imports

In [282…

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import os
from scipy.interpolate import interp1d
import numpy as np
from scipy import signal
import math
from scipy.interpolate import UnivariateSpline
import matplotlib.ticker as ticker
```

read_xyz

In [3]:

```
def read_xyz(file_path):
    with open(file_path, 'r') as file:
        data = [list(map(float, line.split())) for line in file]
    return data
```

resampling

In [4]:

```
def resample_to_match_length(x, y, target_length):
    f = interp1d(x, y)
    new_x = np.linspace(min(x), max(x), target_length)
    new_y = f(new_x)
    return new_x, new_y
```

angle deviation

In [248…

```
def interpolate(x1,y1,x2,y2,smoothing):
    # Determine which dataset has higher resolution and interpolate the other
    spline = UnivariateSpline(x2, y2, s=smoothing)  # s is the smoothing factor, adjust as needed
    y2_interp = spline(x1)
    x_common = x1
    y1_common = y1
    y2_common = y2_interp
    return spline, y1_common, y2_common,x_common
```

```
def slide_window(angle_deviations, x_common, window_size):
    # Apply a moving average to the angle deviations
    window = np.ones(window_size) / window_size
    angle_deviations_smoothed = np.convolve(angle_deviations, window, mode='valid')

    # Adjust x coordinates for moving average
    x_midpoints = (np.array(x_common[:-1]) + np.array(x_common[1:])) / 2
    x_midpoints = x_midpoints[window_size - 1:]
    return x_midpoints, angle_deviations_smoothed
```

In [247…

```
def compute_angle_deviations(x1, y1, x2, y2, window_size=0,smoothing=0.6):
    if len(x1) < len(x2):
        spline, y1_common, y2_common, x_common = interpolate(x1, y1, x2, y2,smoothing)
        y_reference = y1
        x_reference = x1
    else:
        spline, y1_common, y2_common, x_common = interpolate(x2, y2, x1, y1,smoothing)
        y_reference = y2
        x_reference = x2


    # Interpolate to get y values at common x points
    y_interpolated = spline(x_common)


    # Apply sliding window smoothing to the interpolated y values if sliding window > 0
    if window_size > 0:
        x_common, y_smoothed = slide_window(y_interpolated, x_common, window_size)
    else:
        x_common, y_smoothed = x_common, y_interpolated

    # Trim the length of x_common to match that of y_smoothed
    min_len = min(len(x_common), len(y_smoothed))
    x_common = x_common[:min_len]
    y_smoothed = y_smoothed[:min_len]

    # Compute the slopes of the reference and smoothed curves
    slope_reference = np.gradient(y_reference[:min_len], x_reference[:min_len])
    slope_smoothed = np.gradient(y_smoothed, x_common)

    # Compute the angle between the slopes at each point
    angle_deviations = np.arctan((slope_reference - slope_smoothed) / (1 + slope_reference * slope_smoothed))

    # Convert the angle deviations from radians to degrees
    angle_deviations = np.degrees(angle_deviations)
    return x_reference, y_reference, x_common, y_smoothed, angle_deviations
```

find closest element

In [6]:

```
#takes a list, and gives the closes index of the specified number
def find_closest_element(sorted_list, target):
    left = 0
    right = len(sorted_list) - 1

    while left <= right:
        mid = (left + right) // 2

        # If an exact match is found, return the index of that element.
        if sorted_list[mid] == target:
            return mid

        # Adjust the search range.
        if sorted_list[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    # If the loop exits without finding an exact match,
    # compare the elements at indices left and right to find the closest.

