Anhang F
ImageJ

Michael Entrup

April 2017

Inhaltsverzeichnis

1 Eine Einführung zu ImageJ

ImageJ ist ein in Java geschriebenes Programm zur wissenschaftlichen Bildverarbeitung. Auf der Website des Programms (`https://imagej.nih.gov`) heißt es „Image Processing and Analysis in Java“. ImageJ wird von einem ehemaligen Mitarbeiter des National Institutes of Health entwickelt, Wayne Rasband. Der Quellcode unterliegt nicht dem Urheberrecht, sondern ist unter „Public Domain“ freigegeben [Ras04].

Mit ImageJ2 entsteht im Moment eine neue Version, welche auf die Verarbeitung und Analyse von mehrdimensionalen Datensätzen ausgelegt ist. ImageJ2 ist trotzdem vollständig zu ImageJ1 kompatibel. Dies wird dadurch erreicht, dass die grafische Oberfläche von ImageJ2 weiterhin Zugriff auf alle Funktionen von ImageJ1 gewährt. Abbildung 1 zeigt einen Screenshot von ImageJ. Die neue Programm-Logik von ImageJ2 – ImgLib2 genannt – ist vollständig von der grafisch Oberfläche entkoppelt,

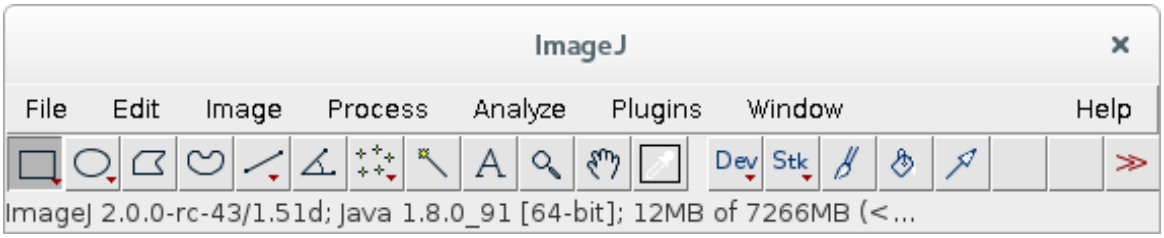


Abbildung 1: Das Hauptfenster von ImageJ (Debian Linux mit Gnome3). In der Statusleiste ist zu erkennen, dass ImageJ2 in der Version 2.0.0-rc-43 vorliegt und ImageJ1 in der Version 1.51d.

so dass man sie mit anderer Software nutzen kann. So ist es möglich Plugins zu schreiben, die auch von anderer Software aus dem SciJava-Projekt genutzt werden können. ImageJ2 und damit auch ImgLib2 sind noch nicht fertiggestellt, weshalb in dieser Arbeit ausschließlich mit Programmcode für ImageJ1 gearbeitet wird. Ausgeführt wird der Code jedoch in ImageJ2, bzw. Fiji. Letzteres wird im folgenden Abschnitt erklärt.

2 Fiji Is Just ImageJ

Neben ImageJ existiert die Distribution Fiji. Auf der Website[1] heißt es dazu: „Fiji is an image processing package—a “batteries-included” distribution of ImageJ, bundling a lot of plugins which facilitate scientific image analysis.“ Das bedeutet, dass neben den Grundfunktionen eine Vielzahl von Plugins vorinstalliert sind. Außerdem hat man Zugriff auf diverse Java-Bibliotheken, wie z. B. Apache Commons Math.

Aus Fiji stammen außerdem mehrere Funktionen, die mittlerweile Bestandteil von ImageJ sind. Mit Hilfe des Updaters erhält man automatisch Aktualisierungen für Plugins und ImageJ selbst. Der Script-Editor ermöglicht ein komfortables Schreiben von Macros und Scripten, dank Syntax-Hervorhebung und einer einfachen Auto-Vervollständigung.

