1 Verarbeitungspipeline für Impulsantworten

Systemparameter

Table 1: Konfigurationsparameter der Verarbeitungspipeline

Parameter	**Wert**
Abtastrate f_s	500 kHz
IR-Länge	30 ms (15 000 Samples)
Terzbänder	13 Bänder nach IEC 61260, 4 kHz–63 kHz
T30-Auswertungsfenster	−5 dB bis −35 dB
Temperatur T	20 °C
Relative Luftfeuchte LF	50 %
Luftdruck p	101,325 kPa
Empfängerpositionen	15 Positionen, 4×4 m-Raster
Schallquellenposition	Ursprung (0, 0)

1.1 Initialisierung und Datenladen

Die Verzeichnisstruktur des Projekts wird über `init_repo_paths()` initialisiert. Die Funktion bestimmt das Repository-Root anhand ihres eigenen Dateipfads, wechselt das Arbeitsverzeichnis entsprechend und fügt den Unterordner `functions/` dem MATLAB-Suchpfad hinzu.

```
thisFile    = mfilename('fullpath');
functionsDir = fileparts(thisFile);
repoRoot    = fileparts(functionsDir);

if ~strcmp(pwd, repoRoot)
    cd(repoRoot);
end
if ~contains(path, functionsDir)
    addpath(functionsDir);
end
```

Listing 1: Pfadinitialisierung via `init_repo_paths.m`

Die Messgeometrie umfasst 15 Empfängerpositionen auf einem gleichmäßigen 4×4 m-Raster sowie eine Schallquelle im Koordinatenursprung (0, 0). Der Abruf erfolgt über `get_geometry()`; die euklidischen Quell-Empfänger-Abstände werden pro Position berechnet und im Struct gespeichert:

```
coords = [
     1, 0,   1.2;   2, 0.3, 1.2;   3, 0.6, 1.2;   4, 1.2, 1.2;
     5, 0,   0.6;   6, 0.3, 0.6;   7, 0.6, 0.6;   8, 1.2, 0.6;
     9, 0,   0.3;  10, 0.3, 0.3;  11, 0.6, 0.3;  12, 1.2, 0.3;
    13, 0.3, 0;    14, 0.6, 0;    15, 1.2, 0
];
source_x = 0;  source_y = 0;

for i = 1:size(coords,1)
    x = coords(i,2);  y = coords(i,3);
```

```
    pos_info(i).distance = sqrt((x-source_x)^2 + (y-source_y)^2);
end
```

Listing 2: Geometriedefinition und Abstandsberechnung (`get_geometry.m`)

Die Rohdaten werden durch einen rekursiven Scan aller `.mat`-Dateien im Verzeichnis `dataraw/` erfasst:

```
dataDir = 'dataraw';
files   = dir(fullfile(dataDir, '*.mat'));
```

Listing 3: Datei-Scan des Rohdatenverzeichnisses (`step1_process_data.m`)

1.2 Pass 1 – IR-Extraktion und Vorverarbeitung

1.2.1 Metadaten-Parsing

Die Funktion `load_and_parse_file()` extrahiert aus dem Dateinamen die Messmetadaten: Variante, Positionsnummer und Messtyp (Quelle oder Mikrofon). Zwei reguläre Ausdrücke decken die im Datensatz vorkommenden Namensschemata ab. Sonderzeichen werden durch eine interne Hilfsfunktion in dateinamenkompatible Unterstriche überführt:

```
% Strategie A: Empfaengerposition am Dateinamenende
[tokens, startIdx] = regexp(fname, ...
    '(?i)[_,;\. ]+pos[_\- ]*([A-Za-z0-9_]+)$', ...
    'tokens', 'start', 'once');
if ~isempty(tokens)
    meta.position = sanitize_string(tokens{1});
    meta.variante = sanitize_string(fname(1:startIdx-1));
    meta.type = 'Mic';
    return;
end

% Strategie B: Quellmessung
[~, startIdx_src] = regexp(fname, '(?i)[_,;\. ]+quelle.*$', ...
    'tokens', 'start', 'once');
if ~isempty(startIdx_src)
    meta.position = '0';
    meta.variante = sanitize_string(fname(1:startIdx_src-1));
    meta.type = 'Source';
end
```

Listing 4: Regex-basiertes Metadaten-Parsing (`load_and_parse_file.m`)

