SEM EBIC Analysis Software: Technical Report

Panteha Farhadi

September 26, 2025

Abstract

This report documents a comprehensive Python software package for analyzing Scanning Electron Microscopy (SEM) Electron Beam Induced Current (EBIC) measurements. The software processes multi-frame TIFF images with embedded metadata, extracts current maps, detects p-n junctions, and calculates diffusion lengths and depletion widths using exponential fitting algorithms. The graphical interface enables interactive visualization and analysis.

Contents

1 Introduction to EBIC Physics

Electron Beam Induced Current (EBIC) is a characterization technique used in semiconductor physics to study minority carrier diffusion lengths and junction properties. When an electron beam strikes a semiconductor sample, it generates electron-hole pairs. In the presence of electric fields (such as near p-n junctions), these carriers separate and generate a measurable current. The current profile across a junction follows an exponential decay, from which key parameters can be extracted:

- **Diffusion length (L)**: Characteristic distance minority carriers travel before recombination
- **Depletion width (W)**: Region around the junction where electric fields dominate
- $1/\lambda$: Inverse decay constant obtained from exponential fitting of EBIC profiles

The relationship is described by:

$$I(x) = I_0 \exp\left(-\frac{|x - x_j|}{L}\right) \tag{1}$$

where x_j is the junction position and L is the diffusion length.

1.1 Voltage Dependence of Depletion Width

The depletion width W in a p-n junction depends on the applied bias voltage V through the following relationship:

$$W(V) = \sqrt{\frac{2\epsilon}{q}\left(\frac{1}{N_A} + \frac{1}{N_D}\right)(V_{bi} - V)} \tag{2}$$

where:

- $W(V)$ is the voltage-dependent depletion width
- ϵ is the permittivity of the semiconductor material
- q is the elementary charge (1.602×10^{-19} C)
- N_A and N_D are the acceptor and donor concentrations, respectively
- V_{bi} is the built-in potential of the junction
- V is the applied bias voltage (negative for reverse bias)

For a one-sided abrupt junction where $N_A \gg N_D$ or vice versa, the equation simplifies to:

$$W(V) = \sqrt{\frac{2\epsilon}{qN}(V_{bi} - V)} \tag{3}$$

where N is the doping concentration on the lightly doped side.

2 Software Architecture Overview

The software consists of four main components:

1. TIFF metadata extraction and current conversion
2. Junction detection algorithms
3. Diffusion length extraction via exponential fitting
4. Interactive GUI for visualization and analysis
5. Iteration over multiple images for bias analysis using pixel matching to track profiles accurately

3 Core Components

3.1 Metadata Extraction (`Metadata` Class)

```python
class Metadata:
    def __init__(self, xml_path):
        self.xml_path = xml_path
        self.data = self._parse_metadata()

    def _parse_metadata(self):
        ns = {'rdf': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
              'image': 'http://ns.pointelectronic.com/Image/1.0/',
              'efa': 'http://ns.pointelectronic.com/EFA/1.0/'}
        # ... XML parsing logic
```

Listing 1: Metadata extraction from XML

This class extracts the following comprehensive set of SEM parameters from XML metadata:

3.1.1 Spatial Calibration Parameters

- `PixelSizeX`, `PixelSizeY`: Physical size of each pixel (meters)
- `PixelUnit`: Unit of measurement for pixel dimensions

3.1.2 Amplifier and Signal Processing Parameters

- `Contrast`, `Brightness`: Image contrast and brightness settings
- `EffectiveAmpGain`, `Amplification`: Amplifier gain factors
- `InputOffset`, `OutputOffset`: Signal offset corrections
- `InverseEnabled`: Whether signal inversion is applied

3.1.3 Beam Parameters

- `BeamVoltage`: Electron beam acceleration voltage (kV)
- `BeamCurrent`: Electron beam current (nA)
- `WorkingDistance`: Distance from objective lens to sample (mm)

3.1.4 Scan Parameters

- `ScanRotation`: Image rotation angle (degrees)
- `ScanSpeed`: Beam scanning speed
- `ScanDwellTime`: Time per pixel acquisition (s)

3.1.5 Detector Parameters

- `DetectorType`: Type of EBIC detector used
- `DetectorGain`: Detector gain setting

3.1.6 Image Acquisition Parameters

- `ImageWidth`, `ImageHeight`: Image dimensions in pixels
- `BitDepth`: Bit depth of the acquired image (typically 16-bit)
- `FrameCount`: Number of frames averaged

3.1.7 Sample and Stage Parameters

- `SampleBias`: Applied bias voltage during measurement (V)
- `StagePositionX`, `StagePositionY`, `StagePositionZ`: Sample stage coordinates

3.1.8 Instrument Identification

- `InstrumentModel`, `InstrumentSerial`: SEM instrument identification

3.1.9 Calibration Parameters

- `CurrentSensitivity`: Current measurement sensitivity
- `VoltageRange`: Voltage measurement range

These parameters are essential for accurate current conversion, spatial calibration, and reproducible EBIC measurements. The metadata extraction ensures proper scaling from pixel values to physical units and provides complete documentation of measurement conditions.

3.2 Current Map Conversion (CurrentMap Class)

The current conversion process involves two main steps: first converting pixel values to voltage measurements, then applying the instrument-specific calibration equation to obtain current values.

3.2.1 Pixel to Voltage Conversion

Raw 16-bit pixel values (0-65535) are converted to voltage using a linear scaling relationship:

$$V_{\text{measured}} = \left(\frac{\text{pixels}}{65535}\right) \times \text{scale} + \text{offset} \tag{4}$$

where scale = 1 mV and offset = -0.5 mV are instrument-specific calibration constants.

3.2.2 Voltage to Current Conversion

The fundamental equation used for converting measured voltage to current is derived from the SEM instrument's signal chain calibration:

$$I = \frac{\left(\frac{V_{\text{measured}} - O}{C}\right) \pm I_{\text{offset}}}{G} \times 10^9 \tag{5}$$

```python
class CurrentMap:
    def _compute_current(self, pixels):
        # Extract calibration parameters from metadata
        C = self.metadata['Contrast']            # Contrast setting
        G = self.metadata['EffectiveAmpGain']  # Amplifier gain
        O = self.metadata['OutputOffset']       # Output offset voltage
        I_offset = self.metadata['InputOffset'] # Input offset correction
        inv = self.metadata['InverseEnabled']   # Signal inversion flag

        # Step 1: Convert 16-bit pixels to voltage (mV)
        scale = 1     # mV full-scale range
        offset = -0.5 # mV zero offset
        voltage = (pixels / 65535) * scale + offset

        # Step 2: Apply instrument calibration equation
        if inv:
            # For inverted signal: I = [((V - O)/C) + I_offset] / G * -1e9 nA
            return (((voltage - O) / C) + I_offset) / G * -1e9
        else:
            # For normal signal: I = [((V - O)/C) - I_offset] / G * +1e9 nA
            return (((voltage - O) / C) - I_offset) / G * +1e9
```

Listing 2: Current conversion algorithm

The conversion algorithm accounts for all instrument-specific parameters:

- **Contrast (C)**: Amplifies or attenuates the signal range
- **Effective Amplifier Gain (G)**: Overall gain of the signal chain
- **Output Offset (O)**: DC offset voltage correction

- **Input Offset (I_offset)**: Additional offset correction term
- **Inversion Mode**: Accounts for signal polarity inversion

The final result is expressed in nanoamperes (nA) by multiplying by 10^9, providing physically meaningful current values for EBIC analysis. This calibrated current map forms the basis for all subsequent junction detection and diffusion length calculations.

3.2.3 SEMProcessor Class

The `SEMProcessor` class serves as the central coordinator that integrates all analysis components and manages the complete EBIC processing workflow:

```python
class SEMProcessor:
    def __init__(self, config):
        self.config = config
        self.metadata_parser = Metadata()
        self.current_converter = CurrentMap()
        self.junction_detector = JunctionAnalyzer()
        self.diffusion_extractor = DiffusionLengthExtractor()
        self.multi_image_analyzer = MultiImageAnalyzer()
```

Listing 3: Main processor class orchestrating the analysis pipeline

The `SEMProcessor` class provides:

- **Unified Interface**: Single entry point for all EBIC analysis operations
- **Workflow Management**: Coordinates the sequential execution of metadata extraction, current conversion, junction detection, and diffusion analysis
- **Error Handling**: Implements robust error handling and validation across all processing stages
- **Batch Processing Support**: Handles both single-image analysis and multi-image bias series
- **Result Aggregation**: Compiles results from all components into comprehensive output reports

This orchestrator class ensures that all analysis components work together seamlessly, providing a streamlined workflow from raw TIFF data to final EBIC parameter extraction.