Der Updater bietet noch eine besondere Funktion. Man kann eigene Update-Sites hinzufügen und erhält auf diese Weise Aktualisierungen zu Plugins, welche nicht Bestandteil von Fiji sind. Jeder Nutzer kann außerdem seine eigenen Update-Sites auf dem Webserver von ImageJ erstellen und damit die selbst geschriebenen Plugins der Allgemeinheit zur Verfügung stellen. Dieses Verfahren wird auch für das Plugin EFTEMj genutzt. Eine Anleitung zum Hinzufügen von EFTEMj zu ImageJ ist auf der GitHub-Website zum Plugin zu finden.[2]

[1] `http://imagej.net/Fiji`
[2] `https://github.com/m-entrup/EFTEMj`

Abbildung 2: Der Macro Recorder von ImageJ. Es wurden diverse Bearbeitungsschritte an einem Bild ausgeführt, zu denen der Recorder die entsprechenden Macro-Befehle auflistet.

3 ImageJ durch eigene Programme erweitern

Ein großer Vorteil von ImageJ ist die Erweiterbarkeit. Eine Vielzahl von verfügbaren Plugins[3] verdeutlicht, dass diese Erweiterbarkeit von vielen genutzt wird [Col07]. Diese einfache Möglichkeit neue Funktionen zu ergänzen beschränkt sich nicht nur auf Plugins, sondern es ist außerdem möglich Macros und Scripte zu schreiben. Durch das Wegfallen des Kompilierens sind Macros und Scripte sehr einfach zu nutzen. Besonders das Schreiben eines ersten Scriptes gestaltet sich als sehr einfach, da man aus einer von acht Programmier- bzw. Scriptsprachen[4] (8 bei Fiji, 6 bei ImageJ) wählen kann und zusätzlich mit Hilfe des Macro-Recorders passende Code-Schnipsel präsentiert bebommt.

Macros für ImageJ schreiben

Macros eignen sich besonders gut um Abläufe zu automatisieren. Dazu startet man den Macro-Recorder **[Plugins > Macros > Record...]** und führt die gewünschten Bearbeitungs- und Auswertungsschritte mit ImageJ aus. Im Textfeld des Macro-Recorders erscheint zu jeder genutzten Funktion ein Befehl der Form `run(''Find Edges'')`. Lässt sich die Funktion durch Parameter anpassen, so sind auch diese im ausgegebenen Befehl enthalten, z. B. `run(''Gaussian Blur...'', ''sigma=3'')`. Abbildung 2 zeigt den Macro Recorder, nachdem schon mehrere Bearbeitungsschritte

[3] `https://imagej.nih.gov/ij/plugins/`
[4] BeanShell, Clojure, Groovy, JavaScript, Python, R, Ruby und Scala (Stand: 29.06.2016)

Listing 1: Ein einfaches Macro für ImageJ

```
isSetSliceLabels = false;
# Ein Dialog wird angezeigt, bei dem man Yes, No und Cancel wählen kann:
if ( getBoolean("Include means as slice labels?") ) {
    isSetSliceLabels = true;
}
# Den Titel des gewählten Bildes auslesen:
title = getTitle();
# Nur die Kopie soll verändert werden:
run("Duplicate ... ", " title=" + title + "-norm duplicate");
# Der Schleifenkörper wird so oft ausgeführt, wie es Slices im Stack gibt:
for (i = 1; i <= nSlices; i++) {
    # Ein Slice des Stacks wird ausgewählt:
    setSlice(i);
    # Es werden verschiedene statistische Daten zum Slice ausgelesen:
    getStatistics(area, mean, min, max, std, histogram);
    # Nur mean wird benutzt, um den Slice pixelweise durch diesen Wert zu dividieren:
    run("Divide ... ", "value=" + mean + " slice");
    # Wurde es gewünscht, dann wird der alte Mittelwert als Slice-Label angegeben:
    if ( isSetSliceLabels ) {
        run("Set Label ... ", "label=" + mean);
    }
}
setSlice(1);
# Der dargestellte Wertebereich muss noch angepasst werden:
run("Enhance Contrast", "saturated=0.35");
```

mit ImageJ ausgeführt wurden. Ein Klick auf **Create** öffnet den Script-Editor und zeigt die aufgenommen Befehle an[5]. Mit Hilfe der Anleitung zur ImageJ Macro Sprache (`https://imagej.nih.gov/ij/developer/macro/macros.html`) kann man die einzelnen Befehle durch das Hinzufügen von Schleifen und Kontrollstrukturen zu einem erstes, komplexeren Macro erweitern. Viele Funktionen lassen sich mit Hilfe der Macro-Recorders nicht erfassen, weshalb die Website `https://imagej.nih.gov/ij/developer/macro/functions.html` all diese Funktionen aufführt, die zusätzlich zur Verfügung stehen. Die Liste wird laufend aktualisiert und führt zu vielen Funktionen hilfreiche Beispiele auf.