1.2.2 IR-Extraktion

Die Extraktion der Impulsantwort aus dem MATLAB-Struct erfolgt über `extract_ir()`. Das Struct wird sequenziell nach Feldern mit festgelegten Namenskonventionen durchsucht (`RiR`, `RIR`, `ir`, `IR`, `aufn`, `audio`). Das erste qualifizierende Feld (numerisch, > 100 Elemente) wird nach `double` konvertiert und als Spaltenvektor zurückgegeben:

```
field_candidates = {'RiR', 'RIR', 'ir', 'IR', 'aufn', 'audio'};

for i = 1:length(field_candidates)
    fn = field_candidates{i};
```

```
    if isfield(S, fn) && isnumeric(S.(fn)) && numel(S.(fn)) > 100
        ir = double(S.(fn)(:));
        return;
    end
end
```

Listing 5: Feldsuche und Typ-Konvertierung (`extract_ir.m`)

1.2.3 DC-Offset-Entfernung

Der Gleichanteil wird durch Subtraktion des arithmetischen Mittelwerts über die gesamte IR beseitigt:

$$ir_{\text{clean}} = ir - \overline{ir} \tag{1}$$

Im Code erfolgt dies innerhalb von `truncate_ir()` über den Aufruf `process_ir_modifications(ir,`

1.2.4 Truncation

Die Funktion `truncate_ir()` identifiziert den signalrelevanten Abschnitt der IR in vier Teilschritten.

Peak-Detektion. Das globale Maximum des Absolutwerts bestimmt die Referenzamplitude und den zeitlichen Ankerpunkt:

```
ir_abs   = abs(ir);
[max_amp, idx_peak] = max(ir_abs);
```

Listing 6: Peak-Detektion (`truncate_ir.m`)

Onset-Detektion. Ausgehend vom Peak-Index wird rückwärts nach dem letzten Sample gesucht, dessen Betrag einen Schwellwert von 2 % der Peak-Amplitude unterschreitet. Dem so bestimmten Onset-Index werden 250 Samples als Vorläufer vorangestellt:

```
threshold_start = max_amp * 0.02;

idx_below  = find(ir_abs(1:idx_peak) < threshold_start, 1, 'last');
if isempty(idx_below), idx_onset = 1;
else, idx_onset = idx_below + 1; end

idx_start  = max(1, idx_onset - 250);    % 250-Sample-Vorlauf
```

Listing 7: Onset-Rückwärtssuche mit Pre-Roll (`truncate_ir.m`)

Rauschpegel-Schätzung. Der Rauschpegel wird aus dem letzten Zehntel der unkomprimierten IR geschätzt und dient als Grundlage für die SNR-Berechnung der truncierten Antwort:

```
noise_section = ir_abs(ceil(N*0.9):end);
mu_noise  = mean(noise_section);
std_noise = std(noise_section);
...
metrics.snr_db = 20*log10(rms(ir_trunc) / (std_noise + eps));
```

Listing 8: Rauschpegel-Schätzung (`truncate_ir.m`)

Feste Länge mit Zero-Padding. Bei aktiviertem Parameter `use_fixed_length=true` wird die IR auf exakt $f_s \cdot t_{fix} = 15\,000$ Samples beschnitten. Unterschreitet die verbleibende Signallänge diesen Wert, wird mit Nullen aufgefüllt:

```
if use_fixed_length
    target_samples = round(fixed_duration_s * fs);   % = 15000
end
[ir_trunc, metrics] = truncate_ir(ir_raw, target_samples);
...
% in truncate_ir():
idx_end = idx_start + fixed_length_samples - 1;
if idx_end <= N
    ir_trunc = ir(idx_start:idx_end);
else
    padding  = zeros(idx_end - N, 1);
    ir_trunc = [ir(idx_start:end); padding];
end
```

Listing 9: Beschneidung auf Festlänge mit Padding (`truncate_ir.m`)

Sammlung von `FS_global_peak`. Über alle im ersten Durchlauf verarbeiteten IRs wird der maximale Absolutwert kumuliert:

```
ir_max         = max(abs(ir_trunc));
FS_global_peak = max(FS_global_peak, ir_max);
```

Listing 10: Akkumulation der Peak-Referenz (`step1_process_data.m`)

1.3 Pass 2 – Spektralanalyse und Energie-Referenzbestimmung

1.3.1 Terzband-Spektrum

Die Funktion `calc_terz_spectrum()` berechnet Terzbandpegel und Breitband-Summenpegel in fünf Einzelschritten.