    # If target is smaller than the first element.
    if right < 0:
        return 0

    # If target is larger than the last element.
    if left >= len(sorted_list):
        return len(sorted_list) - 1

    # Return the index of the closest element based on the difference from the target.
    if target - sorted_list[right] <= sorted_list[left] - target:
        return right
    else:
        return left
```

In [7]:

```
def cut_list(x1, y1, x2, y2):
    x1_max = max(x1)
    x2_max = max(x2)
    # Adjust the lists for maximum values.
    if x1_max < x2_max:
        cut_index = find_closest_element(x2, x1_max)
        x2 = x2[:cut_index + 1]
        y2 = y2[:cut_index + 1]
    else:
        cut_index = find_closest_element(x1, x2_max)
        x1 = x1[:cut_index + 1]
        y1 = y1[:cut_index + 1]

    # Adjust the lists for minimum values.
    x1_min = min(x1)
    x2_min = min(x2)
    if x1_min < x2_min:
        cut_index = find_closest_element(x1, x2_min)
        x1 = x1[cut_index:]
        y1 = y1[cut_index:]
    else:
        cut_index = find_closest_element(x2, x1_min)
        x2 = x2[cut_index:]
        y2 = y2[cut_index:]

    return x1, y1, x2, y2
```

low pass filter

In [8]:

```
def low_pass(y):
    # Apply low-pass filter using Kaiser window method
    cutoff_freq = 0.00001  # Adjust the cutoff frequency as needed
    beta = 0.0001  # Adjust the beta value for sharper roll-off
    filter_length = 50  # Adjust the filter length as needed
    taps = signal.firwin(filter_length, cutoff_freq, window=('kaiser', beta))
    filtered_y = signal.convolve(y, taps, mode='same')
    filtered_y_list = filtered_y.tolist()
    return filtered_y_list
```

plot

In [367…

```
def plot_angle_deviation(scanned_name,virtual_name,save_name,x,y,sliding_window,smoothing):
    sliding_window_size = sliding_window
    # Read data from first .xyz file
    file_name1 = "\\"+scanned_name
    file_path1 = os.getcwd()+ file_name1
    data1 = read_xyz(file_path1)
    data1 =  sorted(data1, key=lambda coordinate: coordinate[x])

    # Read data from second .xyz file
    file_name2 = "\\"+virtual_name
    file_path2 = os.getcwd()+ file_name2
    data2 = read_xyz(file_path2)

    data1 = [[x[0], x[1], x[2]] for x in data1]
    data2 = [[x[0], x[1], x[2]] for x in data2]
    data2 =  sorted(data2, key=lambda coordinate: coordinate[x])

    # Extract coordinates from first data set
    x_coords1 = [item[x] for item in data1]
    y_coords1 = [item[y] for item in data1]


    # Extract coordinates from second data set
    x_coords2 = [item[x] for item in data2]
    y_coords2 = [item[y] for item in data2]

    #cuts the lists to equal length
    x1,y1,x2,y2 = cut_list(x_coords1,y_coords1,x_coords2,y_coords2)

    #butterworth high pass
    #y1 = low_pass(y1)


    #calculate angle deviations
    #x_deviations, angle_deviations, x_interpolated, y_interpolated = compute_angle_deviations(x1,y1,x2,y2,sliding_window_size)
    x_reference, y_reference, x_common, y_interpolated, angle_deviations = compute_angle_deviations(x1,y1,x2,y2,sliding_window_size,smoothi

    # Create figure and the first axis
    fig, ax1 = plt.subplots(figsize=(12, 5))

    # Plot the data on the first axis
    line1, = ax1.plot(x1, y1, label='Scanned surface',alpha=0.6)
    line2, = ax1.plot(x_reference, y_reference, label='Virtual',alpha=0.6)
    line25, = ax1.plot(x_common,y_interpolated,label="Interpolated scan",color="red",linestyle = "dotted")




    #window line
    '''
    sliding_window_color = "brown"
```

```
window_line, = ax1.plot([x1[0],x1[sliding_window_size]],[0,0], label=f'Sliding window size: {(x1[sliding_window_size]-x1[0]):.2f}mm',co
ax1.axvline(x=x1[0], color=sliding_window_color, linestyle='--',linewidth=1)
ax1.axvline(x=x1[sliding_window_size], color=sliding_window_color, linestyle='--',linewidth=1)
'''

#labels and so
ax1.set_xlabel('X [mm]')
ax1.set_ylabel('Form [mm]')
ax1.tick_params('y')
ax1.set_aspect("equal")
ax1.patch.set_alpha(0.0)