3.3 Junction Detection Algorithms

3.3.1 JunctionAnalyzer Class Overview

The `JunctionAnalyzer` class is a crucial component of the EBIC analysis framework. Its main role is to automatically and accurately detect the p-n junction line within a processed EBIC map. The class integrates filtering, edge detection, line fitting, coordinate transformation, and validation into a single robust workflow.

It contains methods for:

- **Preprocessing:** Filtering to reduce noise while keeping junction edges sharp.
- **Junction Detection:** Locating the junction using Canny edge detection and gradient-based analysis.
- **Post-processing:** Refining the noisy detection points into a smooth line using fitting techniques.
- **Coordinate Mapping:** Translating local ROI detections into global image coordinates.
- **Evaluation:** Quantitatively comparing the automatic detection with the manual reference.

3.3.2 Preprocessing with Bilateral Filtering

The preprocessing stage applies a Bilateral Filter to the region of interest (ROI).

- **Purpose:** To reduce random noise while preserving the sharp junction edge.
- **Mechanism:** The bilateral filter considers both pixel intensity and spatial distance, smoothing flat areas without blurring steep gradients.

```python
def _apply_preprocessing_filter(self, roi):
    """
    Applies a Bilateral Filter to smooth noise while preserving sharp edges.
    """
    # Convert the ROI to 8-bit for OpenCV processing
    normalized_roi = cv2.normalize(roi, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    # Apply bilateral filter with optimized parameters
    return cv2.bilateralFilter(normalized_roi, d=9, sigmaColor=75, sigmaSpace=9)
```

Listing 4: Bilateral filter preprocessing

3.3.3 Canny Edge Detection with Otsu Thresholding

The core junction detection relies on column-wise edge analysis.

- **Adaptive Thresholding:** Otsu's method automatically determines thresholds for Canny.

- **Canny Edge Detection:** Extracts strong edge features while suppressing weak or false edges.
- **Gradient-Based Selection:** Ensures that the selected edge corresponds to the steepest EBIC transition.

```python
def _detect_junction_canny(self, roi):
    """
    Detect junction using Canny edge detection with Otsu's adaptive thresholding.
    """
    H, W = roi.shape
    detected_points = np.zeros((W, 2), dtype=float)

    # Convert ROI to 8-bit for OpenCV processing
    roi_8bit = roi.astype(np.uint8)

    # Otsu's method for automatic threshold determination
    otsu_val, _ = cv2.threshold(roi_8bit, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Set Canny thresholds based on Otsu's result
    high_threshold = float(otsu_val)
    low_threshold = 0.5 * high_threshold

    # Perform Canny edge detection
    edges = cv2.Canny(roi_8bit, low_threshold, high_threshold)
    .
    .
    .

    return detected_points
```

Listing 5: Canny edge detection

3.3.4 Post-Processing for Junction Modeling

The detected points are refined into a continuous model.

Linear Fitting Applies linear regression to approximate the junction with a straight line.

```python
def _fit_line_postprocessing(self, points):
    """
    Post-process detected points by fitting a linear model.
    """
    x_coords = points[:, 0]
    y_coords = points[:, 1]

    coeffs = np.polyfit(x_coords, y_coords, deg=1)
    fitted_line = np.polyval(coeffs, x_coords)

    return np.column_stack((x_coords, fitted_line))
```

Listing 6: Line fitting for junction approximation

Spline Fitting Uses cubic splines for more flexible modeling of curved junctions. **This method is not used for planar junctions. This is a good idea for curved junctions which we don't want to loose their structure while fitting**

```python
def _fit_spline_postprocessing(self, points, smoothing=1.0):
    """
    Post-process detected points by fitting a spline model.
    """
    x_coords = points[:, 0]
    y_coords = points[:, 1]

    spline = UnivariateSpline(x_coords, y_coords, s=smoothing)
    fitted_spline = spline(x_coords)

    return np.column_stack((x_coords, fitted_spline))
```

Listing 7: Spline fitting for flexible junction modeling

3.3.5 Coordinate Transformation

The ROI-based detection is translated into global image coordinates.

```python
def _map_detected_to_image_coords(self, detected_points, roi_coords):
    """
    Map detected points from ROI-local to global image coordinates.
    """
    x_offset, y_offset = roi_coords
    mapped_points = detected_points.copy()
    mapped_points[:, 0] += x_offset
    mapped_points[:, 1] += y_offset
    return mapped_points
```

Listing 8: Mapping detected junction to image coordinates

3.3.6 Validation Metrics

The automatic detection is validated against manual junctions using quantitative error metrics.

```python
def _compare_with_manual(self, detected_points, manual_points):
    """
    Compare automatic and manual junction detection results.
    """

    return {
        "mean_error": mean_error,
        "max_error": max_error,
        "rmse": rmse
    }
```

Listing 9: Junction detection validation

3.4 Diffusion Length Extraction Algorithms

3.4.1 DiffusionLengthExtractor Class Overview

The `DiffusionLengthExtractor` class is a core component of the SEM-EBIC analysis software, designed to extract key physical parameters like diffusion length and depletion width from EBIC line profiles. The class's strength lies in its robust fitting algorithm, which is specifically engineered to handle noisy, real-world data and produce stable, reliable results through an innovative iterative fitting approach.

Class Purpose and Key Innovations

- **Iterative Stabilization**: Multi-pass fitting until parameter convergence
- **Head Search Algorithm**: Finding optimal starting points away from junction influence
- **Tail Truncation**: Systematic removal of noise-dominated regions
- **Convergence Detection**: Automated stability assessment for reliable results
- **Physical Parameter Extraction**: Diffusion length and depletion width calculation

3.4.2 Iterative Fitting Architecture

The core algorithm implements a sophisticated iterative process that addresses fundamental challenges in EBIC data analysis:

1. **Stage 1 - Head Search**: Progressive window shifting to find stable exponential decay regions
2. **Stage 2 - Tail Truncation**: Systematic removal of noisy tail regions
3. **Convergence Loop**: Repeated application until parameter stabilization
4. **Final Parameter Extraction**: Physical quantity calculation from stabilized fits

3.4.3 Stage 1: Head Search for Stable Starting Point

The `fit_profile_sides` method implements the critical first stage of finding robust starting points for exponential fitting.

```python
def fit_profile_sides(self, x_vals, y_vals, intersection_idx,
    max_shifts=20):
     """
     Stage 1: Find stable starting point through progressive window
    shifting.
     """

     return stable_left_index, stable_right_index, left_results,
    right_results
```

Listing 10: Head search with progressive window shifting

Head Search Methodology

- **Progressive Shifting**: Moves fitting window incrementally away from junction peak
- **Exponential Model**: Fits standard minority carrier diffusion equation
- **Stability Monitoring**: Tracks diffusion length changes between shifts
- **Convergence Criterion**: Stops when changes fall below pixel-size tolerance
- **Physical Rationale**: Ensures fitting captures bulk material behavior, free from junction region influence

Key Parameters

- `tolerance = self.pixel_size`: Convergence threshold based on physical resolution
- `max_shifts = 20`: Maximum iterations to prevent infinite loops
- `min_points = 10`: Minimum data points required for reliable fitting

3.4.4 Stage 2: Tail Truncation for Noise Mitigation

The `_truncate_tail_and_fit` method refines the fit by systematically removing noisy tail regions.

```python
def _truncate_tail_and_fit(self, x_vals, y_vals, stable_start_idx, side
    ='left'):
    """
    Stage 2: Refine fit by progressively truncating noisy tail region.
    """

        except (RuntimeError, ValueError):
            continue

    # Select optimal result balancing fit quality and data utilization
    if tail_results:
        best_idx = np.argmax([r['r_squared'] * (0.3 + 0.7*r['
    truncation_ratio'])
                                for r in tail_results])
        return tail_results[best_idx]
    else:
        return None
```

Listing 11: Tail truncation for noise reduction

Tail Truncation Strategy

- **Progressive Reduction**: Systematically shortens data from 100% to 50%
- **Quality Assessment**: Uses R^2 values to evaluate fit goodness
- **Balanced Selection**: Considers both statistical quality and data utilization
- **Noise Elimination**: Removes low-SNR regions that distort exponential fits