FFT mit Zero-Padding. Die truncierte IR wird auf die nächste Zweierpotenz erweitert und fouriertransformiert:

```
N     = length(ir);
N_fft = 2^nextpow2(N);
X     = fft(ir, N_fft);
freqs = (0:N_fft-1) * (fs / N_fft);
```

Listing 11: FFT mit Zero-Padding (`calc_terz_spectrum.m`)

Luftabsorptionskorrektur. Die Funktion `airabsorb()` berechnet den frequenzabhängigen Absorptionskoeffizienten $\alpha(f)$ nach ISO 9613-1 und liefert den linearen Korrekturfaktor $A_{lin}(f)$. Der Absorptionskoeffizient ergibt sich aus der molekularen Relaxation von Sauerstoff und Stickstoff:

$$\alpha(f) = 8{,}686\, f^2 \left[1{,}84 \times 10^{-11} \left(\frac{p_a}{p_r}\right)^{-1} \left(\frac{T}{T_0}\right)^{1/2} + \left(\frac{T}{T_0}\right)^{-5/2} \left(\frac{0{,}01275\, e^{-2239{,}1/T}}{f_{rO} + f^2/f_{rO}} + \frac{0{,}1068\, e^{-3352{,}0/T}}{f_{rN} + f^2/f_{rN}} \right) \right] \tag{2}$$

Die Dämpfung über die Distanz s und der lineare Korrekturfaktor berechnen sich zu:

$$A_{\mathrm{dB}}(f) = \frac{\alpha(f) \cdot s}{100}, \qquad A_{\mathrm{lin}}(f) = 10^{A_{\mathrm{dB}}(f)/20} \tag{3}$$

Die Korrektur im Frequenzbereich lautet:

$$X_{\mathrm{korr}}(f) = X(f) \cdot A_{\mathrm{lin}}(f) \tag{4}$$

```
% --- airabsorb.m (Auszug): Koeffizientenberechnung ---
alpha = 8.686 .* f.^2 .* (1.84e-11*(p_a/p_r)^(-1)*(T_kel/T_0)^(0.5) + ...
    (T_kel/T_0)^(-2.5) .* ( ...
        0.01275 .* exp(-2239.1./T_kel) ./ (f_rO + (f.^2./f_rO)) + ...
        0.1068  .* exp(-3352.0./T_kel) ./ (f_rN + (f.^2./f_rN)) ...
    ));
A_dB = alpha * s/100;
A_lin = 10.^(A_dB/20);

% --- calc_terz_spectrum.m: Anwendung auf Spektrum ---
[~, A_lin, ~] = airabsorb(101.325, fs, N_fft, T, LF, dist);
X = X .* A_lin(:);
```

Listing 12: Luftabsorptionskorrektur im Frequenzbereich (`calc_terz_spectrum.m`, `airabsorb.m`)

Einseitige Energiedichte. Das einseitige Leistungsdichtespektrum ergibt sich zu:

$$S_E(f) = \frac{|X(f)|^2}{N} \tag{5}$$

Terzbandgrenzen nach IEC 61260. Die 13 Mittenfrequenzen (4 kHz–63 kHz) folgen der Oktavenbasis 10. Die Bandgrenzen berechnen sich zu:

$$f_{\mathrm{low}} = f_c \cdot 10^{-1/20}, \qquad f_{\mathrm{high}} = f_c \cdot 10^{+1/20} \tag{6}$$

```
f_mitten = [4000 5000 6300 8000 10000 12500 ...
            16000 20000 25000 31500 40000 50000 63000];
indices  = 6:18;
f_exact  = 1000 * 10.^(indices/10);    % Exakte IEC-Mittenfrequenzen

X_mag_sq = (abs(X).^2) / N;            % Einseitige Energiedichte
energy_sum = 0;

for k = 1:length(f_mitten)
    fc  = f_exact(k);
```

```
    fl  = fc * 10^(-1/20);
    fu  = fc * 10^( 1/20);
    idx = freqs >= fl & freqs <= fu;
    if any(idx)
        band_energy = sum(X_mag_sq(idx));
        energy_sum  = energy_sum + band_energy;
        L_dBFS(k)   = 10*log10(band_energy / (FS_global^2 + eps));
    end
end
```

Listing 13: Terzbandgrenzen und Bandintegration (`calc_terz_spectrum.m`)

Pegelberechnung. Terzbandpegel und Breitband-Summenpegel in dBFS:

$$L_{\text{Terz,k}} = 10 \cdot \log_{10}\left(\frac{E_{\text{Band},k}}{FS_{\text{ref}}^2}\right) \tag{7}$$

$$L_{\text{sum}} = 10 \cdot \log_{10}\left(\frac{\sum_k E_{\text{Band},k}}{FS_{\text{ref}}^2}\right) \tag{8}$$

1.3.2 Nachhallzeit T30

Die Funktion `calc_rt60_spectrum()` berechnet die frequenzabhängige Nachhallzeit für jedes Terzband.

