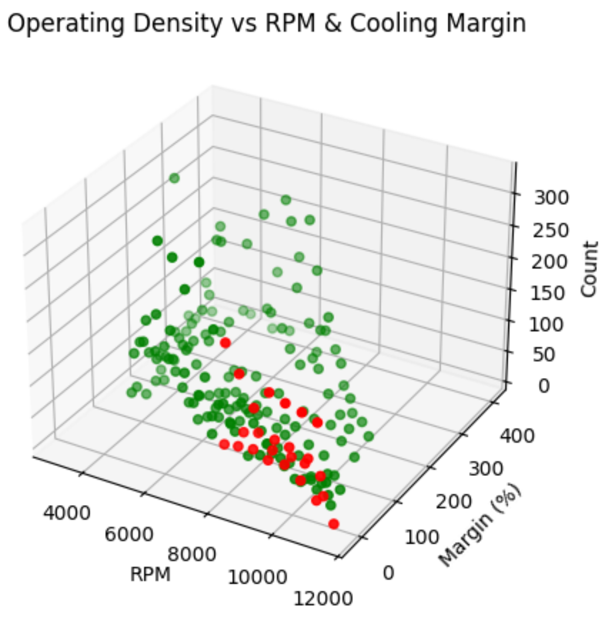

Figure 7: Operating density vs. margin and RPM (example).

### Results Table (CSV Excerpt)

The table below reproduces a subset of the CSV output for context. All rows shown are `Pass` with large positive margins; speeds increase monotonically with gear as expected.

Table 1: Example CSV excerpt of binned operating points and predicted cooling performance.

| MAP_bin | RPM_bin | Count | Result | Margin_pct | Speed_g1 | Speed_g2 | Speed_g3 |
|---:|---:|---:|---:|---:|---:|---:|---:|
| -13 | 7500 | 54 | Pass | 294.4421243 | 24.64994236 | 30.33839059 | 36.97491354 |
| -13 | 8000 | 106 | Pass | 274.0029582 | 26.29327185 | 32.36094997 | 39.43990777 |
| -13 | 8500 | 91 | Pass | 257.9369476 | 27.93660134 | 34.38350934 | 41.90490201 |
| -13 | 9000 | 51 | Pass | 237.0438128 | 29.57993083 | 36.40606871 | 44.36989624 |
| -13 | 9500 | 25 | Pass | 216.2565096 | 31.22326032 | 38.42862809 | 46.83489048 |
| -12 | 6000 | 40 | Pass | 349.1887568 | 19.71995389 | 24.27071248 | 29.57993083 |
| -12 | 6500 | 175 | Pass | 332.0366502 | 21.36328338 | 26.29327185 | 32.04492507 |

**Interpretation.** These bins are at relatively high vacuum (MAP $-12$ to $-13$ psig) and moderate-to-high RPM, hence lower load than near-atmospheric MAP and consequently very large margins. The monotone increase in speed from Gear 1 to Gear 5 at fixed RPM confirms the gearing ratios embedded in the model and provides a useful check on the CSV generation.

## Outputs and Diagnostics

- Console table of $\{\text{MAP}, \text{RPM}, \text{Count}, \text{Result}, \text{Margin}\}$.
- CSV with base columns plus gear-specific speed columns.
- 3D scatter plots: (i) $\text{RPM} \times \text{MAP} \times \text{Margin}$, (ii) $\text{RPM} \times \text{Margin} \times \text{Count}$, and (iii) gear-filtered subsets within a speed tolerance window about $V_\infty$ (with a translucent plane marking $\text{RPM}_*$).

## Notes and Potential Improvements

- **Air properties vs. temperature.** The code contains a partially finished commented function for temperature-dependent $\rho_a(T)$ and $\mu_a(T)$. Finishing this and coupling to the local air temperature would improve result accuracy. Furthering this, having a temperature gradient based on CFD results in combination with the calculated densities and viscosities would be useful as well.
- **Fin model.** The air-side correlation is lumped into a compact $j$-factor form; for cores with different louver geometries, would like to validate $Q$ outputs with real world testing.

**Part II**

# CFD Modeling and Darcy–Forchheimer Porous-Media Coefficients

## Purpose and Scope

This section contains the analysis of the companion Python script that derives porous-media resistance for the radiator core using an Ergun-based model based on Open-Foam and ANSYS documentation. It outputs coefficients in two CFD conventions:

- **SimScale:** viscous $d$ and inertial $f$ with $\Delta p = \mu\, d\, L\, u + \frac{\rho}{2}\, f\, L\, u^2$,
- **ANSYS Fluent:** viscous $1/\alpha$ and inertial $C_2$ with $\Delta p = \mu\,(1/\alpha)\, L\, u + \rho\, C_2\, L\, u^2$.

## Modeling Assumptions

- **Homogenized medium:** The fin–tube network is represented by a uniform porous block in the flow-normal direction.
- **Constant properties:** Resistance is geometry-driven; fluid $\rho, \mu$ only appear in the $\Delta p$ relation.
- **Packed-media surrogate:** Ergun form used as a proxy for louvered fins; fidelity hinges on porosity and length-scale choices.

## Geometry-Driven Porosity and Length Scale

The script forms a compact void fraction and a fin-pitch-based hydraulic length:

$$\phi = 1 - \frac{t_{\text{fin}}}{p_{\text{fin}}} - \frac{n_{\text{tube}}\, t_{\text{wall}}}{H_{\text{core}}}, \qquad d_h = 0.8\, p_{\text{fin}}.$$

Here $p_{\text{fin}}$ is fin pitch, $t_{\text{fin}}$ fin thickness, $t_{\text{wall}}$ tube wall thickness, $n_{\text{tube}}$ tube count across the span, and $H_{\text{core}}$ the open span height (face area divided by tube length).

## Ergun-Based Coefficients (per unit length)

With velocity $u$ through thickness $L$, the per-length pressure gradient is split into viscous (Darcy) and inertial (Forchheimer) parts:

$$k_{\text{perm}} = \frac{\phi^3\, d_h^2}{150\,(1-\phi)^2}, \qquad \beta = \frac{1.75\,(1-\phi)}{\phi^3\, d_h},$$

$$\frac{\Delta p}{L} = \mu\, \frac{u}{k_{\text{perm}}} + \rho\, \beta\, u^2.$$

## Backend Mappings

Match the per-length form $\dfrac{\Delta p}{L} = \mu \dfrac{u}{k_{\text{perm}}} + \rho\,\beta\,u^2$ to each solver's convention:

$$\text{SimScale:}\;\; \Delta p = \mu\,d\,L\,u + \tfrac{\rho}{2}\,f\,L\,u^2 \quad\Rightarrow\quad d = \frac{1}{k_{\text{perm}}},\;\; f = 2\,\beta,$$

$$\text{Fluent:}\;\; \Delta p = \mu\,(1/\alpha)\,L\,u + \rho\,C_2\,L\,u^2 \quad\Rightarrow\quad \frac{1}{\alpha} = \frac{1}{k_{\text{perm}}},\;\; C_2 = \beta.$$

## Algorithmic Steps

1. Convert all geometric inputs to SI (m, m$^2$) if provided in imperial.
2. Build core span height $H_{\text{core}}$ and depth from tube/fin counts and dimensions.
3. Compute $\phi = 1 - \dfrac{t_{\text{fin}}}{p_{\text{fin}}} - \dfrac{n_{\text{tube}}\,t_{\text{wall}}}{H_{\text{core}}}$ and $d_h = 0.8\,p_{\text{fin}}$.
4. Evaluate $k_{\text{perm}} = \dfrac{\phi^3\,d_h^2}{150\,(1-\phi)^2}$ and $\beta = \dfrac{1.75(1-\phi)}{\phi^3 d_h}$.
5. Report SimScale $(d, f)$ with $d = 1/k_{\text{perm}}$, $f = 2\beta$ and Fluent $(1/\alpha, C_2)$ with $1/\alpha = 1/k_{\text{perm}}$, $C_2 = \beta$.

## Sanity Checks and Use

- **Dimensions:** $[d] = [1/\alpha] = \text{m}^{-2}$, $[f] = [C_2] = \text{m}^{-1}$. Using either form—SimScale $\Delta p = \mu d L u + (\rho/2) f L u^2$ or Fluent $\Delta p = \mu(1/\alpha)Lu + \rho C_2 L u^2$—yields $\Delta p$ in pascals for $\mu$ [Pa·s], $\rho$ [kg·m$^{-3}$], $L$ [m], $u$ [m·s$^{-1}$].
- **Trends:** Tighter media ($\phi \downarrow$ or $d_h \downarrow$) $\Rightarrow$ $d$ and $f$ increase (larger $\Delta p$ for a given $u$).
- **Anisotropy:** If needed, scale tangential directions (e.g., 5–20×) to represent fin channeling; keep the face-normal values above.
- **Calibration option:** If a thickness-$L$ fit yields $\Delta p = A\,u + B\,u^2$, then for SimScale $d = A/(\mu L)$ and $f = 2B/(\rho L)$; for Fluent $1/\alpha = d$ and $C_2 = f/2$.