3.4.5 Iterative Convergence Loop

The `fit_profile_sides_iterative` method orchestrates the complete iterative process.

```python
def fit_profile_sides_iterative(self, x_vals, y_vals, intersection_idx,
    max_iterations=5):
    """
    Main iterative fitting loop until parameter convergence.
    """
    prev_left_length = None
    prev_right_length = None
    convergence_tolerance = 1e-3  # Relative tolerance for convergence

    return {
        'left_diffusion_length': left_history[-1] if left_history else
   None,
        'right_diffusion_length': right_history[-1] if right_history
   else None,
        'depletion_width': depletion_width,
        'iterations_used': iteration + 1,
        'convergence_history': (left_history, right_history)
    }
```

Listing 12: Iterative convergence algorithm

Convergence Algorithm Features

- **Iterative Refinement**: Repeated application of head-tail process
- **Relative Tolerance**: Uses relative changes for convergence detection
- **History Tracking**: Maintains complete iteration history for analysis
- **Robust Handling**: Continues processing despite individual iteration failures
- **Convergence Detection**: Automatic stopping when parameters stabilize

Convergence Criteria

- **Tolerance Setting**: `convergence_tolerance = 1e-3` (0.1% relative change)
- **Maximum Iterations**: `max_iterations = 5` prevents infinite loops
- **Bilateral Convergence**: Requires both left and right sides to stabilize
- **Physical Validation**: Ensures results represent true material properties

3.4.6 Physical Model and Parameter Extraction

Exponential Decay Model The core physical model describing minority carrier diffusion away from the junction:

$$I(x) = A \cdot \exp\left(-\frac{|x - x_j|}{L}\right) + B \tag{6}$$

Where:

- A: Amplitude representing maximum EBIC current at junction
- L: Diffusion length (characteristic decay distance)
- x_j: Junction position
- B: Background offset accounting for dark current

```python
def _exponential_model(self, x, a, b, c):
    """
    Exponential decay model: y = a * exp(-b * x) + c
    Corresponds to physical equation: I(x) = A * exp(-x/L) + B
    where L = 1/b (diffusion length)
    """
    return a * np.exp(-b * x) + c

def _calculate_depletion_width(self, x_vals, left_index, right_index):
    """
    Calculate depletion width from stable fitting boundaries.
    """
    if left_index is None or right_index is None:
        return None

    left_boundary = x_vals[left_index]
    right_boundary = x_vals[right_index]
    depletion_width = abs(right_boundary - left_boundary) * self.
  pixel_size_m

    return depletion_width
```

Listing 13: Exponential model implementation

3.4.7 Parameter Initialization for Fitting

A critical part of the fitting process is providing good initial guesses for the fitting parameters. The `_fit_falling` helper function, which handles the actual curve fitting, intelligently sets these initial guesses (`p0`) based on the input data to ensure a robust and fast convergence. The `scipy.optimize.curve_fit` function then uses these guesses to find the best-fit parameters.

The exponential model being fitted is:

$$y = A \cdot e^{-\lambda(x-x_0)} + y_0$$

Initialization Strategy

- y_0 **(Offset)**: Initialized as the last value of the profile, representing the background signal.
- A **(Amplitude)**: Estimated as the difference between the first data point (peak) and the last data point (baseline).
- λ **(Decay Constant)**: Set as the reciprocal of the total range of the x-values, providing a scale-aware starting guess.

- x_0 **(Shift)**: Initialized as the first x-value in the data, corresponding to the junction peak.

This initialization scheme ensures fast and stable convergence even in the presence of noisy experimental EBIC profiles.

3.5 Interactive GUI for Visualization and Analysis

3.5.1 Interactive GUI (`SEMViewer` Class)

The GUI provides comprehensive visualization and interaction capabilities:

```
def _on_click(self, event):  # Line drawing
def _on_scroll(self, event):  # Zoom functionality
def _show_perpendicular_input(self, event):  # Profile generation
def _fit_profiles(self, event):  # Analysis triggering
```

Listing 14: GUI interaction handlers

Key features include:

- Multi-frame navigation between SEM and current maps
- Interactive line drawing for profile extraction
- Zoom and pan functionality with scroll and right-click drag
- Real-time perpendicular profile generation
- Junction detection visualization
- Diffusion length fitting and results display

3.6 Iteration over Multiple Images for Bias Analysis

3.6.1 Overview

The software implements a sophisticated pipeline for aligning multiple Scanning Electron Microscope (SEM) images and mapping key features such as p-n junctions and perpendicular profiles across different measurement conditions. This capability is essential for analyzing voltage-dependent effects and ensuring consistent region-of-interest tracking.

3.6.2 Global Image Alignment

map_all_pixels: Multi-Dataset Alignment Orchestration This high-level function manages the complete alignment process for multiple datasets:

```
def map_all_pixels(self, dataset_dict, window_size, max_disp, ref):
    # Histogram matching for contrast normalization
    matched_dict = {}
    for key, (pix, cur) in dataset_dict.items():
        matched_pix[0] = match_histograms(pix[0], image_ref)
        matched_dict[key] = (matched_pix, cur)

    # Compute mapping for each dataset
    for key, (pix, cur) in matched_dict.items():
        mapping = self.map_pixels(image_ref, pix[0],
                                 window_size=window_size,
                                 search_radius=max_disp)
        mapping_save[key] = mapping
```

Listing 15: Multi-dataset alignment orchestration

- **Histogram Matching**: Normalizes brightness and contrast variations across different measurements
- **Batch Processing**: Iterates through each dataset for comprehensive alignment
- **Result Storage**: Saves alignment mappings for subsequent analysis

map_pixels: Pixel-Level Correspondence The core alignment algorithm implements robust pixel matching using Structural Similarity Index (SSIM):

```
def map_pixels(img1, img2, window_size=11, search_radius=15):
    for i in range(height):
        for j in range(width):
            # Extract reference window
            window1 = img1_padded[pi-pad:pi+pad+1, pj-pad:pj+pad+1]

            # Search for best match in target image
            for x in range(min_i, max_i):
                for y in range(min_j, max_j):
                    window2 = img2_padded[x-pad:x+pad+1, y-pad:y+pad+1]
                    score = ssim(window1, window2, data_range=
    data_range)

                    if score > best_score:
                        best_score = score
                        best_pos = (x-pad, y-pad)
```

Listing 16: SSIM-based pixel matching

- **Structural Similarity Index (SSIM)**: Advanced similarity metrics considering luminance, contrast, and structure
- **Brute-Force Search**: Comprehensive search within defined radius for optimal matches
- **Padding Handling**: Uses reflection padding to handle image boundaries

3.6.3 Line Alignment

match_line_only: Optimized Feature Mapping This function provides computational efficiency for line-based feature tracking:

```
def match_line_only(self, image1, image2, line_coords, step=5):
    # Sample key points along the line
    sampled_idx = np.arange(0, len(line_coords), step)

    # Match only sampled pixels
    for idx in sampled_idx:
        x, y = line_coords[idx].astype(int)
        # ... SSIM matching logic ...
```

```python
            mapping_sampled[idx] = best_pos

    # Interpolate intermediate points
    f_row = interp1d(idxs, rows, kind="linear")
    f_col = interp1d(idxs, cols, kind="linear")
```

Listing 17: Optimized line matching with interpolation

- **Step-Wise Matching**: Reduces computation by matching key points only
- **Linear Interpolation**: Maintains line continuity between matched points
- **Performance Optimization**: Significant speed improvement for long lines

match_junction_and_perps: Structured Feature Mapping Organizes the mapping of junction lines and perpendicular profiles:

```python
def match_junction_and_perps(self, image1, image2, junction_line,
    perpendiculars):
    mapping_junction = self.match_line_only(image1, image2,
    junction_line)

    mapping_perps = []
    for perp in perpendiculars:
        m = self.match_line_only(image1, image2, perp)
        mapping_perps.append(m)
```