Listing 1 zeigt ein einfaches Macro, mit dessen Hilfe man den Mittelwert aller Bilder in einem Stack auf 1 normiert. Die `run()`-Befehle wurden vom Macro-Recorder generiert und anschließend durch String-Verkettung flexibler gestaltet. Die Kommentare, welche mit `#` anfangen erklären die einzelnen Schritte innerhalb des Macros.

[5] Alternativ kann man den Script-Editor über den Menü-Eintrag **[Plugins > Macro > New]** öffnen.

Einschränkungen von ImageJ Macros

Viele Aufgaben lassen sich sehr gut mit Hilfe von Macros lösen. Es gibt jedoch Aufgaben, für welche das Schreiben eines Macros nicht sinnvoll ist. Für gewöhnlich führt man mit den `run()`-Befehlen Funktionen so aus, wie es auch mit Hilfe der grafischen Oberfläche möglich ist. Häufig werden dabei neue Fenster geöffnet. Dabei kann es problematisch sein, dass die Befehle immer auf dem aktuell gewählten Bild ausgeführt wird. Möchte man ein anderes Bild nutzen, so muss man dieses über eine von zwei verfügbaren Methoden machen:

- Auswahl des Fensters mit Hilfe des Titels (`selectWindow("name")`), wobei es zu Problemen bei Namen mit Sonderzeichen kommen kann,
- Auswahl des Fensters mit Hilfe der eindeutigen ID (`selectImage(id)`).

In diesem Moment sollte man über die Nutzung einer Script-Sprache Nachdenken, da diese Bilder als Objekte zwischengespeichert werden können.

Beim Ausführen von Macros sollte man nicht dem Computer interagieren. Aktiviert man versehentlich ein falsches Bild, führt dies zu einem Fehler bei der Ausführung des Macros, oder zu einem falschen Ergebnis.

Ein weiter Nachteil von Macros ist die geringe Ausführungsgeschwindigkeit. Dies hängt damit zusammen, dass Änderungen an Bildern direkt ein Update des Bild-Fensters auslösen. Die Aktualisierung der grafischen Oberfläche ist jedoch ein Prozess, der vergleichsweise langsam abläuft. Dies gilt nicht nur für ImageJ Macros, sondern auch für andere Script- und Programmiersprachen. In einem Macro kann man deshalb den Batch-Modus mit `setBatchMode(arg)` aktivieren.[6] Änderungen an den Objekten der grafischen Oberfläche werden dadurch erst sichtbar, sobald der Batch-Modus beendet wird.

ImageJ lässt sich außerdem vollständig ohne grafische Oberfläche nutzen. Von der Kommandozeile des Betriebssystems ruft man ImageJ mit einem Macro und weiteren optionalen Parametern auf. Dieser Modus eignet sich sehr gut zum Auswerten von Bildern, die sich innerhalb eines, als Parameter übergebenen Ordners befinden. Die Ergebnisse sollte man dabei in eine Textdatei schreiben. Diese Vorgehensweise ist zu empfehlen, wenn sich jedes Bild einzeln auswerten lässt. Muss man mehrere Bilder miteinander verrechnen (Differenz bilden, Korrelation bestimmen und vieles mehr), wird verhältnismäßig viel Code für das Auswählen der Bilder notwendig sein. Die Verwendung einer Script-Sprache ist in diesem Fall empfehlenswert.

Scripte für ImageJ schreiben

ImageJ unterstützt eine Vielzahl von Script- und Programmiersprachen. Für alle unterstützten Sprachen eignet sich der Script-Editor **[Plugins > Macro > New]**

[6] Der Batch-Modus besitzt verschiedene Modi, die über den übergebenen Parameter aktiviert werden.