# Create the second axis
ax2 = ax1.twinx()
ax2.set_zorder(ax1.get_zorder() - 1)

# Plot the data on the second axis
line3, = ax2.plot(x_common, angle_deviations,alpha=1, label='Angle deviations',color="tab:green",linestyle = "dotted")


ax2.set_ylabel('Angle deviation [ \u00b0 ]')
#ax2.tick_params('y')
ax2.set_ylim([-20,20])
ax2.locator_params(axis='y', nbins=15)

ax2.grid(True)

#lines = [line1,line2,line25,line26,line3]
lines = [line1,line2,line25,line3]
labels = [l.get_label() for l in lines]
plt.legend(lines,labels)

# Show the plot
plt.title('Virtual and scanned model, with angle deviation')

plt.grid(True, which='major')
plt.grid(True, which='minor', alpha=0.3)

#save figure
save_path = os.path.join(os.getcwd(), save_name + ".png")
plt.savefig(save_path)
print(f'Figure saved as {save_path}')
plt.show()
```

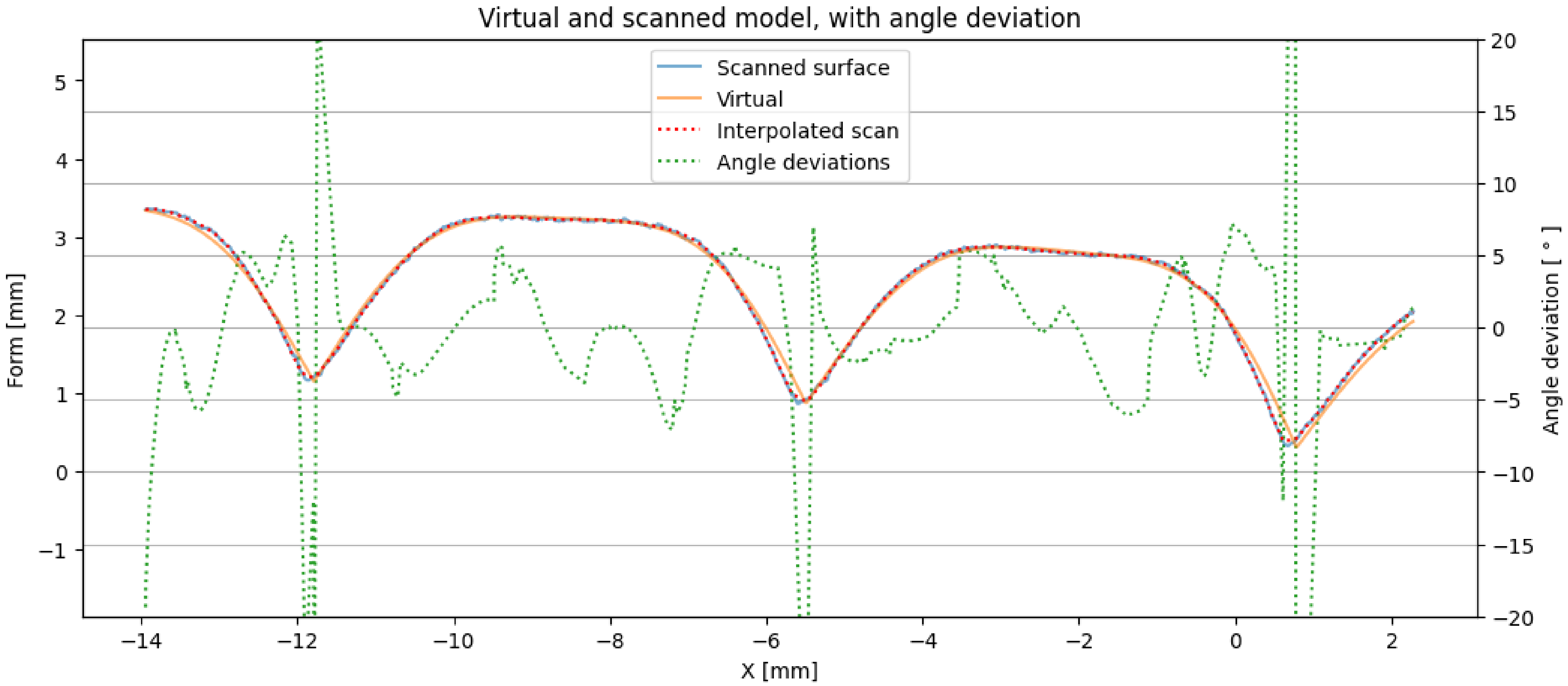
"main"

3d inner ytre

In [380…