# Part III
# Sidepod Design and Geometry Script

## Purpose and Scope

This part documents sidepod design and the tools that help generate them. First, the design calculator that sizes the inlet and outlet tapers around a prescribed radiator center section. It determines inlet/outlet heights, widths, areas, taper lengths, and half-angles subject to user constraints on overall length and allowable inlet expansion angle. The tool provides a fast iterative parametric aid for sidepod design prior to generating multiple detailed CAD models. The model is then generated in SolidWorks for use in CFD. The overall goal is a method to iterate among designs of varying inlet and outlet ratios and varying angles of expansion and contraction to simulate against.

## User Inputs

- Radiator core length $L_{\text{rad}}$ [in].
- Radiator inclination $\theta$ from horizontal [deg].
- Center-section width $W_c$ [in].
- Inlet area ratio $\text{ratio}_{A,\text{in}} = A_{\text{in}}/A_c$ [$-$].
- Outlet area ratio $\text{ratio}_{A,\text{out}} = A_{\text{out}}/A_c$ [$-$].
- Overall sidepod length $L_{\text{tot}}$ [in].
- Maximum inlet resultant half-angle $\alpha_{\max}$ [deg].
- Minimum outlet-to-inlet length ratio $R_{\text{lo,min}}$ [$-$].
- Optional enforced length ratio $R_\ell = L_{\text{in}}/L_{\text{out}}$ (enable/disable) [$-$].
- Center-length buffer $b_L$ [in].
- Center-height buffer $b_H$ [in].

## Methodology and Governing Relations

### Center-section construction

Given the radiator length $L_{\text{rad}}$ inclined at $\theta$,

$$H_c = L_{\text{rad}} \sin\theta + b_H, \tag{40}$$
$$L_c = L_{\text{rad}} \cos\theta + b_L, \tag{41}$$
$$A_c = H_c W_c, \qquad L_{c,\frac{1}{2}} = \frac{L_c}{2}. \tag{42}$$

The remaining length available for tapers is

$$L_{\text{rem}} = L_{\text{tot}} - L_c \quad (> 0 \text{ required}). \tag{43}$$

### Area-ratio scaling for inlet/outlet frames

To realize a target area ratio $s \in \{\text{ratio}_{A,\text{in}}, \text{ratio}_{A,\text{out}}\}$, heights and widths are scaled by $\sqrt{s}$ so that the area scales linearly:

$$H(s) = H_c\sqrt{s}, \qquad W(s) = W_c\sqrt{s}, \qquad A(s) = s\, A_c. \tag{44}$$

Apply at inlet ($s = \text{ratio}_{A,\text{in}}$) and outlet ($s = \text{ratio}_{A,\text{out}}$) to obtain $(H_{\text{in}}, W_{\text{in}}, A_{\text{in}})$ and $(H_{\text{out}}, W_{\text{out}}, A_{\text{out}})$. Define geometry deltas relative to the center frame

$$\Delta H_{\text{in}} = |H_c - H_{\text{in}}|,\ \Delta W_{\text{in}} = |W_c - W_{\text{in}}|; \qquad \Delta H_{\text{out}} = |H_c - H_{\text{out}}|,\ \Delta W_{\text{out}} = |W_c - W_{\text{out}}|. \tag{45}$$

### Allocation of inlet and outlet taper lengths

Two modes are supported.

**Mode A (enforce length ratio).** If the length-ratio switch is active with $R_\ell = L_{\text{in}}/L_{\text{out}}$,

$$L_{\text{in}} = \frac{R_\ell}{1+R_\ell} L_{\text{rem}}, \qquad L_{\text{out}} = L_{\text{rem}} - L_{\text{in}}. \tag{46}$$

**Mode B (angle-limited inlet with feasibility guard).** Given $\alpha_{\max}$ and a minimum outlet/inlet ratio $R_{\text{lo,min}}$,

$$L_{\text{in,req}} = \frac{\sqrt{\left(\frac{\Delta H_{\text{in}}}{2}\right)^2 + \Delta W_{\text{in}}^2}}{\tan \alpha_{\max}}, \tag{47}$$

$$L_{\text{out,req}} = L_{\text{rem}} - L_{\text{in,req}}. \tag{48}$$

If the requested split is feasible, $L_{\text{out,req}} \geq R_{\text{lo,min}} L_{\text{in,req}}$ and $0 \leq L_{\text{out,req}} < L_{\text{in,req}}$, adopt $L_{\text{in}} = L_{\text{in,req}}$, $L_{\text{out}} = L_{\text{out,req}}$. Otherwise, fall back to the guarded ratio:

$$L_{\text{in}} = \frac{L_{\text{rem}}}{1+R_{\text{lo,min}}}, \qquad L_{\text{out}} = L_{\text{rem}} - L_{\text{in}}. \tag{49}$$

### Half-angle calculations

In each taper, report height-half-angle, width-half-angle, and resultant half-angle:

$$\alpha_h = \arctan\left(\frac{\Delta H/2}{L}\right), \quad \alpha_w = \arctan\left(\frac{\Delta W}{L}\right), \quad \alpha_{\text{res}} = \arctan\left(\frac{\sqrt{(\Delta H/2)^2 + \Delta W^2}}{L}\right), \tag{50}$$

applied to inlet $(\Delta H_{\text{in}}, \Delta W_{\text{in}}, L_{\text{in}})$ and outlet $(\Delta H_{\text{out}}, \Delta W_{\text{out}}, L_{\text{out}})$. (Angles are printed in degrees.)

## Outputs and Report

The calculator prints and writes a report containing all the calculated and inputted values. The following is an example results table:

A file is created in the Sidepod folder with name $\lceil\theta\rceil$_ratio$_{A,\text{in}}$_ratio$_{A,\text{out}}$_$\alpha_{\text{res,in}}$\_$\alpha_{\text{res,out}}$`.txt`.

## Sidepod Generation

### 0.2 Base Sketch

From the outputs given in the table, the modeling dimensions are used to generate a SolidWorks model by altering the following sketches.

Table 2: SIDE-POD DIMENSION SUMMARY ($\theta$ from horizontal)

| Item | Symbol | Units | Value |
|---|---|---|---|
| **Center section** | | | |
| Center height | $H_c$ | in | 11.219 |
| Center width | $W_c$ | in | 5.200 |
| Center length (half) | $L_{c,\frac{1}{2}}$ | in | 4.950 |
| **Inlet frame** | | | |
| Inlet height | $H_{\text{in}}$ | in | 7.933 |
| Inlet width | $W_{\text{in}}$ | in | 3.677 |
| Inlet area | $A_{\text{in}}$ | in$^2$ | 29.170 |
| **Outlet frame** | | | |
| Outlet height | $H_{\text{out}}$ | in | 9.045 |
| Outlet width | $W_{\text{out}}$ | in | 4.192 |
| Outlet area | $A_{\text{out}}$ | in$^2$ | 37.921 |
| **Lengths** | | | |
| Inlet length | $L_{\text{in}}$ | in | 11.750 |
| Outlet length | $L_{\text{out}}$ | in | 2.350 |
| Length ratio | $L_{\text{in}}/L_{\text{out}}$ | – | 5.000 |
| Total length | $L_{\text{tot}}$ | in | 24.000 |
| **Inlet angles** | | | |
| Height half-angle | $\alpha_{h,\text{in}}$ | deg | 7.960 |
| Width half-angle | $\alpha_{w,\text{in}}$ | deg | 7.386 |
| Resultant half-angle | $\alpha_{\text{res,in}}$ | deg | 10.795 |
| **Outlet angles** | | | |
| Height half-angle | $\alpha_{h,\text{out}}$ | deg | 24.823 |
| Width half-angle | $\alpha_{w,\text{out}}$ | deg | 23.209 |
| Resultant half-angle | $\alpha_{\text{res,out}}$ | deg | 32.240 |
| **Criterion** | | | |
| Inlet-angle criterion: length-ratio mode ($L_{\text{in}}/L_{\text{out}} = 5$) | | | |

Table 3: DESIGN PARAMETERS

| Radiator Tilt Angle [deg] | Inlet Area Ratio [–] | Exhaust Area Ratio [–] |
|---|---|---|
| 60.0 | 0.50 | 0.65 |

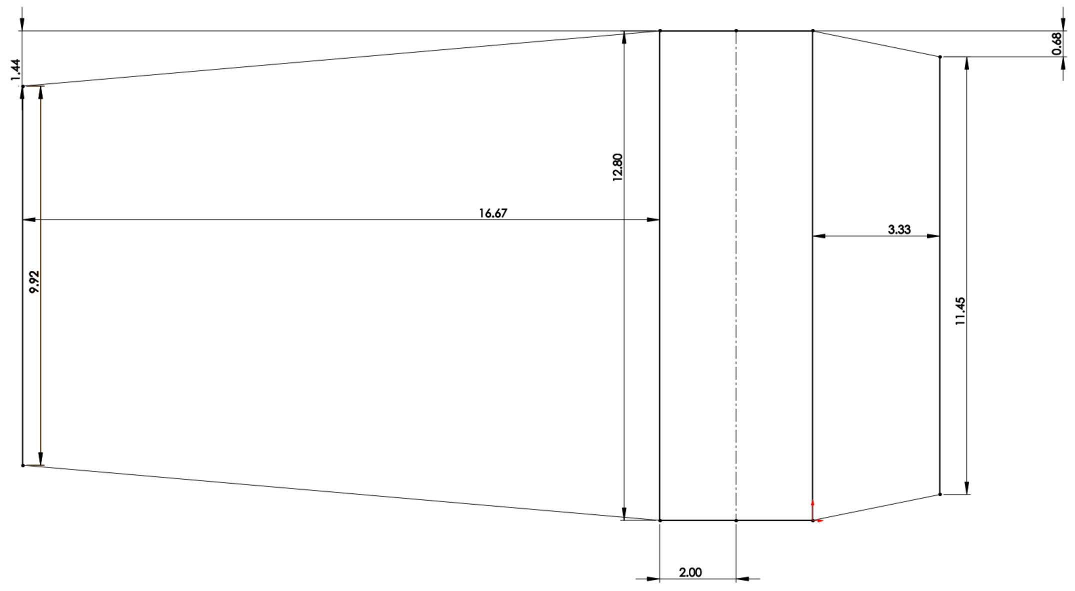

Figure 8: The base sketch in which the sidepod is based off of.