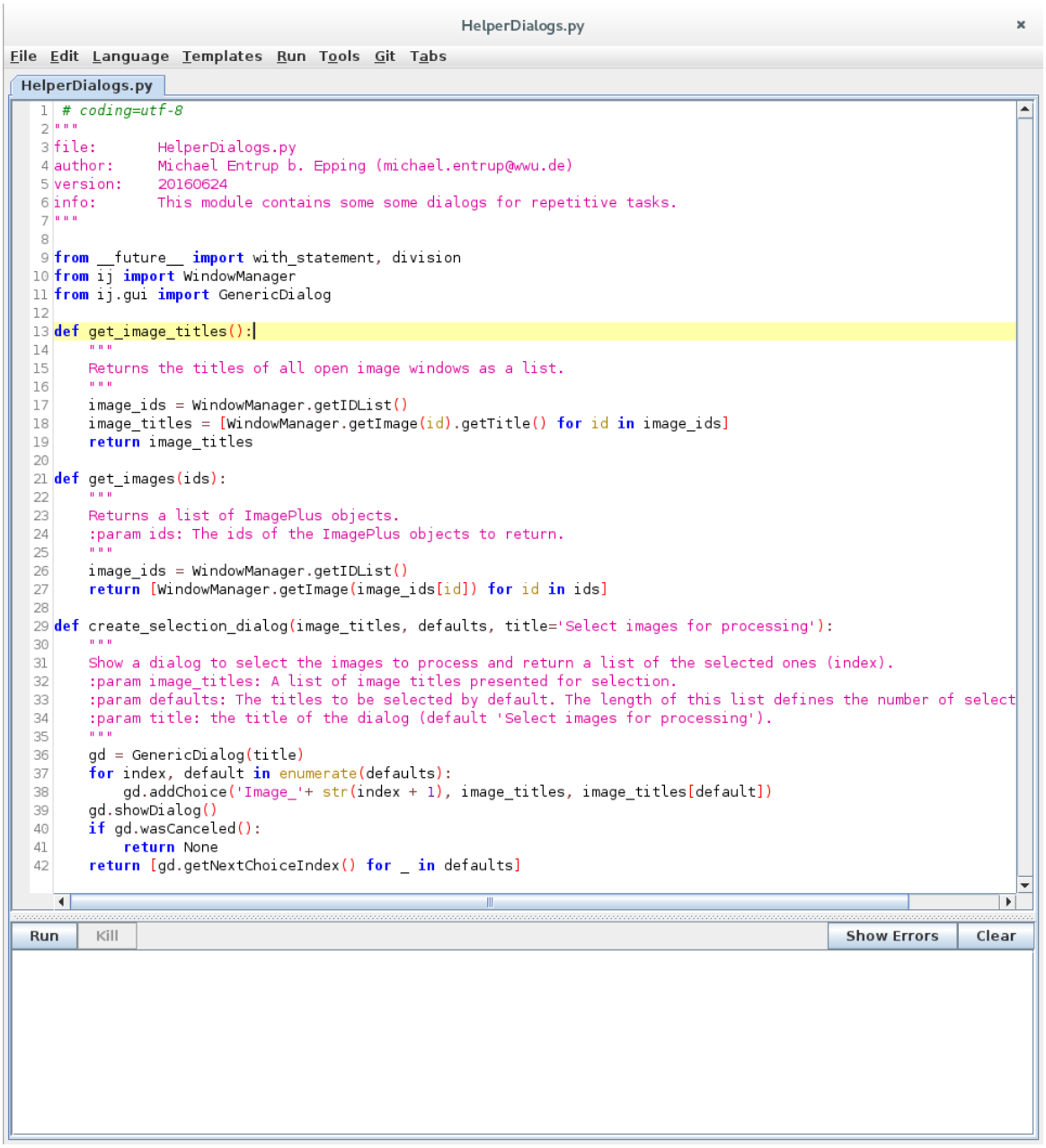


Abbildung 3: Der Script-Editor von ImageJ. Zu sehen ist ein in Python geschriebenes Script. Durch die Syntaxhervorhebung ist der Code einfach zu lesen.

zum Programmieren. Das Menü **[Language]** des Script-Editors führt alle verfügbaren Sprachen auf. Selektiert man eine der aufgelisteten Sprachen, wird die Syntax-Hervorhebung für diese aktiviert und ein Klick auf den Button **[Run]** führt das eingegebene Script mit dem passenden Interpreter aus.[7] Abbildung 3 zeigt den Script-Editor, welcher Python-Code auf Grund der Syntaxhervorhebung farblich darstellt.

