Magellon: The Next Generation of Cryo-EM Software

Frank Schotanus

Dr. Scott Stagg, Biology, Florida State University

Introduction

Background Information

Cryo-electron microscopy (Cryo-EM) is a high magnification imaging technique, enabling detailed imaging of biological macromolecules at near-atomic resolution[1]. Samples are frozen to preserve their native state. The data generated by cryo-EM requires advanced software for efficient data analysis. Current solutions, however, often suffer from outdated interfaces, inefficiencies, and steep learning curves, hindering research. The pioneering software packages Leginon and Appion demonstrated the power of automated data acquisition and real-time processing. Despite advances in automation, data collection and processing still require a good deal of manual involvement of an expert electron microscopist.

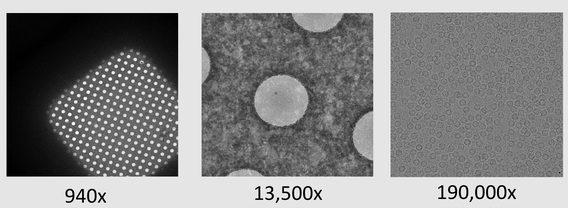

Figure 1. Example Cryo-EM Images

Abstract

As a response to the aforementioned setbacks to existing cryo-EM software, the Stagg lab at Florida State University, along with research teams from the University of Michigan and Scripps Research, have embarked on creating “Magellon”. Magellon aims to be the next generation of software for cryo-EM data collection and processing. Goals for Magellon include a fast and efficient data processing back end utilizing new industry standard tools and machine learning, a user-friendly graphical user interface, and a plugin system that allows developers in the community to create and implement their own plugins. Two of my tasks as a research assistant included updating the code to identify the holes in an image, and setting up a translator to import metadata from another platform.

Acknowledgements

Thank you to my mentor, Dr. Scott Stagg, as well as Behdad Khoshbin, and the research teams at the University of Michigan and Scripps Research.

Updating Hole Finding Code

Objective

A common task in cryo-EM data collection is locating the “holes” in the cryo-EM substrate that are targeted for data collection. Using a template image(right), I had to identify the holes in a given sample, one at a high magnification(center) and another at a lower magnification(left). I had to locate the center of each circle, draw a cross at its center and a circle around it, and assign each hole a confidence score based on how well it matched the template image.

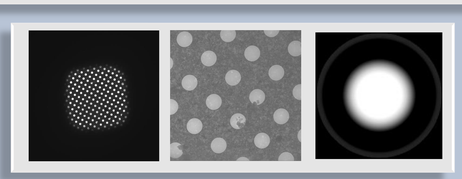

Figure 2. Sample images and template image

Methods

To identify the holes, I utilized the opencv library in Python, which is a library dedicated to computer vision and image processing. I used the matchTemplate function to locate the holes. This function works by sliding the template image across the other images. Each (X, Y) coordinate on the images is assigned a score from 0 to 1 based on how well the template matches in that location. This method of identifying holes results in several coordinates being returned for each hole. However, I only want a single coordinate with the best score for each hole. Using vector math, I identified all the points that were within one radius length of each other. Of those points, I took the one with the best score, and discarded the rest. I was then left with two lists: one with (X, Y) coordinates, and another with their corresponding scores. Using some draw functions in the opencv library, I was then able to draw a crosshair and circle around the final points.

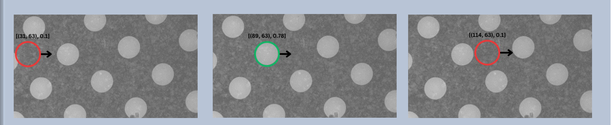

Figure 3. Visual demonstration of how matchTemplate works. The circle is the template image

```
for pt1 in zip(*points):
    point = (pt1[0] + w // 2, pt1[1] + h // 2)
    circle.append(point)
    circle_scores.append(pt1[2])
for pt in circle:
    hypot = np.array(point) - np.array(pt)
    if math.sqrt(pow(hypot[0], 2) + pow(hypot[1], 2)) > radius:
        circle.pop()
        circle_scores.pop()
        max_index = circle_scores.index(max(circle_scores))
        selected_scores.append(max(circle_scores))
        selected_points.append(circle[max_index])
        circle = []
        circle_scores = []
        break
```

Figure 4. Eliminating duplicates to get the best point for each hole

Results

The new code works efficiently to identify the holes in samples. It allows for changing the threshold when needed, depending on how clear the holes in the image are. Each hole has a score from 1-100 that ranks how well of a match it is. For future work on this project, I could work on a system where the template image is automatically resized, creating a new, more accurate template image, or being able to identify a higher magnification image inside of a lower magnification image.