The above image is the side profile of the sidepod in which the main driven dimensions are edited. The sketch is driven by the following dimensions:

- Overall (center-section) height $H_c$ [in]
- Inlet opening height $H_{\text{in}}$ [in]
- Outlet opening height $H_{\text{out}}$ [in]
- Center-section length $L_c$ [in] (the sketch uses the half-length $L_{c,\frac{1}{2}} = L_c/2$)

The taper angles are implicitly defined by the model, but they match the calculated values from the script.

## 0.3 Cut Sketch

The widths are then adjusted via a cut feature from an extrusion off the base sketch of the width of the radiator.

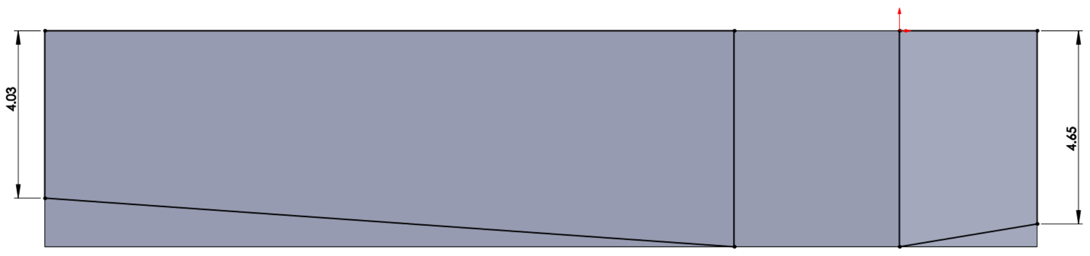

Figure 9: The base sketch in which the sidepod is based off of.

From this sketch the following dimensions are edited:

- Inlet opening width $W_{\text{in}}$ [in]
- Outlet opening width $W_{\text{out}}$ [in]

As before, the width taper angles are implicitly defined by the sketch and match the values calculated by the script.

## A Appendix A: CFD Air Velocity Input – Pass/Fail Conditions Under Load (Python)

C:/Users/pxzuk/OneDrive/Documents/OneDrive-MississippiStateUniversity/MississippiStateFSAE/FSAE2026/Design/Powertrain/Cooling/HeatTransferCalculations/CFDAirVelocityInput-Passfailcondit
py

```
import os
import sys
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from typing import Tuple


# Script to determine pass fail conditions at every operating point as determined by 2025 Autocross and Endurance Events.

#                                                                                    0. USER INPUTS

# Load BHP data: UPDATE THIS TO REFLECT USER ON LOCAL SYSTEM, OTHERWISE FILE WILL NOT BE READ
csv_path = (r C:\Users\pxzuk\OneDrive\Documents\OneDrive - Mississippi State University
            r \Mississippi State FSAE\FSAE 2026\Design\Powertrain\Tunes\VE Calculations
            r \BHP by MAP and RPM.csv )

# Fail fast if the file is missing
if not os.path.isfile(csv_path):
    print(f ERROR: CSV not found at: {csv_path} )
    sys.exit(1)

# Attempt to read; exit on any read/parse issues
try:
    bhp_df = pd.read_csv(csv_path)
except (pd.errors.EmptyDataError, pd.errors.ParserError, UnicodeDecodeError) as e:
    print(f ERROR: Failed to parse CSV: {e} )
    sys.exit(1)
except PermissionError as e:
    print(f ERROR: Permission denied reading CSV: {e} )
    sys.exit(1)
except FileNotFoundError as e:  # in case the file disappears between the check and read
    print(f ERROR: CSV missing: {e} )
    sys.exit(1)
except Exception as e:
    print(f ERROR: Unexpected failure reading CSV: {e} )
    sys.exit(1)


bhp_df = pd.read_csv(csv_path)


# Coolant Flow Rate Values - Experimentally measured from the car. Units: CFM
```

```python
coolant_ref = {
    2000: 0.3878,
    3000: 0.4277,
    4000: 0.6685,
    5000: 0.6950,
    6000: 0.8288,
    7000: 0.9493,
    8000: 0.9892,
    9000: 1.0428,
    10000: 1.1096
}

# Velocity profile and other constants - v_air_profile is CFD input, CHECK UNITS. The rest are assumed values or other inputs. rad_angle_deg is based on the sidepod geometry.
N_seg = 1
v_air_profile = np.array([30], dtype=float)        # ft  s
T_c_inlet = 210.0                                               #  F
T_air = 100.0                                               #  F
thermal_frac = 0.32
mech_eff = 0.28
rad_angle_deg = 90
A_core_face = 0.84                                                      # ft
assert len(v_air_profile) == N_seg

# Vehicle Information
V_inf = 50  # Free Stream Velocity ft/s
speed_tolerance = 0.2
gear_ratios = {1: 28/14, 2: 26/16, 3: 24/18, 4: 24/21, 5: 21/22}
primary_reduction = 70/29
final_drive = 2.75
tire_diam_in = 10

# Sidepod Information
A_in = 0.55           # ft      inlet opening area
A_out = 0.84           # ft      outlet opening area
expansion_angle = 14
contraction_angle = 30

#                                                                              1. GEOMETRY


# Radiator Inputs
n_tube = 11
tube_h = 0.05558
tube_w = 0.006583
tube_L = 1.666
n_rows = 2
fin_d = 0.00625
fin_t = 0.000167
eta_fin = 0.88
wall_t = 0.001583
k_w = 120
# Calculated Radiator Values
A_front = A_core_face * math.sin(math.radians(rad_angle_deg))
A_flow = n_tube * tube_w * tube_h
D_h_c = 4 * (tube_h * tube_w) / (2 * (tube_h + tube_w))
A_c_tot = n_tube * n_rows * (2 * (tube_h + tube_w) * tube_L)
```

```
A_w_tot = A_c_tot
core_W = tube_L
core_H = A_front / core_W
FPI = 1.0 / (fin_d * 12.0)
N_fins = int(FPI * core_W * 12)
core_depth = n_rows * (tube_h + 2*wall_t + fin_t)
W_fin = core_depth
A_fin_tot = 2.0 * N_fins * core_H * W_fin
tube_h_o = tube_h + 2*wall_t
tube_w_o = tube_w + 2*wall_t
A_bare = n_tube * n_rows * (2 * (tube_h_o + tube_w_o) * tube_L)
A_a_tot = eta_fin * A_fin_tot + A_bare
# per-segment areas
A_front_seg = A_front / N_seg
A_a_seg = A_a_tot / N_seg
A_c_seg = A_c_tot / N_seg
A_w_seg = A_w_tot / N_seg

#                                                                            2.
    PROPERTIES

# Coolant
rho_c = 59.8
mu_c = 0.00022
k_c = 0.37
C_p_c = 1.00
mu_c_hr = mu_c * 3600
Pr_c = C_p_c * mu_c_hr / k_c
# Air
rho_air = 0.0749
mu_air = 1.2e-05
k_air = 0.0137
C_p_air = 0.24
Pr_air = C_p_air * (mu_air * 3600) / k_air
D_h_air = tube_h_o
#                                                                            3.
    HELPER FUNCTIONS
rpm_ref = np.array(sorted(coolant_ref.keys()), dtype=float)
cfm_ref = np.array([coolant_ref[r] for r in rpm_ref], dtype=float)
slope_lo = (cfm_ref[1] - cfm_ref[0]) / (rpm_ref[1] - rpm_ref[0])
slope_hi = (cfm_ref[-1] - cfm_ref[-2]) / (rpm_ref[-1] - rpm_ref[-2])

# UPDATE THIS TO DYNAMICALLY CALCULATE AIR PROPERTIES BASED ON TEMP
'''
def air_props_english(T_F: float = 68.0) -> Tuple[float, float]:

    Air density     [lbm / ft ]  and dynamic viscosity     [lbm / ft  s]
    at 1 atm as a function of temperature in   F .

        Density:  ideal-gas scaling from the ISA sea-level reference
               (      = 0.0023769 slug/ft    = 0.07647 lbm/ft   at 59   F ).
        Viscosity: Sutherlands law with
                     = 1.20    10       lbm/ft  s at 59   F  ( T   = 518.67  R ),
               S   = 198.72  R   (Sutherland constant for air).

    Parameters
    ----------
    T_F : float
        Temperature in degrees Fahrenheit (default 68  F ).

    Returns
    -------
    rho_lbm_ft3 : float
        Density in lbm / ft  .
    mu_lbm_fts  : float
        Dynamic viscosity in lbm / ft s.


    # --- conversion to absolute Rankine ---------------------------
    T_R   = T_F + 459.67                     # [ R ]
    T_ref = 518.67                           # 59  F  in  R

    # --- density (ideal gas, p = const = 1 atm) -------------------
    rho_ref_slug = 0.0023769               # [slug/ft  ] at T_ref
    g_c          = 32.174                  # [lbm ft / lbf s  ]
    rho_ref      = rho_ref_slug * g_c      # => 0.07647 lbm/ft

    rho = rho_ref * (T_ref / T_R)          # rho     1/T  (ideal gas)

    # --- viscosity (Sutherland) -----------------------------------
    mu_ref = 1.20e-5                       # [lbm/ft s] at T_ref
    S      = 198.72                        # Sutherland const [ R ]

    mu = mu_ref * (T_R / T_ref)**1.5 * (T_ref + S) / (T_R + S)

    return rho, mu

'''


def coolant_cfm(rpm: float) -> float:
       Linear interpolate / extrapolate coolant flow [cfm] at a given rpm.
    if rpm < rpm_ref[0]:
        return cfm_ref[0] + slope_lo * (rpm - rpm_ref[0])
    if rpm > rpm_ref[-1]:
        return cfm_ref[-1] + slope_hi * (rpm - rpm_ref[-1])
    return float(np.interp(rpm, rpm_ref, cfm_ref))


def Nu_MB_louver(Re: float, Pr: float) -> float:
    j = 0.6522 * Re**-0.5403 * \
        (1 + 5.269e-5 * Re**1.340) * Pr**(1/3) * (1 + 0.504 / Pr**(2/3))
    return j * Re * Pr**(1/3)


def eps_crossflow_unmixed(NTU: float, C_r: float) -> float:
    if C_r == 0.0:
        return 1.0 - math.exp(-NTU)
    term = (1.0 / C_r) * (1.0 - math.exp(-C_r * NTU**0.78)) * NTU**0.22
    return 1.0 - math.exp(-term)


#                                                                      4.
   MAIN LOOP


results = []