```
plot_angle_deviation("sectioned_3d_indrelinse_ytre.xyz","sectioned_3d_indrelinse_ytre_ref.xyz","3d_innerLens_ytre",1,2,0,0.8)
```

```
Figure saved as C:\Users\bjorn\Documents\emnerNtnu\master\Jupyter\angleDeviation\3d_innerLens_ytre.png
```

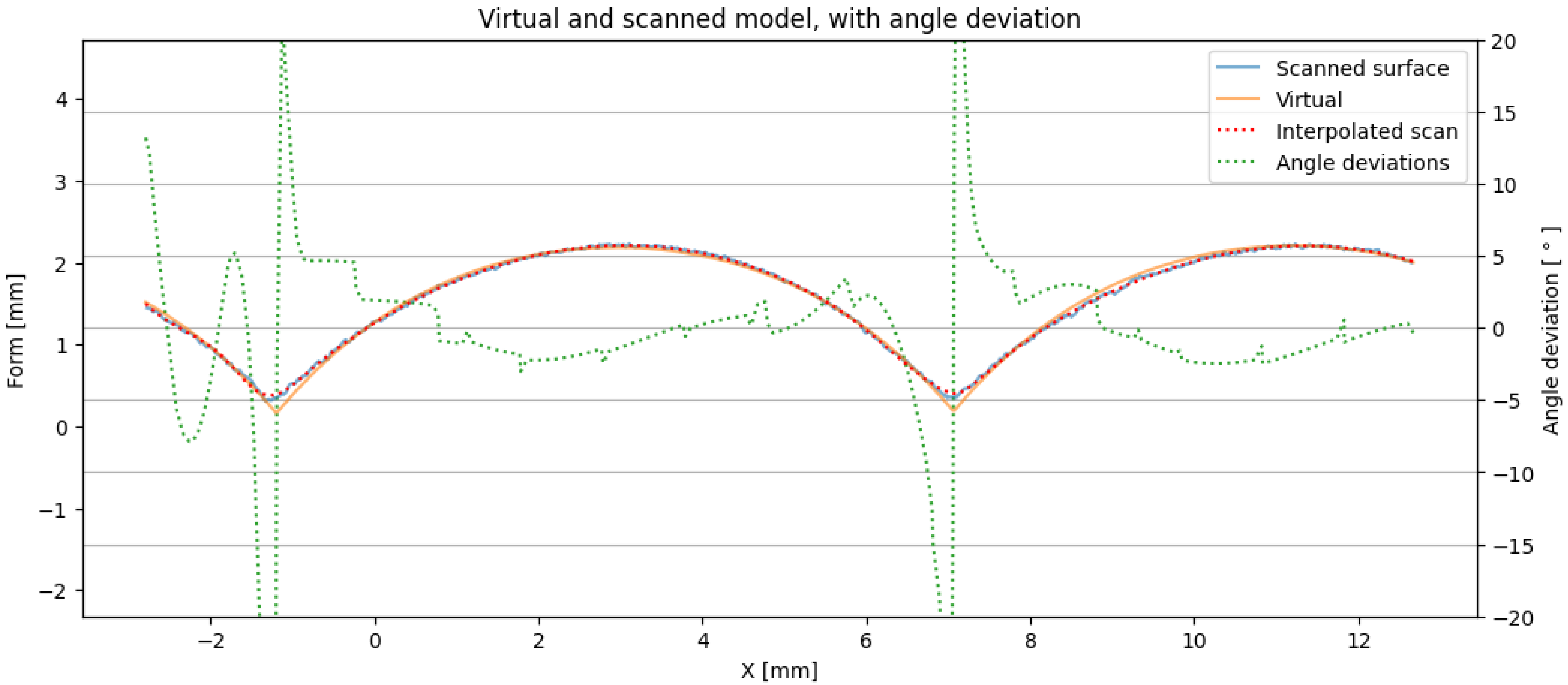


3d inner indre

In [369…