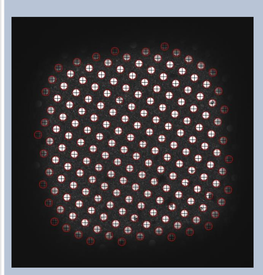

Figure 5. Identified holes on lower magnification image

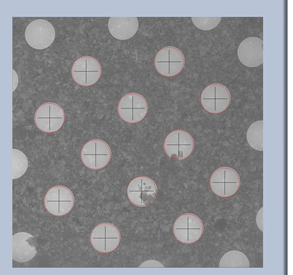

Figure 6. Identified holes on higher magnification image

Metadata Translation

Objective

There is a platform called EPU that has been used for the automation of data collection for cryo-EM. One of the goals of Magellon is to allow for the transfer of data from previous platforms like EPU. However, EPU and Magellon have different methods of structuring their metadata for images. EPU structures it using XML, while Magellon uses JSON. While the transition between these file formats is typically straightforward, challenges arise due to certain discrepancies. These challenges include accounting for differing nomenclature and trying to minimize unnecessary data transfer.

Methods

The first step in this process was to go into the EPU XML metadata and manually search the files for their Magellon metadata counterpart. For example, in EPU they have "NominalMagnification" whereas Magellon refers to it as "Magnification". To automate the translation from EPU to Magellon, I had to write some code in Python to parse the XML. To do that, I used the “etree” module from the “LXML” library. Etree is used to convert the hierarchy of XML into a tree. In this tree, each element is broken down into its “branch”. For example, in **Figure 7**, if you wanted to get to the "x" value of “Binning”, you would take the following path: **microscopeData/acquisition/camera/Binning/x**. This is the basic concept I used to locate the Magellon metadata fields within the EPU metadata. After locating each field, I put it in a Python dictionary, where the metadata field is the key, and its value between the brackets is the value. After iterating through the whole file, I am left with a dictionary of key-value pairs for every Magellon metadata field, ready to be used by Magellon.

```
<microscopeData>
        <acquisition>
            <camera>
                <Binning>
                    <x>1</x>
                    <y>1</y>
                </Binning>
            </camera>
    </acquisition>
</microscopeData>
```

Figure 7. Example of XML hierarchy

```
for field in fields:
    if path.endswith(':' + field):
        fields[field] = path
        val = root.find('.//' + fields[field], namespaces)  # get value of the field
        fields[field] = val.text  # assign the value
    if (path.endswith(':Key')):
        val = root.find('.//' + path, namespaces).text
        if (val in fields):
            path = path.replace(']/ns1:Key', ']/ns1:Value')  # a little hard coded, gets the value of the key
            fields[val] = root.find('.//' + path, namespaces).text  # assign the value
```

Figure 8. Identifying all the Magellon metadata fields and getting their values

Results

Although the implementation of this code into Magellon is still a work in progress, it shows promise for contributing to the versatility of Magellon as a platform. It is able to read in metadata files from EPU, and output a Magellon metadata file, including only the necessary metadata fields with their corresponding values.

Discussion

The development of software is always ongoing and everchanging. There will always be room for improvement, as is the case with my two projects. With my hole finding code I could implement an algorithm that automatically resizes the template image based off the input image, I could also work on creating a more accurate template image based off data from several thousand holes. For my metadata translation, the next step is to actually implement it into Magellon’s backend, which will require much more work. Overall, the future direction of Magellon will remain to be constantly working to adapt the software to our modern environment.

Literature Cited: 1. Sheibani, Sara et al. “Nanoscale Characterization of the Biomolecular Corona by Cryo-Electron Microscopy, Cryo-Electron Tomography, and Image Simulation.” Nature communications 12.1 (2021): 573–9.