Butterworth-Bandpassfilterung. Ein Butterworth-Bandpassfilter 4. Ordnung wird mit den Terzband-Grenzfrequenzen entworfen und nullphasig (`filtfilt`) auf die IR angewendet:

```
fc = f_exact(k);
fl = fc * 10^(-1/20);
fu = fc * 10^( 1/20);
[b, a]  = butter(4, [fl fu]/(fs/2), 'bandpass');
filt_ir = filtfilt(b, a, ir);
```

Listing 14: Nullphasige Bandpassfilterung (`calc_rt60_spectrum.m`)

Schroeder-Integration. Die Energieverfall-Kurve (EDC) entsteht durch Rückwärtsintegration:

$$\text{EDC}(t) = \sum_{\tau=t}^{\infty} h^2(\tau) \tag{9}$$

Lineare Regression und Extrapolation. Im Bereich $-5\,\text{dB}$ bis $-35\,\text{dB}$ der normalisierten EDC wird eine lineare Regression (Methode der kleinsten Quadrate) durchgeführt. Die Regressionssteigung $\hat{s}$ wird auf $-60\,\text{dB}$ extrapoliert:

```
edc_db     = calc_edc(filt_ir);        % normierte EDC in dB
idx_start  = find(edc_db <= -5,  1, 'first');
idx_end_r  = find(edc_db(idx_start:end) <= -35, 1, 'first');
```

```
idx_end    = idx_start + idx_end_r - 1;

y_segment  = edc_db(idx_start:idx_end);
x_segment  = (0:length(y_segment)-1)' / fs;
p          = polyfit(x_segment, y_segment, 1);
slope      = p(1);                          % Abklingrate [dB/s]

if slope < 0
    T_meas = -60 / slope;                   % Extrapolation auf -60 dB
end
```

Listing 15: Regression und T30-Extrapolation (`calc_rt60_spectrum.m`)

Luftabsorptionskorrektur der Nachhallzeit. Der frequenzabhängige Einfluss der Luftabsorption wird subtraktiv kompensiert:

$$\frac{1}{T_{\text{Raum}}} = \frac{1}{T_{\text{gemessen}}} - \frac{1}{T_{\text{Luft}}} \tag{10}$$

Dabei ergibt sich T_{Luft} aus der Schallgeschwindigkeit c und dem Absorptionskoeffizienten m_{Luft} [dB/m]:

```
m_air  = alpha_val / 100;               % Daempfung in dB/m
c      = 343;
T_air  = 60.0 / (c * m_air);
if T_meas < T_air
    t60_vals(k) = (T_meas * T_air) / (T_air - T_meas);
else
    t60_vals(k) = T_meas;
end
```

Listing 16: Luftabsorptionskorrektur der Nachhallzeit (`calc_rt60_spectrum.m`)

1.3.3 Globale Energiereferenz

Aus allen verarbeiteten IRs wird die maximale Breitband-Energie bestimmt:

$$FS_{\text{global\_energy}} = \sqrt{\max_{\text{IRs}} \sum_{k} E_{\text{Band},k}} \tag{11}$$

Die finale Vollaussteuerungsreferenz ergibt sich als Maximum beider Durchläufe:

$$FS_{\text{global\_final}} = \max(FS_{\text{global\_peak}},\ FS_{\text{global\_energy}}) \tag{12}$$

```
% Pass 2: Spektren mit FS_global=1 -> lineare Energien
[L_terz_tmp, L_sum_tmp, ~] = calc_terz_spectrum(ir_trunc, fs, 1.0, ...
                                                dist, T, LF);
energy_sum     = 10^(L_sum_tmp / 10);
max_energy_sum = max(max_energy_sum, energy_sum);
...
FS_global_energy = sqrt(max_energy_sum);
FS_global_final  = max(FS_global_peak, FS_global_energy);
```

Listing 17: Bestimmung der globalen Referenz (`step1_process_data.m`)

Diese doppelte Referenzbildung stellt sicher, dass weder der maximale Zeitbereichs-Peak noch die maximale spektrale Gesamtenergie den Wert 0 dBFS übersteigt.