```
plot_angle_deviation("sectioned_3d_indrelinse_indre.xyz","sectioned_3d_indrelinse_indre_ref.xyz","3d_innerLens_indre",0,1,0,0.8)
```

```
Figure saved as C:\Users\bjorn\Documents\emnerNtnu\master\Jupyter\angleDeviation\3d_innerLens_indre.png
```

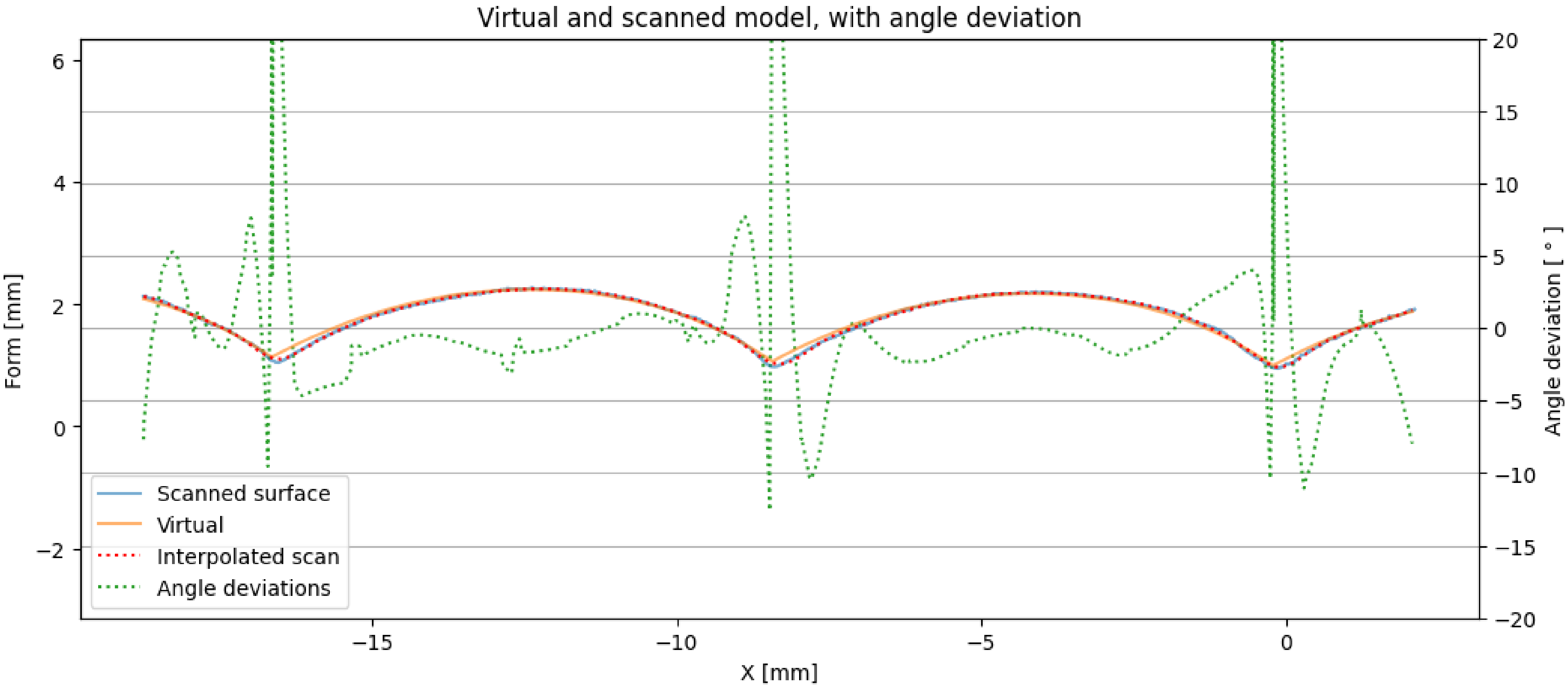


3d outer ytre

In [377…