```

```
for _, row in bhp_df.iterrows():
    rpm = float(row[ RPM_bin ])
    bhp = float(row[ average_BHP ])
    map_bin = row[ MAP_bin ]          # kept as label only

    Q_req = (bhp / mech_eff) * thermal_frac * 2545            # Btu  h r
    cf_cfm = coolant_cfm(rpm)

    # Coolant-side film coefficient
    v_c = (cf_cfm / 60.0) / A_flow                            # ft   s
    Re_c = rho_c * v_c * D_h_c / mu_c
    Nu_c = 0.023 * Re_c**0.8 * Pr_c**0.4
    h_c = Nu_c * k_c / D_h_c

    m_dot_c = rho_c * v_c * A_flow * 3600                     # lb  h r
    C_c = m_dot_c * C_p_c

    T_c_in = T_c_inlet
    Q_total = 0.0

    for j in range(N_seg):
        v_air = float(v_air_profile[j])

        Re_a = rho_air * v_air * D_h_air / mu_air
        Nu_a = Nu_MB_louver(Re_a, Pr_air)
        h_a = Nu_a * k_air / D_h_air

        R_air = 1.0 / (h_a * A_a_seg)
        R_wall = wall_t / (k_w * A_w_seg)
        R_cool = 1.0 / (h_c * A_c_seg)
        UA_seg = 1.0 / (R_air + R_wall + R_cool)

        m_dot_a = rho_air * v_air * A_front_seg * 3600
        C_a = m_dot_a * C_p_air

        C_min = min(C_c, C_a)
        C_r = C_min / max(C_c, C_a)
        NTU = UA_seg / C_min
        eps = eps_crossflow_unmixed(NTU, C_r)

        q_seg = eps * C_min * (T_c_in - T_air)
        T_c_in = T_c_in - q_seg / C_c
        Q_total += q_seg

    margin_pct = (Q_total - Q_req) / Q_req * 100.0
    results.append((map_bin, rpm, row[ count ],
                     Pass  if Q_total >= Q_req else  Fail ,
                    margin_pct))

#                                                                         4-
    bis. VEHICLE SPEEDS
tire_circ_ft = math.pi * (tire_diam_in / 12)


def vehicle_speed(engine_rpm: float, gear: int) -> float:
    total_ratio = gear_ratios[gear] * primary_reduction * final_drive
    wheel_rps = (engine_rpm / 60) / total_ratio
    return wheel_rps * tire_circ_ft     # ft/s
```

```


# add one speed column per gear
for g in gear_ratios:
    col = f Speed_g{g}
    for i, row in enumerate(results):
        rpm = row[1]
        speed = vehicle_speed(rpm, g)
        results[i] = (*row, speed)      # extend the tuple

#                                                                  5. REPORT

print(f {'MAP':>5} | {'RPM':>6} | {'Count':>4} | {'Result':>4} | {'Margin (%)':>11} )
print( -  * 49)
for m, r, c, flag, pct, *_ in results:     #    * _   swallows the speed columns
    print(f {m:>5} | {int(r):>6} | {int(c):>4} | {flag:>4} | {pct:>11.2f} )


#                                                                  5-bis.  SAVE TABLE TO CSV

cols_base = [ MAP_bin ,  RPM_bin ,  Count ,  Result ,  Margin_pct ]
cols_speed = [f Speed_g{g}  for g in gear_ratios]      # g = 1  5
df_out = pd.DataFrame(results, columns=cols_base + cols_speed)

out_path = (r C:\Users\pxzuk\OneDrive\Documents\OneDrive - Mississippi State University\Mississippi State FSAE\FSAE 2026\Design\Powertrain\Cooling\Heat Transfer Calculations/testingSpeed.csv )

# ensure folder exists
os.makedirs(os.path.dirname(out_path), exist_ok=True)
df_out.to_csv(out_path, index=False)

print(f \nCSV written to: {out_path} )

#                                                                  6. PLOTS

maps = [r[0] for r in results]
rpms = [r[1] for r in results]
counts = [r[2] for r in results]
flags = [r[3] for r in results]
margins = [r[4] for r in results]
speeds = {g: [r[5+g-1] for r in results] for g in gear_ratios}  # dict of lists
colors = ['green' if f == 'Pass' else 'red' for f in flags]

# 3-D scatter: RPM    MAP    Margin
fig3d = plt.figure(figsize=(7, 5))
ax3d = fig3d.add_subplot(111, projection='3d')
ax3d.scatter(rpms, maps, margins, c=colors, marker='o')

#        translucent reference plane at 0 % margin

x_plane = [min(rpms), max(rpms)]
```

```
y_plane = [min(maps), max(maps)]
X, Y = np.meshgrid(x_plane, y_plane)
Z = np.zeros_like(X)                          # z = 0 %
ax3d.plot_surface(X, Y, Z,
                  color='lightcoral', alpha=0.18, linewidth=0, antialiased=False)

# axis labels and title
ax3d.set_xlabel( RPM )
ax3d.set_ylabel( MAP_bin )
ax3d.set_zlabel( Margin (%) )
ax3d.set_title( Cooling Margin vs RPM & MAP )

# ---------- additional 3-D plot: RPM    Margin    Count --------------
counts = [row[2] for row in results]      # z-axis values
fig_cnt = plt.figure(figsize=(7, 5))
ax_cnt = fig_cnt.add_subplot(111, projection='3d')
ax_cnt.scatter(rpms, margins, counts, c=colors, marker='o')

ax_cnt.set_xlabel( RPM )
ax_cnt.set_ylabel( Margin (%) )
ax_cnt.set_zlabel( Count )
ax_cnt.set_title( Operating Density vs RPM & Cooling Margin )

plt.show()

'''# Optional 2-D pass/fail map (comment out if not needed)
fig2d, ax2d = plt.subplots(figsize=(6, 5))
ax2d.scatter(rpms, maps, c=colors, marker='o')
ax2d.set_xlabel ( RPM )
ax2d.set_ylabel ( MAP_bin )
ax2d.set_title ( Pass/Fail Map (green = pass, red = fail) )

plt.show()
'''

#                                                                          Gear-
    filtered 3-D plots
# Global axis limits for consistency across all gear figures
rpm_min,  rpm_max = min(rpms),  max(rpms)
map_min,  map_max = min(maps),  max(maps)
count_min, count_max = 0, max(counts)

v_low, v_hi = (1 - speed_tolerance) * V_inf, (1 + speed_tolerance) * V_inf

for g, ratio in gear_ratios.items():
    filt = [v_low <= v <= v_hi for v in speeds[g]]
    if not any(filt):
        continue

    fig = plt.figure(figsize=(6, 4.5))
    ax = fig.add_subplot(111, projection='3d')

    # scatter points that satisfy the speed window
    ax.scatter(
        np.array(rpms)[filt],           # X : RPM
        np.array(maps)[filt],           # Y : MAP_bin
        np.array(counts)[filt],         # Z : Count
        c=np.array(colors)[filt],
        marker='o'
    )

    # fixed, dataset-wide axis limits
    ax.set_xlim(rpm_min,   rpm_max)
    ax.set_ylim(map_min,   map_max)
    ax.set_zlim(count_min, count_max)

    ax.set_xlabel( RPM )
    ax.set_ylabel( MAP_bin )
    ax.set_zlabel( Count )
    ax.set_title(
        f Gear {g}: bins with vehicle speed {v_low:.0f}   {v_hi:.0f} ft/s )

    #         translucent vertical plane at RPM where vehicle speed = V_inf
    rpm_eq = V_inf * ratio * primary_reduction * final_drive * 60 / tire_circ_ft
    Yp, Zp = np.meshgrid([map_min, map_max], [count_min, count_max])
    Xp = np.full_like(Yp, rpm_eq)
    ax.plot_surface(Xp, Yp, Zp, color='lightcoral', alpha=0.18, linewidth=0)

plt.show()
```