Listing 18: Structured feature mapping

3.6.4 Final Data Extraction and Validation

set_line_from_mapping: Comprehensive Profile Extraction The final workflow step generates physically meaningful data from aligned images:

```python
def set_line_from_mapping(self, junction_line, mapping_junction, perps=
    None, mapping_perps=None):
    # Map junction line with fallback mechanisms
    for (x, y) in junction_line.astype(int):
        key = (int(y), int(x))
        if key in mapping_junction:
            r2, c2 = mapping_junction[key]
            mapped_junc_list.append([c2, r2])
        else:
            # Neighborhood search fallback
            for dy in (-1, 0, 1):
                for dx in (-1, 0, 1):
                    k = (int(y)+dy, int(x)+dx)
                    if k in mapping_junction:
                        r2, c2 = mapping_junction[k]
                        mapped_junc_list.append([c2, r2])

    # Interpolate missing points
    valid = ~np.isnan(mapped_junction[:, 0])
    if valid.sum() >= 2:
        mapped_junction[:, 0] = np.interp(idxs, idxs[valid],
    mapped_junction[valid, 0])
```

Listing 19: Profile extraction and validation

```python
# Process each perpendicular profile
for i, (orig_perp, m) in enumerate(zip(perps, mapping_perps)):
    # Map and interpolate points
    # ...

    # Calculate physical distances
    dist_um = np.linspace(-((length_px-1)*self.pixel_size*1e6)/2,
                          ((length_px-1)*self.pixel_size*1e6)/2,
                          length_px)

    # Sample SEM and current data
    sem_vals = map_coordinates(sem_map, [ys, xs], order=1, mode="
nearest")
    cur_vals = map_coordinates(current_map, [ys, xs], order=1, mode="
nearest")
```

Listing 20: Perpendicular profile processing

Key Features of Data Extraction

- **Robust Mapping**: Fallback mechanisms handle imperfect matches through neighborhood searching
- **NaN Handling**: Intelligent interpolation maintains data continuity
- **Physical Unit Conversion**: Converts pixel distances to micrometers for quantitative analysis
- **Data Sampling**: Uses `map_coordinates` for precise SEM and EBIC data extraction
- **Intersection Detection**: Automatically finds junction intersection points

```python
if show:
    fig, ax = plt.subplots()
    ax.imshow(self.pixel_maps[0], cmap="gray", origin="upper")
    if self.detected_junction_line is not None:
        ax.plot(self.detected_junction_line[:, 0],
                self.detected_junction_line[:, 1],
                color="green", linewidth=2, label="Mapped junction")
    # Plot perpendiculars and intersection points
```

Listing 21: Visual validation

This multi-image alignment capability enables precise comparative analysis of semiconductor properties under different bias conditions, providing crucial insights into voltage-dependent behavior of diffusion lengths and depletion regions.

4 Analysis Workflow

4.1 Step 1: Data Loading and Conversion

1. User selects TIFF files through GUI
2. Software extracts XML metadata and parses instrument parameters
3. Converts pixel values to current maps using calibration data
4. Saves processed data as CSV files for each frame

4.2 Step 2: Junction Detection

1. User draws approximate line along suspected junction
2. Software extracts rectangular ROI perpendicular to the line
3. Applies Canny edge detection with spline smoothing
4. Returns refined junction position with accuracy metrics

4.3 Step 3: Profile Analysis

1. Generates perpendicular profiles across the junction
2. Extracts SEM contrast and current values along each profile
3. Marks intersection points with detected junction
4. Applies low-pass filtering to reduce high-frequency noise

4.4 Step 4: Exponential Fitting

1. Fits falling exponentials to both sides of each profile
2. Uses shifted fitting windows to achieve stability
3. Calculates inverse diffusion lengths (1/) for each fit
4. Computes average depletion widths and diffusion lengths
5. Generates quality metrics (R^2 values) for each fit

5 Software Workflow Overview

This section describes the comprehensive workflow of the SEM-EBIC analysis tool. Currently, two distinct versions are available: a GUI-focused implementation emphasizing user interaction and visualization, and a characterization-oriented version prioritizing analytical capabilities. The long-term objective involves merging these implementations into a unified codebase while extending the overall functionality.

5.1 GUI-Based Software Workflow

5.1.1 TIFF File Processing

The analysis workflow commences with the processing of TIFF image files. The initial interface presents a file browser enabling users to select directories containing SEM data in TIFF format.

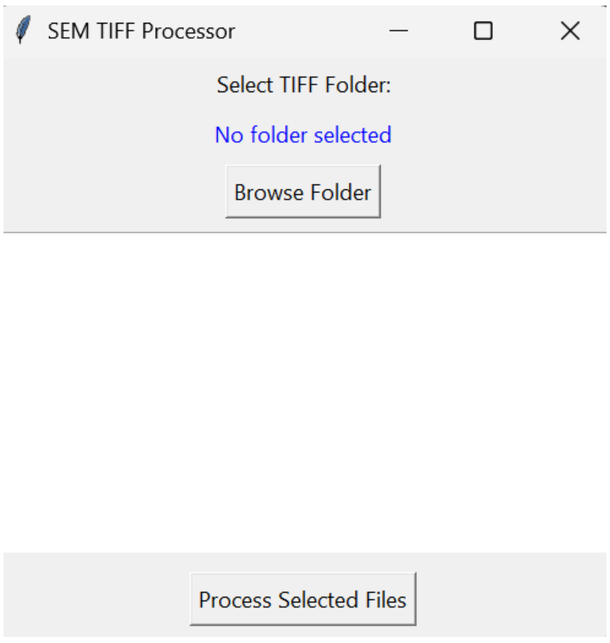
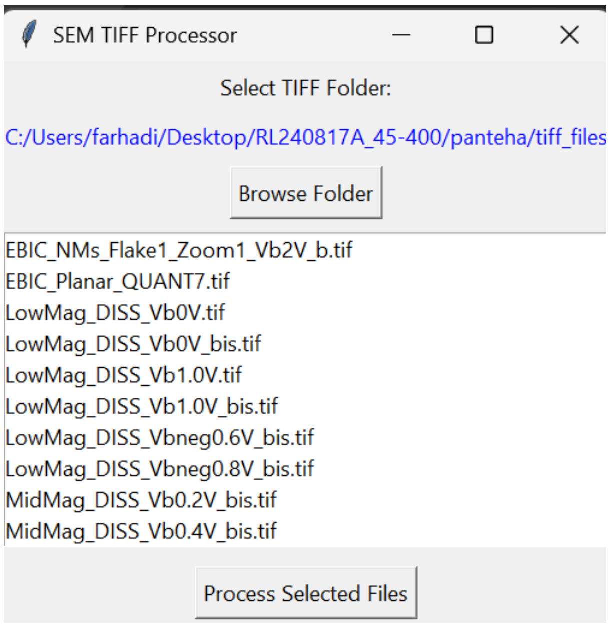


Figure 1: Directory selection interface for TIFF file processing

Upon directory selection, the system displays all available TIFF files within the chosen folder. Users may select individual files or multiple files for batch processing. Subsequently, the software prompts for an output directory where processed results will be stored.