```
plot_angle_deviation("sectioned_3d_outerLens_outer.xyz","sectioned_3d_outerLens_outer_ref.xyz","3d_outerLens_outer",1,2,0,0.8)
```

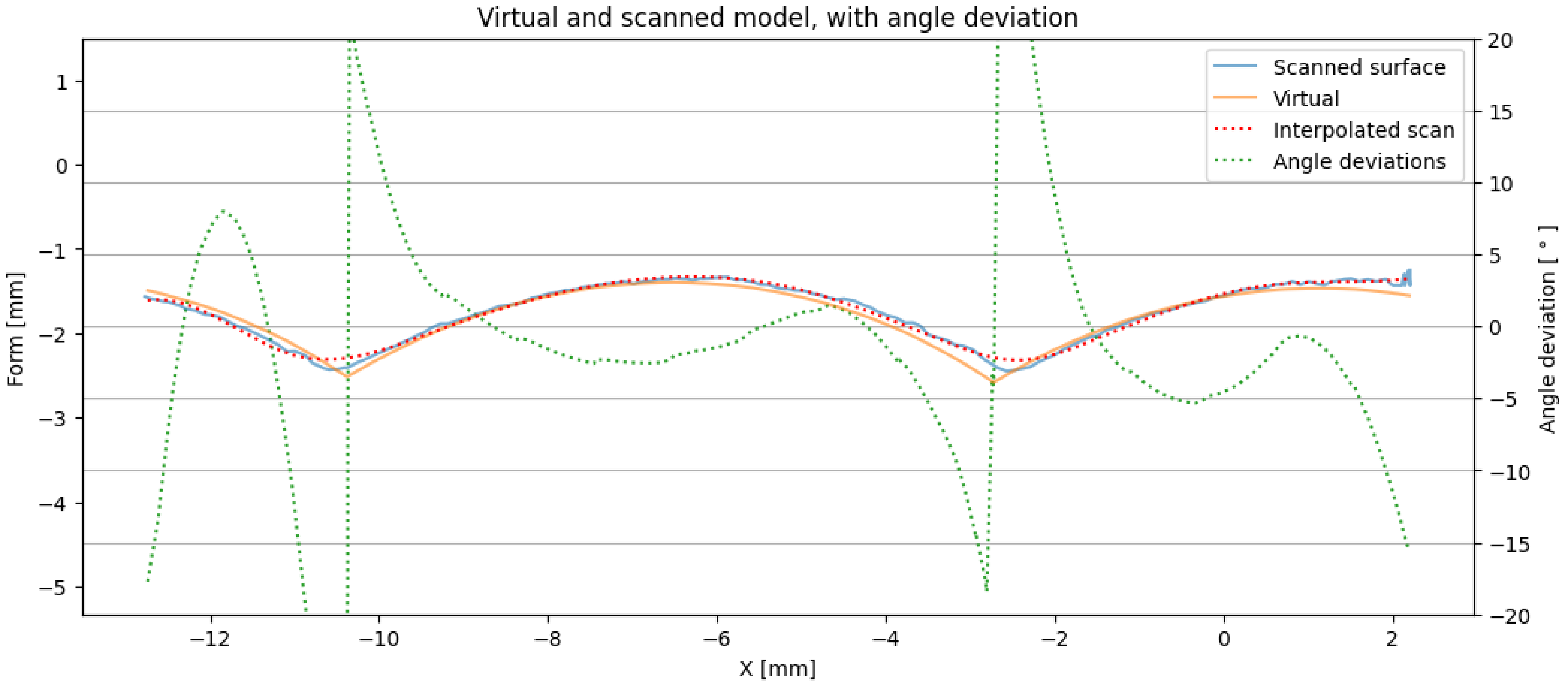
```
Figure saved as C:\Users\bjorn\Documents\emnerNtnu\master\Jupyter\angleDeviation\3d_outerLens_outer.png
```



i ytre

```
In [378…  plot_angle_deviation("i_outer_section.xyz","i_outer_section_ref.xyz","i_outer",1,2,0,0.8)
```