## B Appendix B: Darcy–Forchheimer Calculator (Python)

C:/Users/pxzuk/OneDrive/Documents/OneDrive-MississippiStateUniversity/MississippiStateFSAE/FSAE2026/Design/Powertrain/Cooling/HeatTransferCalculations/DarcyForchheimerCalculator.py

```
# Darcy Forchheimer coefficients for radiator core:
# - Outputs BOTH SimScale (d, f) and Fluent (1/ , C2) porous media inputs.
# - Uses Ergun-based correlations with a chosen hydraulic length scale.

import math
import numpy as np

#                                                                 USER INPUTS

V_inf = 58.0            # ft/s   free-stream test speed (not used directly here)
A_in = 0.55             # ft      inlet opening area
A_out = 0.75            # ft      outlet opening area
fin_pitch = 0.00625*12*25.4   # mm   (= 1.9 mm)  estimate from geometry
Cd_inlet = 1.05         # external form-drag coefficient (not used here)
Cd_plate = 1.17         # vertical flat-plate  C d      (not used here)
N_seg = 1
v_air_profile = np.array([30.0], dtype=float)   # ft/s (not used here)
T_c_inlet = 210.0       #  F  (not used here)
T_air = 100.0           #  F  (not used here)
thermal_frac = 0.32     # (not used here)
mech_eff = 0.28         # (not used here)
rad_angle_deg = 60.0
A_core_face = 0.84      # ft
rho_air = 0.075         # lbm/ft   (used only if converting to SI; not needed for d,f from Ergun)
g = 32.2                # ft/s
assert len(v_air_profile) == N_seg

expansion_angle = 14 # degrees (not used here)
contraction_angle = 30 # degrees (not used here)

# Radiator core geometry (imperial -> will convert to SI)
n_tube = 11
tube_h = 0.05558        # ft
tube_w = 0.006583       # ft
tube_L = 1.666          # ft
n_rows = 2
fin_d = 0.0119          # ft     (fin pitch along flow-normal,     spacing)
fin_t = 0.000167        # ft     (fin thickness)
eta_fin = 0.88
wall_t = 0.001583       # ft     (tube wall thickness)
k_w = 120.0             # W/m-K (not used here)

# Air properties (only needed if converting from   pu   fits; kept for completeness)
rho_air = 0.0749        # lbm/ft
mu_air = 1.2e-05        # lbm/(ft s)
k_air = 0.0137          # BTU/(hr ft   F) (not used here)
C_p_air = 0.24          # BTU/(lbm   F)    (not used here)

#                                       UNIT CONVERSION CONSTANTS
```

```
ft_to_m  = 0.3048
in_to_m  = 0.0254
mm_to_m  = 0.001
lbm_ft3_to_kg_m3   = 0.453592/(ft_to_m**3)
lbm_ft_s_to_kg_m_s = 0.453592/ft_to_m

#                                       CONVERT PRIMARY INPUTS TO SI

V_inf         *= ft_to_m                   # m/s
v_air_profile = v_air_profile * ft_to_m # m/s
A_in          *= ft_to_m**2                #  m
A_out         *= ft_to_m**2                #  m
A_core_face   *= ft_to_m**2                #  m
tube_h        *= ft_to_m                   # m
tube_w        *= ft_to_m                   # m
tube_L        *= ft_to_m                   # m
fin_d         *= ft_to_m                   # m
fin_t         *= ft_to_m                   # m
wall_t        *= ft_to_m                   # m
rho_air_SI    = 0.0749 * lbm_ft3_to_kg_m3           # kg/ m
mu_air_SI     = 1.2e-05 * lbm_ft_s_to_kg_m_s        # kg/( m  s )
g_SI          = 32.2 * ft_to_m                      # m/ s

#    GEOMETRY

A_front    = A_core_face * math.sin(math.radians(rad_angle_deg))
A_flow     = n_tube * tube_w * tube_h
D_h_c      = 4 * (tube_h * tube_w) / (2 * (tube_h + tube_w))
A_c_tot    = n_tube * n_rows * (2 * (tube_h + tube_w) * tube_L)
A_w_tot    = A_c_tot
core_W     = tube_L
core_H     = A_core_face / core_W
FPI        = 1.0 / (fin_d * 39.3701)                   # fins per inch (informational)
N_fins     = int(FPI * core_W * 39.3701)
core_depth = n_rows * (tube_h + 2*wall_t + fin_t)   # m
W_fin       = core_depth
A_fin_tot   = 2.0 * N_fins * core_H * W_fin
tube_h_o    = tube_h + 2*wall_t
tube_w_o    = tube_w + 2*wall_t
A_bare      = n_tube * n_rows * (2*(tube_h_o + tube_w_o) * tube_L)
A_a_tot     = eta_fin * A_fin_tot + A_bare

#                                                       POROSITY & LENGTH SCALE

# Porosity model (same as original; ensure dimensionless)
porosity = 1.0 - (fin_t / fin_d) - (n_tube * wall_t) / core_H
phi = porosity

# Choice of hydraulic length scale for Ergun (modeling choice for finned cores)
hydraulic_len_scale = 0.8 * fin_d  # m, tunable; consider passage hydraulic diameter if available

#                                                           ERGUN-BASED COEFFICIENTS

# Per-length coefficients:
# k_perm [m  ] from Ergun viscous term; beta [1/m] (Fluent convention) from
```

```
inertial term
k_perm = (phi**3 * hydraulic_len_scale**2) / (150.0 * (1.0 - phi)**2)  #  m
beta   = (1.75 * (1.0 - phi)) / (phi**3 * hydraulic_len_scale)         # 1/m  (Fluent C2 form)

#                                                          OUTPUTS FOR SIMSCALE VS FLUENT
# SimScale uses  p  =    d L u + (  /2) f L u         d [1/m  ], f [1/m]
d_sim = 1.0 / k_perm                                 # 1/m
f_sim = (3.5 * (1.0 - phi)) / (phi**3 * hydraulic_len_scale)  # 1/m  (note 3.5 = 2*1.75)

# Fluent uses  p  =    (1/  ) L u +    C2 L u         (1/  ) [1/m  ], C2 [1/m]
D11_fluent = 1.0 / k_perm                            # 1/m
C2_fluent  = beta                                     # 1/m

#
PRINT RESULTS

print(
     Porous-media coefficients (Ergun-based):\n
    f       Porosity   ....................... {phi:9.5f}\n
    f       Hydraulic length scale dh........ {hydraulic_len_scale:11.4e}  m\n\n
     SimScale  D a r c y Forchheimer  inputs ( p  =     dLu     + (  /2)    fLu    ):\n
    f       d  (Darcy, viscous)............. {d_sim:11.4e}  1/m  \n
    f       f  (Forchheimer, inertial)....... {f_sim:11.4e}  1/m\n
           Local flow direction e1.......... (1, 0, 0)  # set to face-normal in your CAD\n
           Cross-flow (e2,e3): scale d,f up (e.g.,  10 ) for anisotropy if desired.\n\n
     ANSYS Fluent porous-zone inputs ( p  =     (1/  )  Lu   +      C2Lu     ):\n
    f       1/   (viscous resistance)......... {D11_fluent:11.4e}  1/m  \n
    f       C2  (inertial resistance)........ {C2_fluent:11.4e}  1/m\n
)

#                                                       OPTIONAL: FIT CONVERSION HELPERS
def fit_to_simscale(A, B, mu, rho, L):

    Convert a  p  = A*u + B*u^2 fit across thickness L into SimScale per-length coefficients.
    A : linear fit coefficient      [Pa s/m]   (units depend on how the fit was formed)
    B : quadratic fit coefficient   [Pa  s  /m  ] (ditto)
    mu: dynamic viscosity           [Pa s]
    rho: density                    [kg/m  ]
    L : porous thickness used in fit [m]
    Returns: d [1/m  ], f [1/m]

    d = A / (mu * L)
    f = (2.0 * B) / (rho * L)
    return d, f

# Example usage (commented):
# A_fit, B_fit = 0.0, 0.0   # from your   pu   regression
# d_from_fit, f_from_fit = fit_to_simscale(A_fit, B_fit, mu_air_SI, rho_air_SI, core_depth)
# print(f From fit     d = {d_from_fit:.4e} 1/m  , f = {f_from_fit:.4e} 1/m )
```