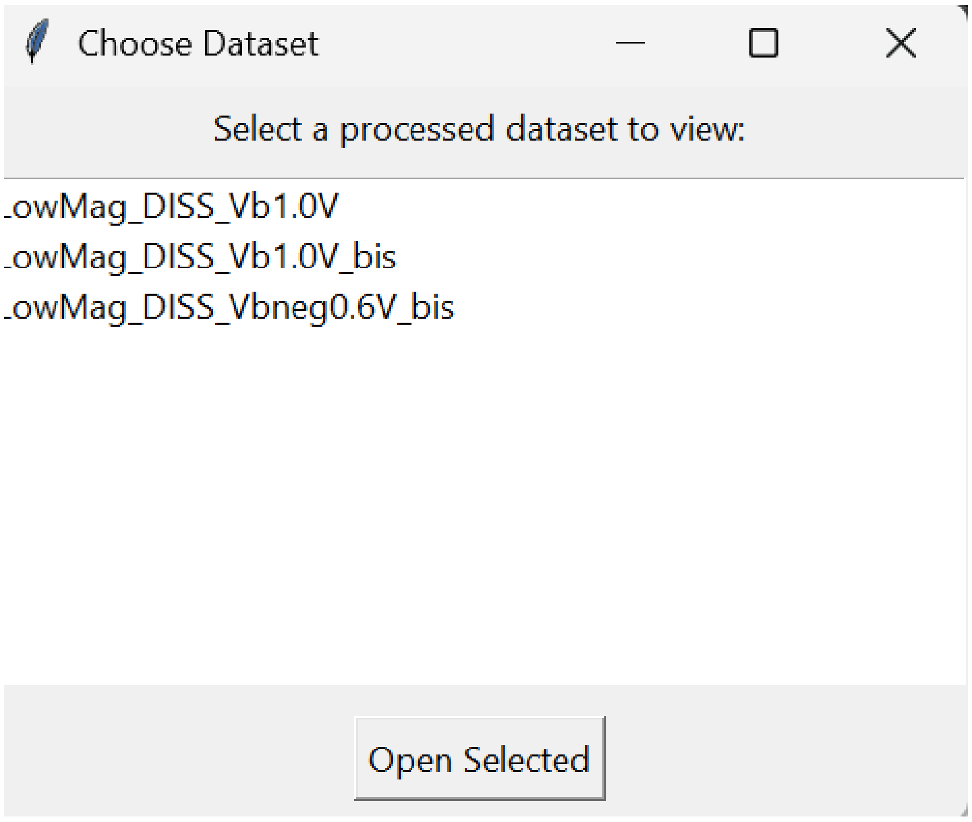


C:/Users/farhadi/Desktop/RL240817A_45-400/panteha/tiff_files

EBIC_NMs_Flake1_Zoom1_Vb2V_b.tif
EBIC_Planar_QUANT7.tif
LowMag_DISS_Vb0V.tif
LowMag_DISS_Vb0V_bis.tif
LowMag_DISS_Vb1.0V.tif
LowMag_DISS_Vb1.0V_bis.tif
LowMag_DISS_Vbneg0.6V_bis.tif
LowMag_DISS_Vbneg0.8V_bis.tif
MidMag_DISS_Vb0.2V_bis.tif
MidMag_DISS_Vb0.4V_bis.tif

Figure 2: TIFF file selection interface showing available datasets

Following processing completion, the interface presents a list of available images for visualization and analysis. Users can select specific datasets for detailed examination.



.owMag_DISS_Vb1.0V
.owMag_DISS_Vb1.0V_bis
.owMag_DISS_Vbneg0.6V_bis

Figure 3: Processed file selection interface for data visualization

5.1.2 Main Analysis Interface

The primary analysis window provides comprehensive access to all analytical tools and functionalities. This centralized interface facilitates seamless navigation through the complete EBIC analysis pipeline.

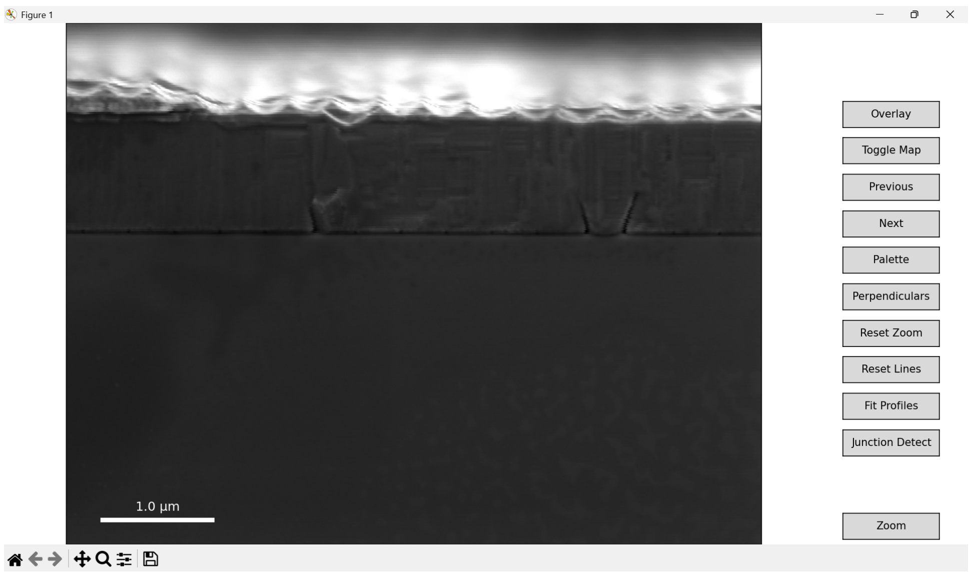


Figure 4: Main analysis interface with complete tool accessibility

The interface includes navigation controls for multi-frame datasets, allowing sequential examination of different image frames and toggling between various data representations.

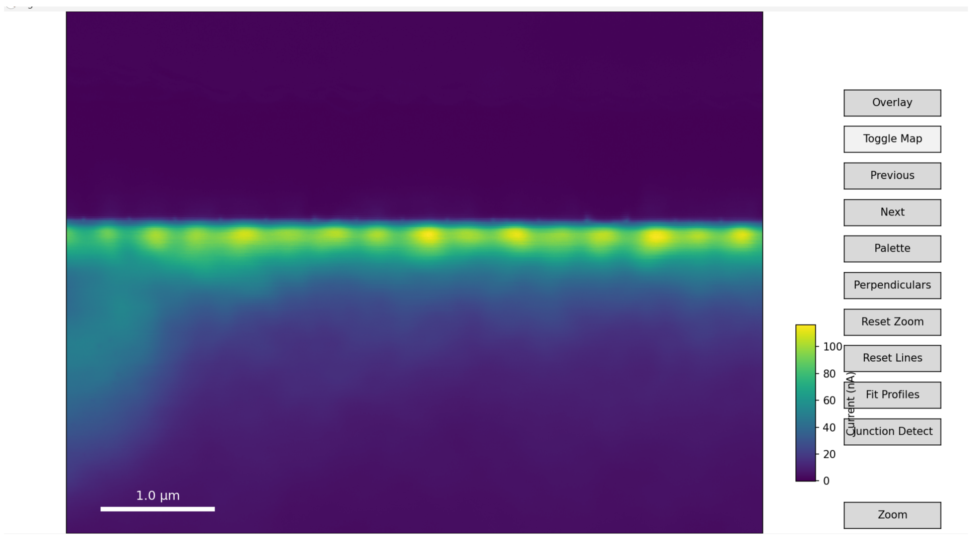


Figure 5: Frame navigation and display mode toggle controls

5.1.3 Image Visualization Tools

The software incorporates advanced visualization capabilities, including overlay functionality for comparative analysis of different data channels or processing results.

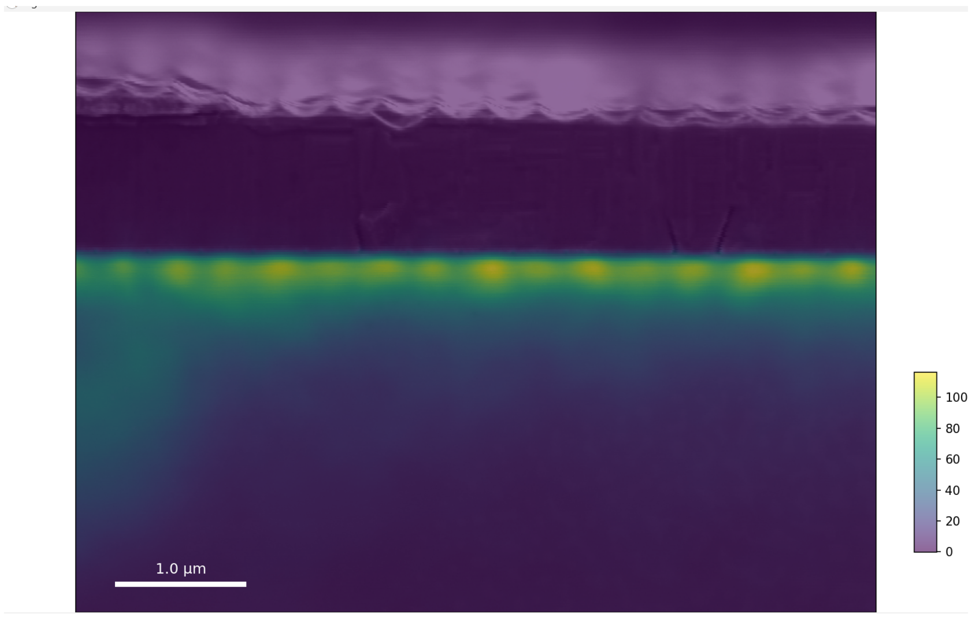


Figure 6: Data overlay functionality for multi-channel visualization

The zoom functionality employs a rectangular selection mechanism. Users activate the zoom tool, delineate the region of interest, and subsequently deactivate the tool to return to full-image viewing.