Das Schreiben eines Scriptes ähnelt eher dem Entwickeln eines Java-Plugins, als dem Erstellen eines Macros. Es besteht voller Zugriff auf die Programmierschnittstelle von ImageJ und die installierten Bibliotheken (zum Beispiel Apache Commons Math).[8] Damit man darauf zugreifen kann, muss man jedoch einen Import eines Paketes, beziehungsweise einer Klasse ausführen. In Python wird beispielsweise `from ij import IJ` verwendet, um die Klasse `IJ` aus dem Paket `ij` zu importieren. Mit `IJ.run(imp, ''Gaussian Blur...'', ''sigma=5'')` kann man anschließend eine Funktion von ImageJ aufrufen, wie es in einem ImageJ-Macro mit `run(''Gaussian Blur...'', ''sigma=5'')` möglich ist. Der zusätzliche Parameter `imp` ist dabei eine Variable, die auf ein `ImagePlus`-Objekt verweist. Dieses `ImagePlus`-Objekt repräsentiert das Bild, auf das die Operation angewendet werden soll. Der Variable `imp` kann man auf verschiedene Weisen ein Bild zuweisen. Eine Möglichkeit ist die Verwendung von `imp = IJ.open(''Pfad/zur/Datei'')`[9], womit man eine Datei vom lokalen Dateisystem lädt.

Mit dem Macro-Recorder kann man außerdem Befehle für Script-Sprachen und Java aufnehmen[10]. Am wichtigsten ist jedoch die API-Dokumentation von ImageJ, die man unter `https://imagej.nih.gov/ij/developer/api/` findet. Diese Website wird automatisch aus dem JavaDoc von ImageJ generiert und listet sämtliche Klassen mit allen enthaltenen Methoden auf.[11] Außerdem kann der Sourcecode von ImageJ sich als hilfreich erweisen. Über die URL `https://imagej.nih.gov/ij/developer/source/` ruft man diesen im Browser auf. Viele Funktionen von ImageJ sind so implementiert, dass sie das Ergebnis einer Berechnung als Bild anzeigen (zum Beispiel die Kreuzkorrelation). Ein Blick in den Sourcecode von `FFTMath`[12] verrät, wie man mit Hilfe der *Fast Hartley Transform* (`FHT`)[13] in wenigen Schritten eine Kreuzkorrelation durchführt, deren Ergebnis als ein `ImagePlus`-Objekt vorliegt.

Python Grundlagen für ImageJ

Mit EFTEMj-pyLib und EFTEMj-pyScripts enthält EFTEMj diverse Scripte für ImageJ, welche die Auswertung von ESI Aufnahmen erleichtern. Zu den Scripten ge-

[7] Die Script-Sprachen müssen nicht kompiliert werden und können somit direkt innerhalb von Java ausgeführt werden (vergleiche `https://en.wikipedia.org/wiki/Interpreter_(computing)`).

[8] Im Englischen ***A**pplication Programming **I**nterface*, abgekürzt als **API**.

[9] Statt einem Pfad kann man auch eine URL angeben, um ein Bild aus dem Internet zu laden. Wird Die Methode ohne Parameter aufgerufen, kann der Nutzer die Datei selber auswählen.

[10] Zur Auswahl stehen neben **Macro**, **JavaScript**, **BeanShell** und **Java** (Stand: 29.06.2016).

[11] Leichter zu merken ist `http://javadoc.imagej.net/`, wo man die Dokumentation zu fast allen Projekten findet, die mit ImageJ in Verbindung stehen.

[12] `https://imagej.nih.gov/ij/developer/source/ij/plugin/FFTMath.java.html`

[13] `https://imagej.nih.gov/ij/developer/source/ij/process/FHT.java.html`

hören eine Driftkorrektur und diverse Scripte zur Berechnung von Jump-Ratio- und Element-Verteilungs-Bildern. Da der Quellcode zu umfangreich ist, um ihn hier abzudrucken, soll auf das Repository bei Github verwiesen werden, welches über die URL `https://github.com/m-entrup/EFTEMj` zu erreichen ist.