## Appendix C: Sidepod Geometry Calculator (Python)

C:/Users/pxzuk/OneDrive/Documents/OneDrive-MississippiStateUniversity/MississippiStateFSAE/FSAE2026/Design/Powertrain/Cooling/Sidepod/SidePodGeomertyCalculator.py

```python
import math
import os, sys
from contextlib import redirect_stdout

#


# USER INPUTS                                                  --
# -------------------------------------------------------------
L_rad = 11.8        # radiator core length   [in]
theta = 60          # inclination from HORIZONTAL [deg]
W_c   = 5.2         # center-section width   [in]

ratio_A_in  = 0.50    # inlet  area / center area [-]
ratio_A_out = 0.65    # outlet area / center area [-]

L_tot     = 24.0    # overall inlet-to-outlet length [in]
angle_lim = 15.0    # max desired inlet half-angle    [deg]
R_lo_min  = 0.20    # min L_out / L_in                [-]

# --- NEW: optional length-ratio control ------------------------
R_len      = 5    # desired L_in / L_out (ignored unless use_R_len=True)
use_R_len  = True   # set True to enforce R_len
# ------------------------------------------------------------

center_length_buffer  = 4
center_height_buffer  = 1
#



# center-SECTION GEOMETRY
H_c = L_rad * math.sin(math.radians(theta)) + center_height_buffer
L_c = L_rad * math.cos(math.radians(theta)) + center_length_buffer
A_c = H_c * W_c
L_c_half = L_c / 2


def scaled_dims(scale: float, H_ref: float, W_ref: float):
    s = math.sqrt(scale)
    return H_ref * s, W_ref * s


H_in,  W_in  = scaled_dims(ratio_A_in,  H_c, W_c)
H_out, W_out = scaled_dims(ratio_A_out, H_c, W_c)
A_in,  A_out = A_c * ratio_A_in, A_c * ratio_A_out

 H_in  ,   W_in   = abs(H_c - H_in),  abs(W_c - W_in)
 H_out  ,  W_out  = abs(H_c - H_out), abs(W_c - W_out)

top_drop_in   =  H_in   / 2.0
top_drop_out  =  H_out  / 2.0

```

```
#        SPLIT REMAINING LENGTH BETWEEN TAPERS

L_rem = L_tot - L_c
if L_rem <= 0:
    raise ValueError( Total length too small to accommodate center section. )

if use_R_len:                                          #     new branch
    L_in  = (R_len / (1 + R_len)) * L_rem
    L_out = L_rem - L_in
    angle_status = f length-ratio mode (L_in / L_out = {R_len:g})
else:
    angle_lim_rad = math.radians(angle_lim)
    L_in_req  = math.hypot( H_in  / 2.0,  W_in ) / math.tan(angle_lim_rad)
    L_out_req = L_rem - L_in_req

    if (L_out_req >= R_lo_min * L_in_req) and (0 <= L_out_req < L_in_req):
        L_in, L_out = L_in_req, L_out_req
        angle_status = f     {angle_lim:0.1f}   satisfied
    else:
        L_in  = L_rem / (1 + R_lo_min)
        L_out = L_rem - L_in
        angle_status = (f angle > {angle_lim:0.1f}   or L_in      L_out
                             lengths redistributed )


def half_angles(dH: float, dW: float, L: float):
    angle_h  = math.degrees(math.atan((dH/2)/L))
    angle_w  = math.degrees(math.atan(dW / L))
    angle_r  = math.degrees(math.atan(math.hypot(dH/2, dW) / L))
    return angle_h, angle_w, angle_r


angleh_in,  anglew_in,  angleres_in  = half_angles( H_in ,   W_in ,  L_in)
angleh_out, anglew_out, angleres_out = half_angles( H_out ,  W_out , L_out)


def line(lbl, val):
    print(f {lbl:<29s} {val:>10.3f} )


print( \nSIDE-POD DIMENSION SUMMARY (   from horizontal) )
print(      * 43)
line( center height  H_c  [in] , H_c)
line( center width   W_c  [in] , W_c)
line( center length  L_c_half  [in] , L_c_half)
print(      * 43)
line( Inlet  height  H_in [in] , H_in)
line( Inlet  width   W_in [in] , W_in)
line( Inlet  area    A_in [in  ] , A_in)
line( Outlet height  H_out[in] , H_out)
line( Outlet width   W_out[in] , W_out)
line( Outlet area    A_out[in  ] , A_out)
print(      * 43)
line( Inlet length   L_in [in] , L_in)
line( Outlet length  L_out[in] , L_out)
line( L_in / L_out         [-] ,  L_in / L_out)   # shows actual ratio
line( Total length   L_tot[in] , L_tot)
print(      * 43)
line( Inlet angle_height [deg] , angleh_in)
line( Inlet angle_width  [deg] , anglew_in)
line( Inlet angle_result [deg] , angleres_in)
print(      * 43)
line( Outlet angle_height[deg] , angleh_out)
line( Outlet angle_width [deg] , anglew_out)
line( Outlet angle_result[deg] , angleres_out)
print(      * 43)
print(f Inlet-angle criterion: {angle_status} )

#                                                      REPORT (unchanged)


output_dir = (r C:\Users\pxzuk\OneDrive\Documents\OneDrive -
              r Mississippi State University\Mississippi State FSAE\FSAE
              r 2026\Design\Powertrain\Cooling\Sidepod )

def make_filename(theta_deg, r_in, r_out, alpha_in, alpha_out):
    return (f {int(round(theta_deg))}_{r_in:g}_{r_out:g}_
            f {alpha_in:.1f}_{alpha_out:.1f}.txt )

fname = make_filename(theta, ratio_A_in, ratio_A_out,
                      angleres_in, angleres_out)
if output_dir:
    os.makedirs(output_dir, exist_ok=True)
    fname = os.path.join(output_dir, fname)


class Tee:
    def __init__(self, *streams): self.streams = streams
    def write(self, d):           [s.write(d) for s in self.streams]
    def flush(self):              [s.flush()  for s in self.streams]


with open(fname,  w , encoding= utf-8 ) as fh, \
     redirect_stdout(Tee(sys.__stdout__, fh)):

    print( \nDESIGN PARAMETERS )
    print( Radiator Tilt Angle [deg]   Inlet Area Ratio [-]   Exhaust Area Ratio
        [-] )
    print(f {theta:>22.1f}{ratio_A_in:>25.2f}{ratio_A_out:>27.2f} )
    print(      * 43)

    # (table section identical to console-printed block)
    # ... (omitted here for brevity) ...

print(f \nReport written to: {fname} )
```

## Appendix D: Calculated BHP Values From datalogs (Python)

C:/Users/pxzuk/OneDrive/Documents/OneDrive-MississippiStateUniversity/MississippiStateFSAE/FSAE2026/Design/Powertrain/Tunes/VECalculations/ScripttoCalculateVEfromDatalog.py

```
import math
import pandas as pd
import numpy as np

# Script To Calculate VE from a Datalog CSV with Time, RPM, MAP, O2 Cyl #1, AFR, Air Temp, and Fuel Pressure. All calculations are done in English Units.

file_path = r c:\Users\pxzuk\Desktop\Datalogs\Bennett Endurance_20250603-1735.csv
df = pd.read_csv(file_path)

# A List of the reference columns to search the CSV for. These are dependednt on the datalogging software. The following are for Fueltech and with O2 Lambda being collected on O2 Cyl #01

reference_columns = [ TIME ,                              # In Seconds
                      RPM ,
                      TPS ,
                      MAP ,                               # In PSIG
                      O2_Cyl_#01 ,                        # In Lambda
                      Primary_injection_time ,            # In Milliseconds (ms)
                      Air_temperature ,                   # In degrees F
                      Fuel_pressure                       # In PSIG
                     ]

# Find which columns are present in the CSV

found_columns = [col for col in df.columns if col in reference_columns]
found_indicies = [df.columns.get_loc(col) for col in found_columns]

results = pd.DataFrame({
     Reference Column : found_columns,
     Index in CSV : found_indicies
})


# Define Variables for User Input

stoich_AFR = 14.7
man_injector_flow = 42          # In lbs/hr, from manufacturer
injector_pressure = 43.5        # In PSIG, the pressure the injector flow is measured at
air_gas_constant = 640.2        # In in*lbf/lbm*R
engine_displacement = 27.46     # In cubic inches
number_of_cyl = 1               # Number of cylinders your engine has
exhaust_pipe_diameter = 1.75    # In inches
distance_to_O2_sensor = 24      # In inches
k_rho = 2.0                     # Hot/Cold volume expansion ratio constant (no EGT data so this is an assumption)
fuel_energy_content = 20600     # In BTU/lb
thermal_efficiency = 0.3        # Assumed value


# Perform Calculations to get all terms in the right units