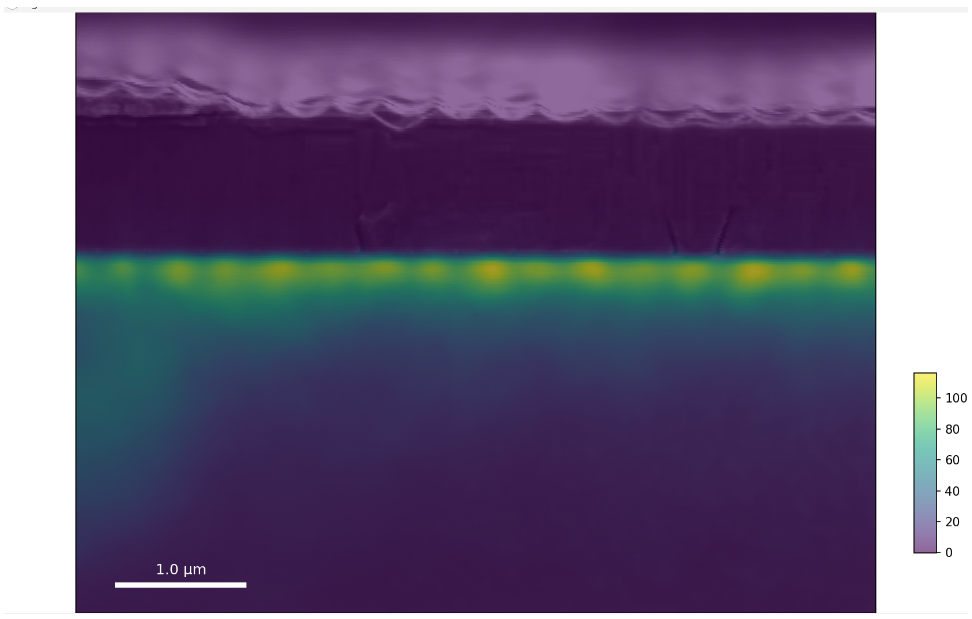


Figure 7: Interactive zoom feature with rectangular region selection

5.1.4 Junction Analysis Workflow

The junction detection process initiates with manual line drawing approximating the p-n junction location. The system subsequently prompts for window width specification to define the analysis region, optimizing junction detection while mitigating interference from extraneous features.

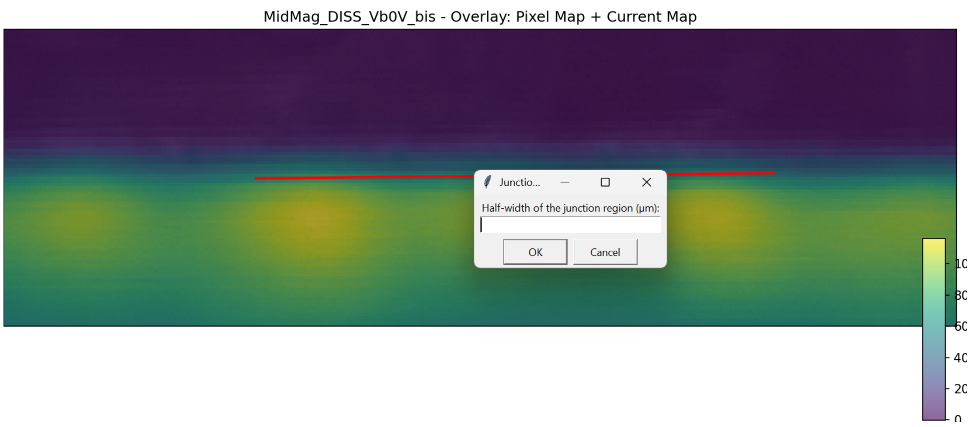

MidMag_DISS_Vb0V_bis - Overlay: Pixel Map + Current Map
Junctio...
Half-width of the junction region (μm):
OK
Cancel
10
80
60
40
20
0

Figure 8: Manual junction approximation with adjustable analysis window

The perpendicular profile analysis module generates cross-sectional profiles across the detected junction. Users specify the quantity and length of perpendicular lines, with the resulting profiles displaying both SEM intensity and EBIC current data. Intersection points between profiles and the junction are explicitly marked at the zero-position reference.