Bei den Scripten in EFTEMj-pyLib und EFTEMj-pyScripts wird Python verwendet, da diese Programmiersprache besonders leicht zu erlernen ist. Außerdem ist der Code sehr gut zu lesen, da in Python Code-Blöcke durch das Einrücken und nicht durch geschwungene Klammern definiert werden. Python-Code ließt sich außerdem viel eher wie natürliche Sprache, da zum Beispiel das Schlüsselwort `not` zum Negieren verwendet wird und nicht das Ausrufungszeichen.

3.0.1 ImageJ verwendet Jython 2.5.3[14]

Python ist eine Interpreter-Sprache und benötigt somit ein Programm, welches den Code ausführt. Der meist genutzte Interpreter ist CPython, welcher in C geschrieben ist. Für Java existiert der Interpreter Jython, welcher in ImageJ in der Version 2.5.3 enthalten ist. Das bedeutet, dass noch nicht alle Funktionen verfügbar sind, die Python in der aktuellen Version 2.7.x bietet[15]. Eine große Einschränkung ist dieser Versions-Rückstand nicht.

Leider muss man in Jython auf viele Python-Bibliotheken verzichten. NumPy, matplotlib und viele mehr sind in C geschrieben und lassen sich dadurch nicht direkt in Jython importieren.

3.0.2 Importieren von selbst geschriebenen Modulen

Damit die modularisierte Drifkorrektur funktioniert, muss das Haupt-Script die benötigten Module finden. Scripte werden bei ImageJ im Ordner `plugins/Scripts/` gespeichert. Unterordner bestimmen dabei, in welchem Menü das Script angezeigt wird. So sorgt der Pfad `plugins/Scripts/Plugins/Utilities/Record_Desktop.py` dafür, dass im Menü **[Plugins > Utilities]** der Eintrag **[Record Desktop]** erscheint. Der Unterstrich im Dateinamen ist wichtig, damit das Script im Menü angezeigt wird. Beim Menüeintrag werden Unterstriche durch Leerzeichen ersetzt und die Dateiendung wird entfernt. Dateien, die keinen Unterstrich enthalten, werde nicht angezeigt. Sie können jedoch als Modul in anderen Scripten geladen werden. Dazu muss man jedoch die Variable `sys.path` um den Pfad zum Script erweitern. Einfacher ist es, wenn man ein Package[16] im Ordner `jars/Lib` ablegt. Dieser Pfad ist schon in `sys.path` enthalt, weshalb Jython dort nach Modulen sucht.

Bei EFTEMj-pyLib und EFTEMj-pyScripts werden die einzelnen Scripte außerdem in einer JAR-Datei[17] verpackt. Dies dient dazu die vielen Script-Datein einfacher über

[14] Stand: 29.06.2016

[15] Python besitzt zwei aktuelle Versionen. Da viele Projekte auf Version 2.7.x basieren, wird diese parallel zu Version 3.4.x noch mit Updates versorgt.

[16] Ein Package ist ein Ordner der Python-Dateien (Module) enthält, wobei die Datei `__init__.py` unbedingt erforderlich ist.

[17] Dabei handelt es sich im Prinzip um eine ZIP-Datei

eine ImageJ Update-Site zu verteilen. `EFTEMj-pyLib.jar` enthält das Python-Modul EFTEMj_pyLib und wird im Ordner `jars/Lib` abgelegt. Dadurch kann man in einem Python-Script mit Hilfe von `import EFTEMj_pyLib` das Modul importieren und die enthaltenen Scripte nutzen. `EFTEMj-pyScripts_.jar` enthält einzelne Scripte, die ImageJ im Menü **[Plugins > EFTEMj]** anzeigt. Damit dies funktioniert, muss innerhalb der JAR-Datei die zuvor beschriebene Verzeichnisstruktur eingehalten werden. Der Pfad `/scripts/Plugins/EFTEMj/ESI/Calculate_Jump-ratio.py` sorgt dafür, dass im Menü **[Plugins > EFTEMj > ESI]** der Eintrag **[Calculate Jump-ratio]** erscheint. `EFTEMj-pyScripts_.jar` wird dazu im Ordner `plugins` abgelegt[18].