```

```
df[ MAP_PSIA ] = df[ MAP ] + 14.7
df[ air_temp_R ] = df[ Air_temperature ] + 459.67
df[ AFR ] = df[ O2_Cyl_#01 ] * stoich_AFR
df[ inj_time_hr ] = df[ Primary_injection_time ] / 3600000
exh_pipe_area = math.pi * (exhaust_pipe_diameter ** 2) / 4

# Perform Calculations to determine VE inital

df[ injector_flow ] = man_injector_flow * np.sqrt(df[ Fuel_pressure ] / injector_pressure)
df[ fuel_mass_per_cycle ] = df[ injector_flow ] * df[ inj_time_hr ]
df[ air_mass_per_cycle ] = df[ fuel_mass_per_cycle ] * df[ AFR ]
df[ air_density ] = (14.7) / (air_gas_constant * df[ air_temp_R ])
df[ air_volume_per_cycle ] = (df[ air_mass_per_cycle ] / df[ air_density ])
df[ VE_initial ] = (df[ air_volume_per_cycle ] / engine_displacement) * 100

# Start Interpolation Cycle For VE convergence

VE_old =df[ VE_initial ]

# Set Iteration Peramters

max_iter = 1000
tol = 0.001
for iteration in range(max_iter):

    # Compute Exhaust Gas Velocity for Determining O2 Sensor Time Delay

    df[ engine_volflow_cold ] = number_of_cyl * engine_displacement * (df[ RPM ] / 120) * (VE_old/100) # cubic inches per second
    df[ engine_volflow_hot ] = df[ engine_volflow_cold ] * k_rho
    df[ exh_exit_velocity ] = df[ engine_volflow_hot ] / exh_pipe_area      # In inches per second

    # Compute Time Delay to O2 Sensor from Valve

    df[ time_delay ] = distance_to_O2_sensor / df[ exh_exit_velocity ]

    # Compute the desired time to read AFR off of

    df[ desired_time ] = df[ TIME ] + df[ time_delay ]

    # Perform Linear Interpolation on AFR vs TIME:
    df[ AFR_corrected ] =  np.interp(
        x=df[ desired_time ].to_numpy(),
        xp=df[ TIME ].to_numpy(),
        fp=df[ AFR ].to_numpy()
    )

    # Re-calculate VE using AFR_Corrected

    df[ air_mass_per_cycle_corr ] = df[ fuel_mass_per_cycle ] * df[ AFR_corrected ]
    df[ air_volume_per_cycle_corr ] = df[ air_mass_per_cycle_corr ] / df[ air_density ]
    VE_new = (df[ air_volume_per_cycle_corr ] / engine_displacement) * 100

    # Check Convergence
    max_diff = (VE_new - VE_old).abs().max()
    if max_diff < tol:
        VE_old = VE_new.copy()
        break
    VE_old = VE_new.copy()

df[ VE ] = VE_old
df[ VE_delta ] = ((df[ VE ] - df[ VE_initial ]) / df[ VE_initial ]) * 100

# Calculate Expected HP Value
df[ brake_horsepower_BTU ] = (df[ fuel_mass_per_cycle ] * fuel_energy_content *
    thermal_efficiency * (30 * df[ RPM ])) / 2545

# Export Data to CSV:
output_path =  C:\\Users\\pxzuk\\OneDrive\\Documents\\OneDrive - Mississippi State
     University\\Mississippi State FSAE\\FSAE 2026\\Design\\Powertrain\\Tunes\\VE
    Calculations\\Endurance.csv
df.to_csv(output_path, index=False)

print(df[[
     MAP_PSIA ,
     air_temp_R ,
     AFR ,
     injector_flow ,
     inj_time_hr ,
     air_mass_per_cycle ,
     air_density ,
     air_volume_per_cycle ,
     VE_initial ,
     engine_volflow_cold ,
     engine_volflow_hot ,
     exh_exit_velocity ,
     time_delay ,
     VE ,
     VE_delta ,
     brake_horsepower_BTU
]].head(10)
)
```

## Appendix E: Variable Definitions by Section

Table 4: Data-driven coolant flow

| Symbol | Definition |
|---|---|
| RPM | Engine speed [rev $\text{min}^{-1}$] |
| $i$ | Index of tabulated data point ($i = 1, \ldots, N$) |
| $N$ | Number of tabulated points (nondimensional) |
| $\dot{V}_{\text{coolant}}$ | Coolant volumetric flow rate [$\text{ft}^3\ \text{min}^{-1}$] |
| $s_{\text{lo}}$ | Low-end slope for $\dot{V}_{\text{coolant}}$ vs. RPM extrapolation [$\text{ft}^3\ \text{min}^{-1}\ \text{RPM}^{-1}$] |
| $s_{\text{hi}}$ | High-end slope for $\dot{V}_{\text{coolant}}$ vs. RPM extrapolation [$\text{ft}^3\ \text{min}^{-1}\ \text{RPM}^{-1}$] |

Table 5: Core geometry and areas

| Symbol | Definition |
|---|---|
| $A_{\text{core,face}}$ | Radiator core face area [ft$^2$] |
| $\theta_{\text{rad}}$ | Radiator tilt angle (to free stream) [°] |
| $A_{\text{front}}$ | Projected face area = $A_{\text{core,face}} \sin\theta_{\text{rad}}$ [ft$^2$] |
| $n_{\text{tube}}$ | Number of tubes across the core (nondimensional) |
| $n_{\text{rows}}$ | Number of tube rows through the depth (nondimensional) |
| $w_{\text{tube}}$ | Tube internal width [ft] |
| $h_{\text{tube}}$ | Tube internal height [ft] |
| $L_{\text{tube}}$ | Tube length (spanwise) [ft] |
| $A_{\text{flow}}$ | Coolant flow area = $n_{\text{tube}}\, w_{\text{tube}} h_{\text{tube}}$ [ft$^2$] |
| $D_{h,c}$ | Coolant-side hydraulic diameter [ft] |
| $A_{c,\text{tot}}$ | Total coolant-side primary area [ft$^2$] |
| $d_{\text{fin}}$ | Fin pitch (center-to-center spacing) [in] |
| FPI | Fins per inch = $1/(12\, d_{\text{fin}})$ [in$^{-1}$] |
| $N_{\text{fins}}$ | Number of fins across span (rounded) (nondimensional) |
| $t_{\text{wall}}$ | Tube wall thickness [ft] |
| $t_{\text{fin}}$ | Fin sheet thickness [ft] |
| $h_{\text{tube,o}}$ | Tube outer height = $h_{\text{tube}} + 2t_{\text{wall}}$ [ft] |
| $w_{\text{tube,o}}$ | Tube outer width = $w_{\text{tube}} + 2t_{\text{wall}}$ [ft] |
| $W_{\text{core}}$ | Core width (spanwise) [ft] |
| $H_{\text{core}}$ | Core height (face normal to span) [ft] |
| $W_{\text{fin}}$ | Fin wetted width through depth [ft] |
| $A_{\text{fin,tot}}$ | Total fin area (both sides) [ft$^2$] |
| $A_{\text{bare}}$ | Bare primary external area (tube outside) [ft$^2$] |
| $\eta_{\text{fin}}$ | Fin efficiency (nondimensional) |
| $A_{a,\text{tot}}$ | Total effective air-side area = $\eta_{\text{fin}} A_{\text{fin,tot}} + A_{\text{bare}}$ [ft$^2$] |
| $N_{\text{seg}}$ | Number of vertical segments (nondimensional) |
| $A_{\text{front,seg}}$ | Per-segment face area [ft$^2$] |
| $A_{a,\text{seg}}$ | Per-segment air-side area [ft$^2$] |
| $A_{c,\text{seg}}$ | Per-segment coolant-side area [ft$^2$] |
| $A_{w,\text{seg}}$ | Per-segment wall conduction area [ft$^2$] |

Table 6: Fluid properties and nondimensional groups

| Symbol | Definition |
|---|---|
| $\rho$ | Density [lbm ft$^{-3}$] |
| $\mu$ | Dynamic viscosity [lbm ft$^{-1}$ s$^{-1}$] |
| $\mu_{\mathrm{hr}}$ | Dynamic viscosity in hour units [lbm ft$^{-1}$ hr$^{-1}$] |
| $k$ | Thermal conductivity [Btu hr$^{-1}$ ft$^{-1}$ °F$^{-1}$] |
| $c_p$ | Specific heat [Btu lbm$^{-1}$ °F$^{-1}$] |
| Re | Reynolds number (nondimensional) |
| Pr | Prandtl number (nondimensional) |
| Nu | Nusselt number (nondimensional) |
| $\rho_a,\ \rho_c$ | Air, coolant densities [lbm ft$^{-3}$] |
| $\mu_a,\ \mu_c$ | Air, coolant viscosities [lbm ft$^{-1}$ s$^{-1}$] |
| $k_a,\ k_c$ | Air, coolant thermal conductivities [Btu hr$^{-1}$ ft$^{-1}$ °F$^{-1}$] |
| $c_{p,a},\ c_{p,c}$ | Air, coolant specific heats [Btu lbm$^{-1}$ °F$^{-1}$] |