```
Figure saved as C:\Users\bjorn\Documents\emnerNtnu\master\Jupyter\angleDeviation\i_outer.png
```

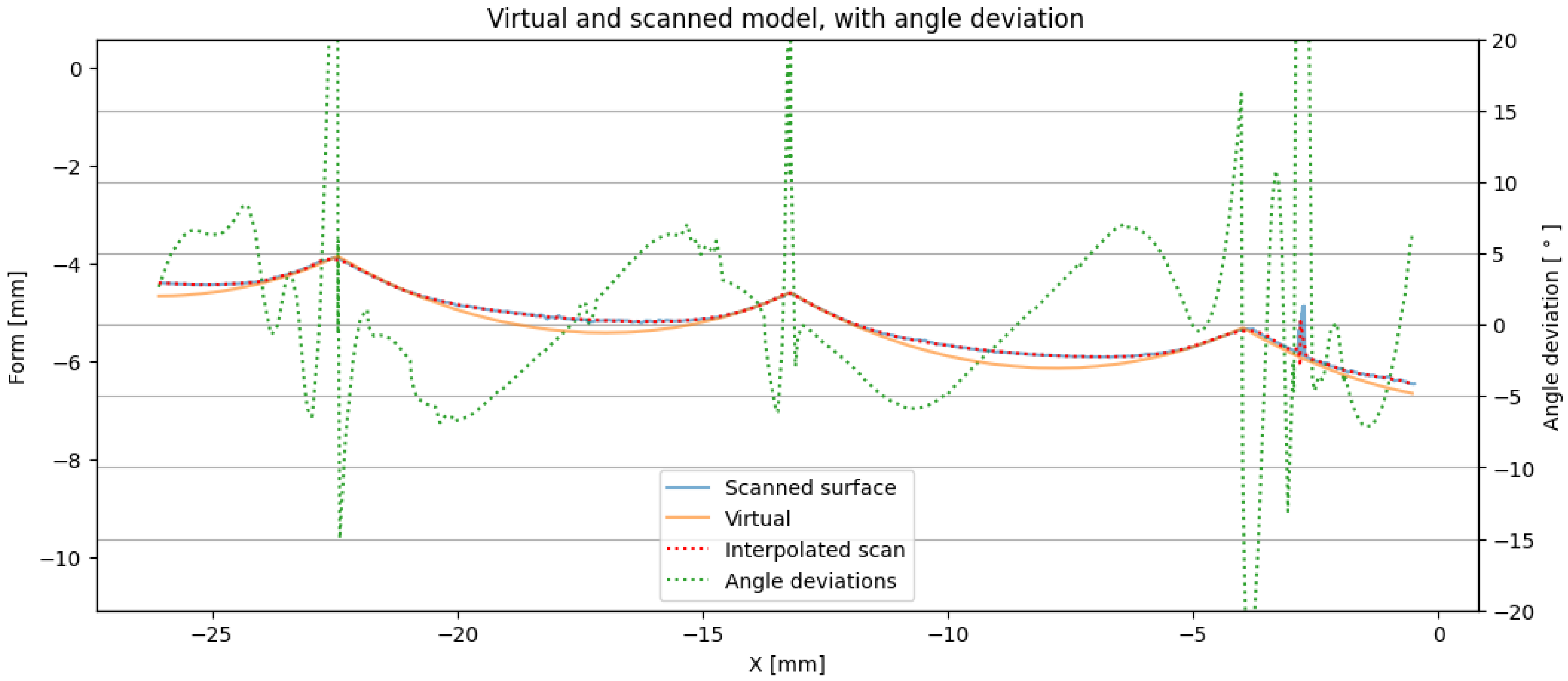


Vii Outer

In [379…

```
plot_angle_deviation("vii_25x35_section.xyz","vii_25x35_section_ref.xyz","vii_outer",1,2,0,0.8)
```

```
Figure saved as C:\Users\bjorn\Documents\emnerNtnu\master\Jupyter\angleDeviation\vii_outer.png
```



This notebook contains code and analysis assisted by OpenAI's GPT-4 model.

```
In [ ]:
```