Proof of Concept for Automated STEM Tilting Using Image Analysis for Quantifoil Grids

Lynnicia Massenburg, Elizabeth Heon, Sai Venkata Gayathri Ayyagari, Andrew Balog, Darel Pates, Sita Sirisha Madugula

Tilt allows a microscopist to align specific orientations of the sample to the electron beam for imaging and diffraction. Virtually all TEM experiments will begin with tilting the sample to a desired orientation; thus, sample tilting is a clear goal for automated electron microscopy efforts. This project focuses on a simplified case of automated tilting: aligning a Quantifoil grid with circular holes perpendicular to the beam. When the grid is tilted relative to the beam, projection of the 3D grid into the 2D image plane will make the holes appear as ellipses, whereas when the grid is perpendicular to the beam the holes should appear perfectly circular, (Figure 1). In this case, tilting can be done at low magnification in real space by tracking the appearance of the holes. Therefore, the goal of this effort is to develop a proof-of-concept auto-tilting algorithm by optimizing the circularity of holes in a Quantifoil grid.

Methods:

In this work, synthetic images will be used for proof of concept. The work is divided into two main parts:

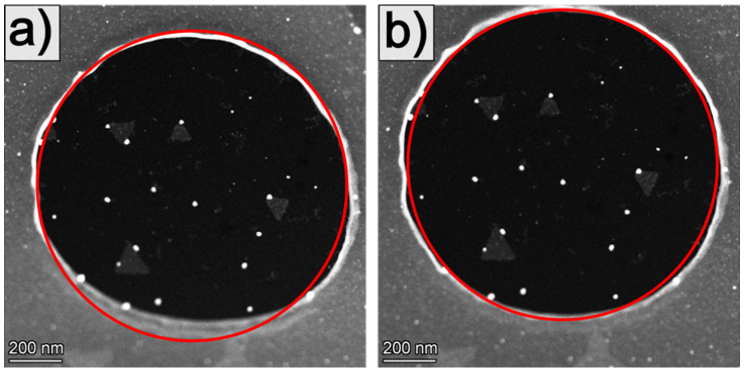

Figure 1: Experimental STEM HAADF image of a hole in a Quantifoil grid both a) tilted relative to the electron beam, and b) after tilting to the grid perpendicular to the electron beam. This data was acquired experimentally before the hackathon.

Digital Twin: Digital twins are virtual representations of physical objects; in microscopy a digital twins simulates such actions as tilting, focusing, and collecting images and diffraction patterns, returning either simulated images or pre-captured data [1]. The use of a digital twin allows for the development of algorithms like this one, which require iterative collection of data at given parameters, without the need to be connected to a real instrument.

For this work, a digital twin was created which produced appropriate simulated images for the auto-tilting algorithm. This was done by treating the grid (and by extension the hole) as a plane in 3D space. The normal to this plane was tracked and updated appropriately when tilt commands were issued by the simulated auto tilt algorithm. The matplotlib library [2] was then used to generate an image of the projected ellipse.

Circle Recognition and Optimization: A method was developed to recognize circular figures in synthesized pseudo-micrographs. The circle recognition and optimization implements a complete pipeline that takes a folder of images at different tilts, segments the main (assumed circular) object in each image, measures its geometric circularity, and performs robust circle fitting, while saving visual diagnostics and a summary table of metrics.

Overall Autotilting Workflow: A starting image is first generated with the grid at a random tilt. The circle optimization code is performed to determine the circularity metric.

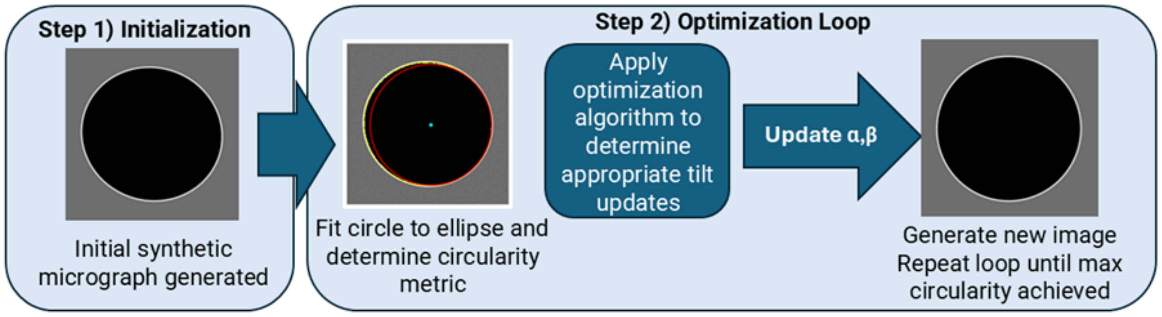
Step 1) Initialization
Initial synthetic micrograph generated
Step 2) Optimization Loop
Fit circle to ellipse and determine circularity metric
Apply optimization algorithm to determine appropriate tilt updates
Update α,β
Generate new image Repeat loop until max circularity achieved

Figure 2: Auto-tilting workflow for low-magnification, real-space tilting of Quantifoil grids based on circularity of included grid holes

Circle Recognition and Optimization function: It first reads each in grayscale, denoising with Gaussian blurring, and applying inverted Otsu thresholding followed by morphological closing to obtain a clean binary region corresponding to the object. From this binary image, it extracts the largest external contour, approximates it with a polygon using cv2.approxPolyDP (with the approximation tightness controlled by epsilon_ratio), and uses this polygon to generate two outputs: an overlay image where the detected region is highlighted in red on the original grayscale image, and a binary polygon mask saved for later fitting. The code then computes a standard circularity metric for each image, $4\pi\, Area/Perimeter^2$ where a value of 1 indicates a perfect circle, and stores this in a DataFrame alongside the image name. For more detailed geometry, it loads the polygon masks, refines the contour coordinates to subpixel accuracy with cornerSubPix, subsamples the boundary to a manageable number of points, and uses RANSAC with CircleModel to obtain a robust initial circle estimate and a set of inlier points that agree with that circle. Starting from this RANSAC initialization, it performs a nonlinear least-squares minimization of radial distance residuals using scipy.optimize.least_squares, yielding optimized center and radius parameters together with the root-mean-square error (RMSE) of the fit and an arc-coverage fraction that quantifies how much of the boundary is well described by a single circle. For each image, the function saves two diagnostic plots: an image with the fitted circle, center, and sampled contour overlaid, and an angular error plot showing radial deviation as a function of angle around the circle.

[1] R. Sweat, J. G. Park, R. Liang, R. Sweat, J. G. Park, and R. Liang, “A Digital Twin Approach to a Quantitative Microstructure-Property Study of Carbon Fibers through HRTEM Characterization and Multiscale FEA,” *Materials 2020, Vol. 13,* vol. 13, no. 19, Sep. 2020, doi: 10.3390/MA13194231.

[2] “Matplotlib — Visualization with Python.” Accessed: Dec. 17, 2025. [Online]. Available: https://matplotlib.org/