Table 7: Convection coefficients

| Symbol | Definition |
|---|---|
| $j$ | Colburn $j$-factor (air side) (nondimensional) |
| $\mathrm{Re}_a,\ \mathrm{Re}_c$ | Air, coolant Reynolds numbers (nondimensional) |
| $\mathrm{Pr}_a,\ \mathrm{Pr}_c$ | Air, coolant Prandtl numbers (nondimensional) |
| $\mathrm{Nu}_a,\ \mathrm{Nu}_c$ | Air, coolant Nusselt numbers (nondimensional) |
| $D_{h,\mathrm{air}}$ | Air-side hydraulic diameter proxy [ft] |
| $h_a,\ h_c$ | Air, coolant film coefficients [Btu hr$^{-1}$ ft$^{-2}$ °F$^{-1}$] |

Table 8: Flows and capacity rates

| Symbol | Definition |
|---|---|
| $v_a,\ v_c$ | Air, coolant bulk velocities [ft s$^{-1}$] |
| $\dot{m}_a,\ \dot{m}_c$ | Air, coolant mass flow rates [lbm hr$^{-1}$] |
| $C_a,\ C_c$ | Air, coolant heat capacity rates [Btu hr$^{-1}$ °F$^{-1}$] |

Table 9: Thermal resistances and overall conductance

| Symbol | Definition |
|---|---|
| $R_{\mathrm{air}}$ | Air-side thermal resistance [hr °F Btu$^{-1}$] |
| $R_{\mathrm{wall}}$ | Wall conduction resistance [hr °F Btu$^{-1}$] |
| $R_{\mathrm{cool}}$ | Coolant-side thermal resistance [hr °F Btu$^{-1}$] |
| $UA_{\mathrm{seg}}$ | Overall conductance per segment [Btu hr$^{-1}$ °F$^{-1}$] |

Table 10: Effectiveness–NTU parameters

| Symbol | Definition |
|---|---|
| $C_{\min},\ C_{\max}$ | Min/max of $\{C_c,\, C_a\}$ [Btu $\mathrm{hr}^{-1}$ $^\circ\mathrm{F}^{-1}$] |
| $C_r$ | Capacity ratio $= C_{\min}/C_{\max}$ (nondimensional) |
| NTU | Number of Transfer Units $= UA_{\mathrm{seg}}/C_{\min}$ (nondimensional) |
| $\varepsilon$ | Heat-exchanger effectiveness (nondimensional) |

Table 11: Segment heat and temperatures

| Symbol | Definition |
|---|---|
| $q_{\mathrm{seg}}$ | Heat transferred in a segment [Btu $\mathrm{hr}^{-1}$] |
| $T_{c,\mathrm{in}},\ T_{c,\mathrm{out}}$ | Coolant inlet/outlet temperatures (segment) [°F] |
| $T_a$ | Air temperature [°F] |
| $Q_{\mathrm{total}}$ | Total heat rejection $\sum q_{\mathrm{seg}}$ [Btu $\mathrm{hr}^{-1}$] |

Table 12: Requirement and margin

| Symbol | Definition |
|---|---|
| BHP | Brake horsepower [hp] |
| $\eta_{\mathrm{mech}}$ | Mechanical efficiency (nondimensional) |
| $f_{\mathrm{thermal}}$ | Fraction of fuel energy to coolant load (nondimensional) |
| $Q_{\mathrm{req}}$ | Required heat rejection [Btu $\mathrm{hr}^{-1}$] |
| Margin | Percent margin $= 100\,(Q_{\mathrm{total}} - Q_{\mathrm{req}})/Q_{\mathrm{req}}$ [%] |

Table 13: Vehicle speed mapping

| Symbol | Definition |
|---|---|
| $r_{\mathrm{gear}}$ | Selected gear ratio (nondimensional) |
| $r_{\mathrm{primary}}$ | Primary reduction ratio (nondimensional) |
| $r_{\mathrm{final}}$ | Final drive ratio (nondimensional) |
| $r_{\mathrm{tot}}$ | Total ratio $= r_{\mathrm{gear}}\, r_{\mathrm{primary}}\, r_{\mathrm{final}}$ (nondimensional) |
| $D_{\mathrm{tire}}$ | Tire diameter [ft] |
| $C_{\mathrm{tire}}$ | Tire circumference $= \pi D_{\mathrm{tire}}$ [ft] |
| wheel rps | Wheel rotations per second [$\mathrm{s}^{-1}$] |
| $v$ | Vehicle speed [ft $\mathrm{s}^{-1}$] |
| $V_\infty$ | Target free-stream speed [ft $\mathrm{s}^{-1}$] |
| $\mathrm{RPM}_*$ | RPM giving $V_\infty$ in a given gear [rev $\mathrm{min}^{-1}$] |

Table 14: Sidepod geometry and taper allocation variables

| Symbol | Definition |
|---|---|
| $L_{\mathrm{rad}}$ | Radiator core length [in] |
| $\theta$ | Radiator inclination from horizontal [$^\circ$] |
| $W_c$ | Center-section width [in] |
| $\mathrm{ratio}_{A,\mathrm{in}}$ | Inlet area ratio $= A_{\mathrm{in}}/A_c$ [$-$] |
| $\mathrm{ratio}_{A,\mathrm{out}}$ | Outlet area ratio $= A_{\mathrm{out}}/A_c$ [$-$] |
| $L_{\mathrm{tot}}$ | Overall sidepod length [in] |
| $\alpha_{\max}$ | Maximum allowed inlet resultant half-angle [$^\circ$] |
| $R_{\mathrm{lo,min}}$ | Minimum outlet-to-inlet length ratio [$-$] |
| $R_\ell$ | Enforced length ratio $= L_{\mathrm{in}}/L_{\mathrm{out}}$ (if enabled) [$-$] |
| $b_L$ | Center-section length buffer [in] |
| $b_H$ | Center-section height buffer [in] |
| $H_c$ | Center-section height $= L_{\mathrm{rad}}\sin\theta + b_H$ [in] |
| $L_c$ | Center-section projected length $= L_{\mathrm{rad}}\cos\theta + b_L$ [in] |
| $A_c$ | Center-section area $= H_c W_c$ [in$^2$] |
| $L_{c,\frac{1}{2}}$ | Half-length of center section $= L_c/2$ [in] |
| $L_{\mathrm{rem}}$ | Remaining length for tapers $= L_{\mathrm{tot}} - L_c$ [in] |
| $s$ | Area scaling factor ($s \in \{\mathrm{ratio}_{A,\mathrm{in}}, \mathrm{ratio}_{A,\mathrm{out}}\}$) [$-$] |
| $H_{\mathrm{in}}$ | Inlet frame height $= H_c\sqrt{s}$ [in] |
| $W_{\mathrm{in}}$ | Inlet frame width $= W_c\sqrt{s}$ [in] |
| $A_{\mathrm{in}}$ | Inlet area $= sA_c$ [in$^2$] |
| $H_{\mathrm{out}}$ | Outlet frame height $= H_c\sqrt{s}$ [in] |
| $W_{\mathrm{out}}$ | Outlet frame width $= W_c\sqrt{s}$ [in] |
| $A_{\mathrm{out}}$ | Outlet area $= sA_c$ [in$^2$] |
| $\Delta H_{\mathrm{in}}$ | Height change from center to inlet frame [in] |
| $\Delta W_{\mathrm{in}}$ | Width change from center to inlet frame [in] |
| $\Delta H_{\mathrm{out}}$ | Height change from center to outlet frame [in] |
| $\Delta W_{\mathrm{out}}$ | Width change from center to outlet frame [in] |
| $L_{\mathrm{in}}$ | Inlet taper length [in] |
| $L_{\mathrm{out}}$ | Outlet taper length [in] |
| $L_{\mathrm{in,req}}$ | Inlet length required by $\alpha_{\max}$: $\sqrt{(\Delta H_{\mathrm{in}}/2)^2 + \Delta W_{\mathrm{in}}^2}/\tan\alpha_{\max}$ [in] |
| $L_{\mathrm{out,req}}$ | Outlet length from feasibility split $= L_{\mathrm{rem}} - L_{\mathrm{in,req}}$ [in] |
| $\alpha_{h,\mathrm{in}}$ | Inlet height half-angle $= \arctan\big((\Delta H_{\mathrm{in}}/2)/L_{\mathrm{in}}\big)$ [$^\circ$] |
| $\alpha_{w,\mathrm{in}}$ | Inlet width half-angle $= \arctan\big(\Delta W_{\mathrm{in}}/L_{\mathrm{in}}\big)$ [$^\circ$] |
| $\alpha_{\mathrm{res,in}}$ | Inlet resultant half-angle $= \arctan\Big(\sqrt{(\Delta H_{\mathrm{in}}/2)^2 + \Delta W_{\mathrm{in}}^2}/L_{\mathrm{in}}\Big)$ [$^\circ$] |
| $\alpha_{h,\mathrm{out}}$ | Outlet height half-angle $= \arctan\big((\Delta H_{\mathrm{out}}/2)/L_{\mathrm{out}}\big)$ [$^\circ$] |
| $\alpha_{w,\mathrm{out}}$ | Outlet width half-angle $= \arctan\big(\Delta W_{\mathrm{out}}/L_{\mathrm{out}}\big)$ [$^\circ$] |
| $\alpha_{\mathrm{res,out}}$ | Outlet resultant half-angle $= \arctan\Big(\sqrt{(\Delta H_{\mathrm{out}}/2)^2 + \Delta W_{\mathrm{out}}^2}/L_{\mathrm{out}}\Big)$ [$^\circ$] |