1.4 Pass 3 – Finale Pegelberechnung

Im dritten Durchlauf werden sämtliche Terzbandpegel mit $FS_{\text{global\_final}}$ neu berechnet:

$$L = 10 \cdot \log_{10}\left(\frac{E}{FS^2_{\text{global\_final}}}\right) \tag{13}$$

Pegel oberhalb 0 dBFS werden auf 0 dBFS gedeckelt:

```
L_terz = 10 * log10((E_terz + eps) / (FS_global_final^2));
L_sum  = 10 * log10((E_sum  + eps) / (FS_global_final^2));
L_terz = min(L_terz, 0);              % Deckelung bei 0 dBFS
L_sum  = min(L_sum,  0);
```

Listing 18: Finale dBFS-Berechnung mit Deckelung (`step1_process_data.m`)

Die Ergebnisse werden in einem strukturierten `Result`-Struct zusammengefasst und als `.mat`-Datei gespeichert:

```
Result.meta.fs             = fs;
Result.meta.FS_global_used = FS_global_final;
Result.meta.T              = proc.T;
Result.meta.LF             = proc.LF;

Result.time.ir             = proc.ir_trunc;
Result.time.metrics        = proc.metrics;     % SNR, Energie, Indizes

Result.freq.f_center       = proc.f_center;
Result.freq.terz_dbfs      = L_terz;
Result.freq.sum_level      = L_sum;
Result.freq.t30            = proc.t30_vals;

saveName = sprintf('Proc_%s_Pos%s.mat', meta.variante, meta.position);
save(fullfile(procDir, saveName), 'Result');
```

Listing 19: Aufbau des Result-Structs und Speicherung (`step1_process_data.m`)

1.5 Zusammenfassung und Mittelwertbildung

Aus allen Einzelmessungen wird eine Summary-Tabelle erstellt (Felder: Variante, Position, Summenpegel, SNR) und in zwei Formaten persistiert:

```
summary_table = cell2table(summary_data, ...
    'VariableNames', {'Variante','Position','SumLevel','SNR','File'});
save(fullfile(procDir, 'Summary_Database.mat'), 'summary_table');
writetable(summary_table, fullfile(procDir, 'Summary.xlsx'));
```

Listing 20: Erstellung der Summary-Tabelle (`step1_process_data.m`)

1.5.1 Positions-Mittelwertbildung

Pro Variante wird ein positionsgemitteltes Ergebnis berechnet. Der Mittelungsprozess beschränkt sich auf gültige Empfängerpositionen (numerische Indizes 1–15); Quellpositionen sowie geometrische Zwischenpunkte werden ausgeschlossen:

```
for j = 1:height(summary_table)
    pos    = summary_table.Position{j};
    posNum = str2double(pos);
    if ~isnan(posNum) && posNum >= 1 && posNum <= 15
        mask_valid(j) = true;
    end
end
```

Listing 21: Filterung auf gueltige Empfaengerpositionen (`step1_process_data.m`)

Die Mittelung erfolgt im linearen Energiebereich; die Rückrechnung in die logarithmische Darstellung lautet:

$$L_{\text{avg}} = 10 \cdot \log_{10}\left(\frac{\bar{E}}{FS^2_{\text{global\_final}}}\right), \qquad \bar{E} = \frac{1}{N}\sum_{n=1}^{N} E_n \tag{14}$$

```
sum_E_terz  = 0;
sum_E_total = 0;

for k = 1:n
    D = load(fullfile(procDir, subset.File{k}));
    E_terz = 10.^(D.Result.freq.terz_dbfs / 10);
    E_terz(~isfinite(E_terz)) = 0;
    sum_E_terz  = sum_E_terz  + E_terz;
    E_tot       = 10^(D.Result.freq.sum_level / 10);
    if ~isfinite(E_tot), E_tot = 0; end
    sum_E_total = sum_E_total + E_tot;
end

Result.freq.terz_dbfs = 10 * log10((sum_E_terz/n) + eps);
Result.freq.sum_level = 10 * log10((sum_E_total/n) + eps);

saveName = sprintf('Proc_%s_Average.mat', variante);
save(fullfile(procDir, saveName), 'Result');
```

Listing 22: Energetische Mittelwertbildung und Rueckrechnung (`step1_process_data.m`)

Dieses Verfahren gewährleistet eine physikalisch korrekte Energieaddition, da die logarithmische Mittelung zu systematischen Unterschätzungen führen würde. Die gemittelten Ergebnisse werden unter `processed/Proc_[Variante]_Average.mat` abgelegt und besitzen dieselbe Struct-Struktur wie die Einzelpositions-Ergebnisse.