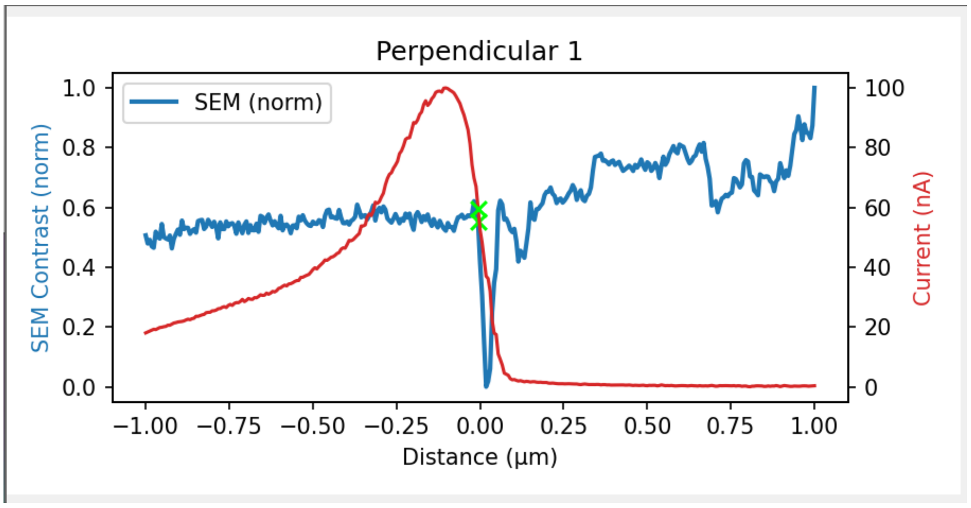


Figure 9: Perpendicular profile analysis showing SEM and EBIC data intersections

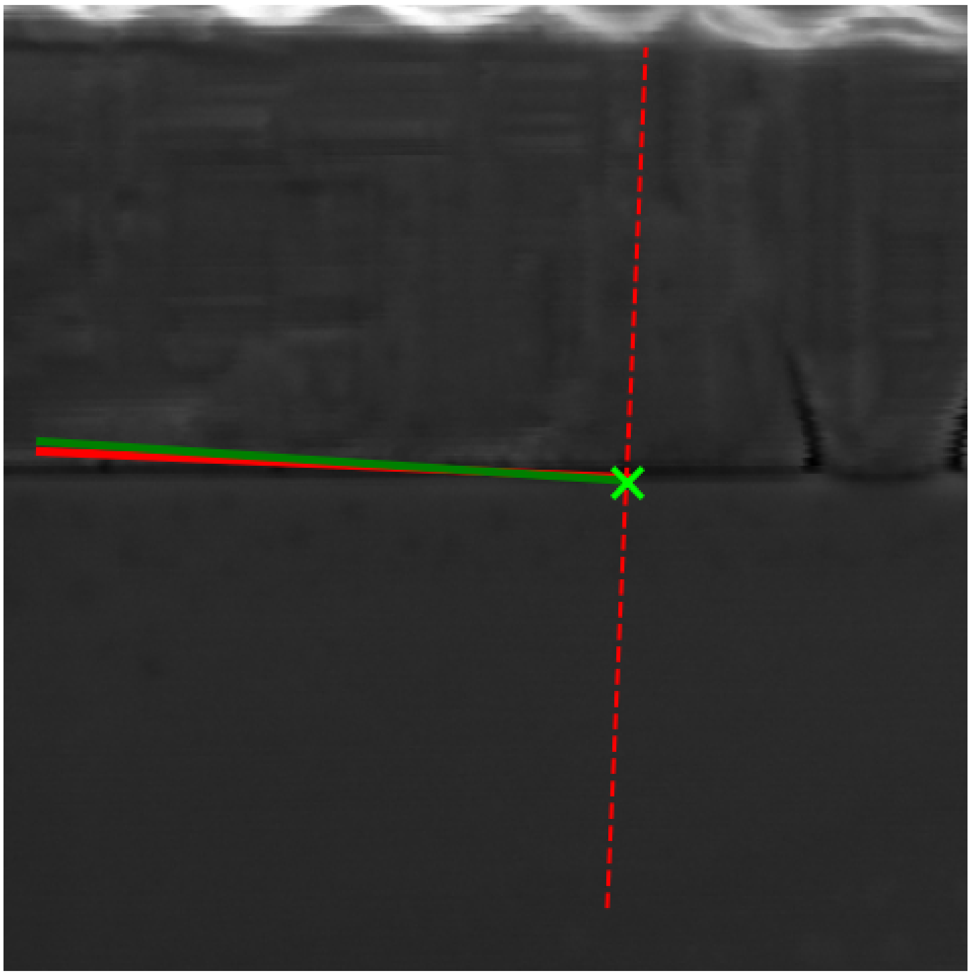


Figure 10: Detected junction and manual junction with perpendicular line

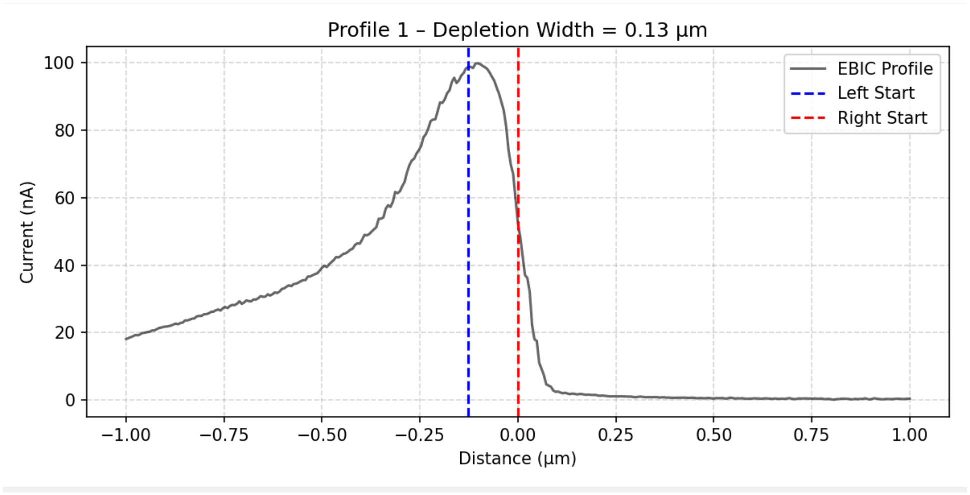


Figure 11: Depletion region characterization and width quantification

A visualization of the data used for fitting is overlaid on the original profile, providing an intuitive understanding of the fitting process. The visualization shows different truncations on tail which is shortening the tail. The left side is inverted to suit falling exponential model.

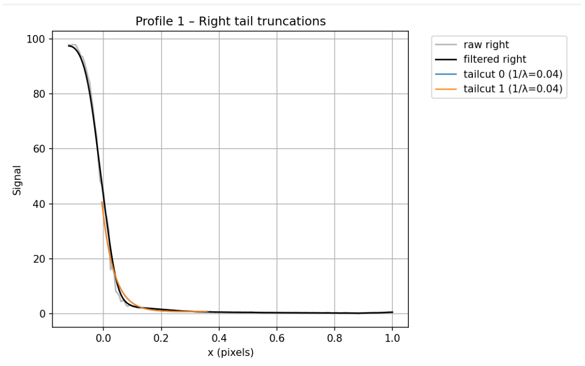


Figure 12: Right tail truncation

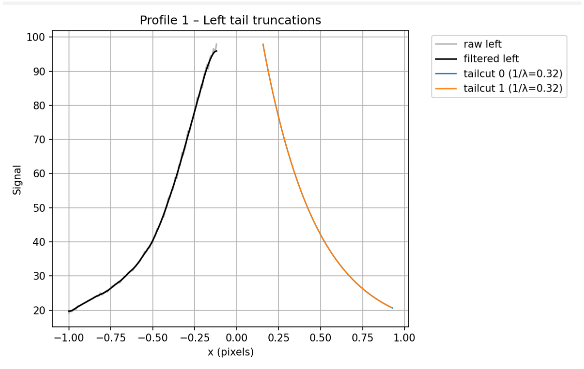


Figure 13: Left tail truncation

5.2 Parameter Extraction and Results

5.2.1 Iterative Stabilization Algorithm for Diffusion Length Extraction

The core innovation in the diffusion length extraction methodology is the iterative stabilization algorithm that ensures robust and physically meaningful parameter estimation. The process follows a sophisticated multi-stage approach:

5.2.2 Algorithm Workflow

The stabilization process operates through the following sequential steps:

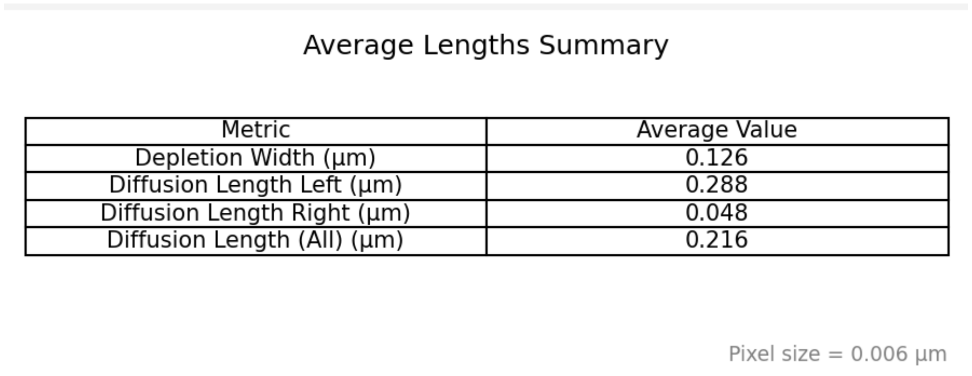
1. **Initial Shift Phase**: Systematic shifting of the fitting window starting point both left and right away from the junction position
2. **Exponential Fitting**: Application of nonlinear least-squares fitting to the exponential decay model $I(x) = A \cdot e^{-\lambda(x-x_0)} + y_0$ for each shifted window position
3. **Convergence Monitoring**: Continuous evaluation of the extracted decay constant λ (and thus diffusion length $L = 1/\lambda$) until stability is achieved
4. **Tail Truncation Phase**: Progressive removal of data points from the tail region where signal-to-noise ratio deteriorates
5. **Stability Verification**: Repeated fitting with truncated data to confirm parameter stability
6. **Iterative Refinement**: Cyclic repetition of the entire process until both sides of the junction demonstrate convergence

5.2.3 Convergence Criteria Enhancement

As noted, the convergence condition can be improved by requiring bilateral agreement between the left and right side fittings. The enhanced convergence criterion becomes:

$$\text{Converged} = \left(|L_{\text{left}}^{(k)} - L_{\text{left}}^{(k-1)}| < \epsilon\right) \wedge \left(|L_{\text{right}}^{(k)} - L_{\text{right}}^{(k-1)}| < \epsilon\right) \wedge \left(|L_{\text{left}}^{(k)} - L_{\text{right}}^{(k)}| < \delta\right) \tag{7}$$

Where δ represents an additional tolerance for inter-side consistency, ensuring that the extracted parameters are physically plausible and consistent across both sides of the junction.



Average Lengths Summary

Metric	Average Value
Depletion Width (µm)	0.126
Diffusion Length Left (µm)	0.288
Diffusion Length Right (µm)	0.048
Diffusion Length (All) (µm)	0.216

Pixel size = 0.006 µm

Figure 14: Comprehensive fitting results with quantitative parameters

The depletion region analysis provides specific characterization of the junction properties, displaying the spatially resolved depletion width measurements.

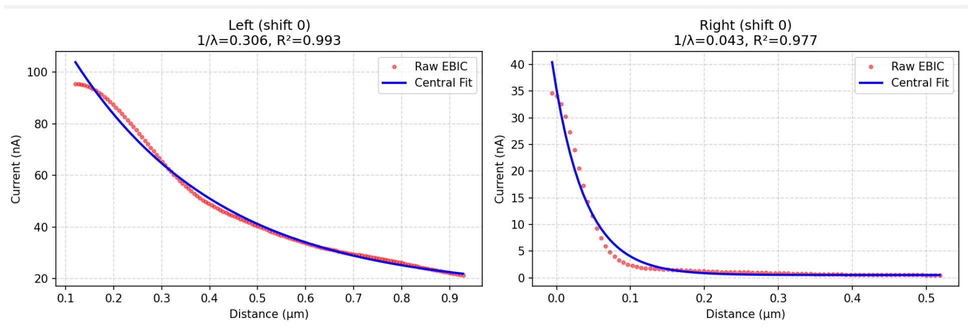


Figure 15: Fitted profiles at 0 original starting point (0 shift)

5.2.4 Enhancement of Physical Model: Bi-exponential Diffusion

The current model is a simple exponential is a simple exponential as discussed before, while it can be improved in the way that is discussed bellow.
Minority carrier concentration decay with distance is modeled using a bi-exponential function:

$$C(x) = A_1 e^{-x/L_1} + A_2 e^{-x/L_2} \tag{8}$$

where L_1, L_2 are diffusion lengths of two transport components, and A_1, A_2 are their relative amplitudes.

The two terms reflect distinct mechanisms:

- L_1, A_1: bulk diffusion with longer characteristic length
- L_2, A_2: surface or defect-related recombination with shorter length

A signal-weighted average provides a practical single parameter:

$$L_{\text{eff}} = A_1 L_1 + A_2 L_2 \tag{9}$$

5.2.5 Significance

Compared to single-exponential models, the bi-exponential approach offers:

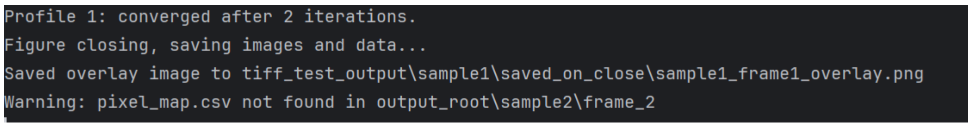
- better representation of multiple recombination pathways
- improved fit to complex decay profiles
- insight into both bulk and surface effects

Thus, while L_{eff} is a convenient summary, the full model captures the richer physics of carrier transport.

5.3 Pixel matching

You can go ahead with drawing junction and defining perpendicular profiles, the red line is the manual line and the green line is the detected junction.
keep in mind that this version is using Spline post processing which means that the detected junction is not a line. 3.3.4

After compeleting the same workflow as done before, you can close every plot and also the first image, then processing of the second image starts. You should then see the last line in this image, meaning that the second image is started to be processed.



```
Profile 1: converged after 2 iterations.
Figure closing, saving images and data...
Saved overlay image to tiff_test_output\sample1\saved_on_close\sample1_frame1_overlay.png
Warning: pixel_map.csv not found in output_root\sample2\frame_2
```

Figure 16: Second image processing

Then the matched junction and perpedicular line show up, as it's obvious the length and orientation has changed a little. Change of orientation is ok, but the length should not change. The change in the length is caused by line fitting that may include pixels that has moved alot.

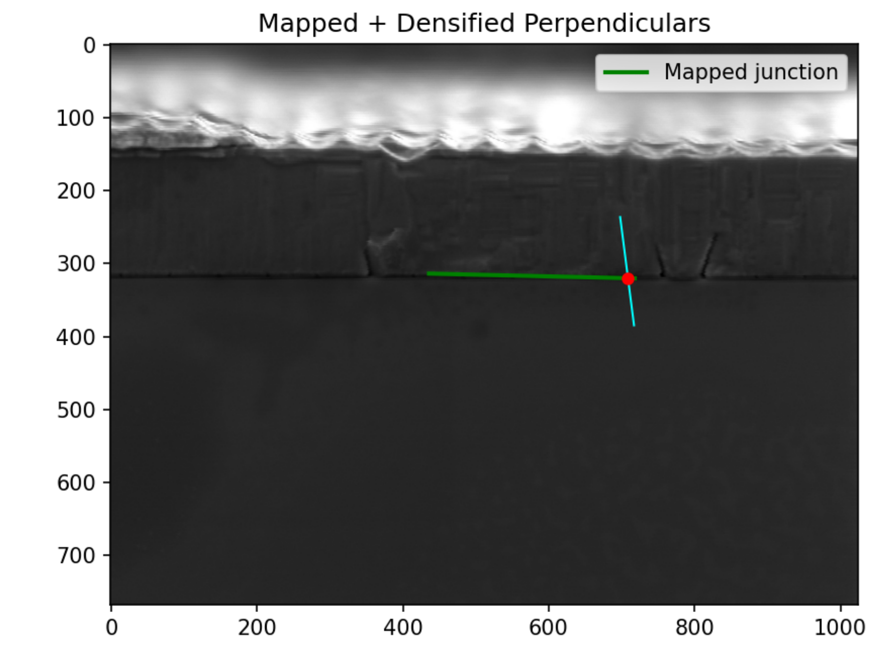


Figure 17: matched lines

Then you can fit profiles as you did before, the results, somehow, match our predictions.

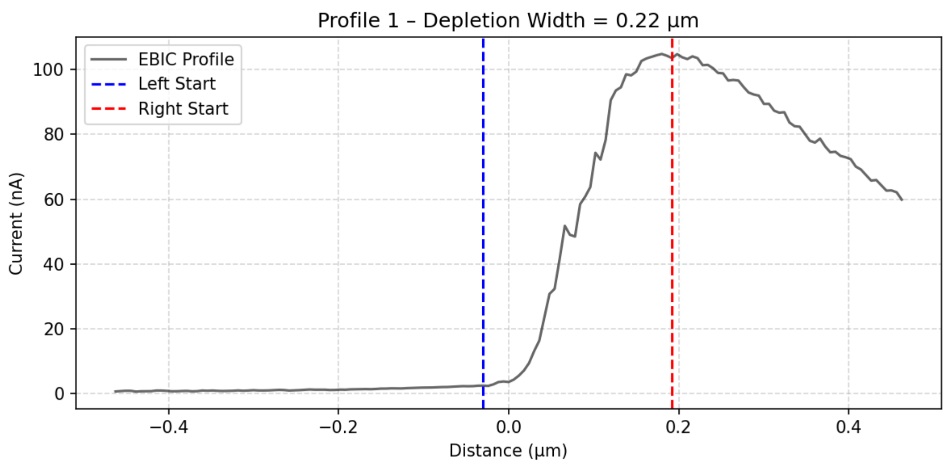


Figure 18: Depletion region defined after pixel matching

5.3.1 Profile Matching Precision

Since the detected profile is not perfectly perpendicular to the junction, the fitting is not sufficiently precise. This limitation can be mitigated by improving the algorithm used for matching the profiles across images. Also simple line fitting is not the final sufficient destiny, the procedure can be improved. The method is noted here 3.6.3

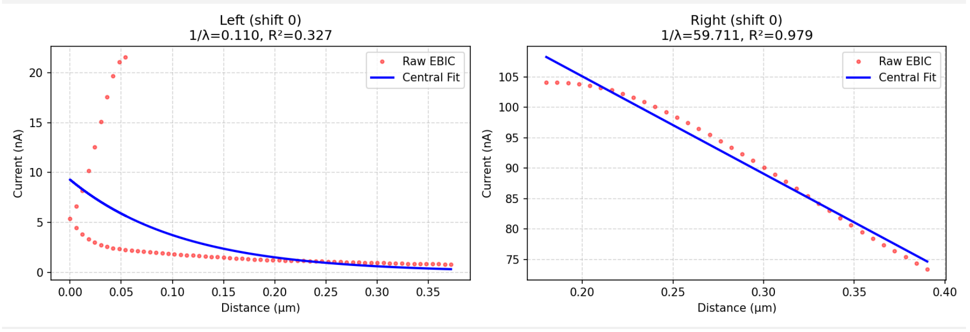


Figure 19: Bad profiles for fitting

6 Future Development Plans

6.1 Enhanced Edge Detection Based on Current Map

Development of advanced edge detection algorithms specifically optimized for EBIC current maps, improving junction detection accuracy and robustness.

6.2 Software Version Integration

Integration of the currently separate GUI-based and characterization-focused versions into a unified codebase with comprehensive functionality.

6.3 Convergence Criteria Enhancement

As discussed in Section 5.2.3

6.3.1 Enhancement of Physical Model: Bi-exponential Diffusion

As discussed in Section 5.2.4

6.4 Advanced Pixel Matching Algorithms

Implementation of more sophisticated pixel matching techniques for improved multi-image alignment and feature tracking across different measurement conditions.

6.5 Pixel Matching and Profile Matching Precision

As discussed in Section 5.3.1, the profile alignment can be improved ...