4 Python-Scripte zur Auswertung von ESI Daten

Dieser Abschnitt beschreibt diverse in Python geschriebene Scripte, welche Teil von EFTEMj sind. Den Code vollständig zu beschreiben würde den Umfang dieser Arbeit unnötig erhöhen. Es wird deshalb nur grob skizziert, wie vorgegangen wird und welche Auswirkungen das auf die Implementierung hat. Der vollständige Code ist im zugehörigen Repository bei GitHub finden, welches über die URL `https://github.com/m-entrup/EFTEMj` zu erreichen ist.

In Abschnitt 3 wurde die Nutzung von Python-Modulen in ImageJ beschrieben. Für die Auswertung von ESI Daten wird davon Gebrauch gemacht. Die Programmlogik ist in Module ausgelagert, die sich wiederverwenden lassen.

Die Driftkorrektur ist ein gutes Beispiel für die Verwendung von Modulen. Das Script `CorrectDrift.py` enthält Funktionen, um die Drift zu bestimmen, die zur Korrektur notwendigen Verschiebungsvektoren zu ermitteln und anschließend die Verschiebung durchzuführen. Die Bestimmung der Drift lässt sich dabei mit zwei verschiedenen Methoden durchführen. Mit Hilfe der Kreuzkorrelation bestimmt das Script `CrossCorrelation.py` die Verschiebung von zwei Bildern zueinander. Mit dem Script `pySIFT.py` steht eine Implementierung des Scale-invariant feature transform (SIFT) Algorithmus zur Verfügung, mit dessen Hilfe ebenfalls die Verschiebung von zwei Bildern zueinander bestimmt werden kann. SIFT wurde von Henriette Merkel auf die Eignung zur Driftkorrektur von TEM-Bildern untersucht. Die Ergebnisse können in ihrer Bachelorarbeit nachgelesen werden [Mer16]. Die Driftkorrektur kann durch den modularen Aufbau sehr einfach um weitere Methoden zur Detektion der Drift erweitert werden.

Auch die gesamte Driftkorrektur ist so aufgebaut, dass sie in möglichst vielen Scripten verwendet werden kann. Der Aufruf von

```
corrected_image_list = CorrectDrift.get_corrected_images(limage_list)
```

genügt, um eine Liste von Bildern zu korrigieren. Mit dem optinalen Parameter `mode` kann man von SIFT auf Kreuzkorrelation umschalten. Da die Verwendung von SIFT zur Detektion von Drift fast nur Vorteile gegenüber der Verwendung von Kreuzkorrelation hat, ist SIFT die als Standard genutzte Methode.

[18] In neueren Versionen von ImageJ (ab 1.51.?) ist es auch möglich `EFTEMj-pyScripts_.jar` in `jars` abzulegen.

Jump-Ratio Bilder berechnen

Um eine Jump-Ratio-Map zu berechnen, genügt eine Division des Post-edge Bilders durch das Pre-edge Bild. Dazu dient bei ImageJ der *ImageCalculator*.[19] Auf Grund von Drift ist das erhaltene Ergebnis meist nicht sehr überzeugend, wie in Abbildung 4a zu sehen ist. Es sind Strukturen zu erkennen, die eigentlich nicht vorhanden sind. Im Vergleich zur Jump-ratio-Map, mit vorheriger Korrektur der Drift (siehe Abbildung 4b), sind die Unterschiede gut sichtbar. Letztlich genügen drei Befehle, um eine Driftkorrektur auszuführen und eine Jump-Ratio-Map zu erzeugen. Diese drei Befehle sind in Listing 2 aufgeführt. Die ersten drei Zeilen zeigen die Importe, die notwendig sind, um die zu verwendenden Befehle zu laden. Der erste Befehl führt die Driftkorrektur aus, wobei sich der eigentliche Programmcode im Modul `CorrectDrift` befindet. Als Eingabe dient ein Tuple aus zwei Bildern und ein String mit der zu verwendenden Methode. Das Ergebnis ist ein Stack aus zwei Bildern, der in Zeile 8 zu einer Liste umgewandelt wird. Da die Länge der Liste bekannt ist, können wird die beiden Einträge direkt in die Variablen `img1` und `img2` entpacken. In Zeile 9 wird der *ImageCalculator* mit den Parametern `Divide` und `create` aufgerufen, womit `img2` durch `img1` geteilt wird und für das Ergebnis ein neues Bild erzeugt wird, welches wir der Variable `img_ratio` zuweisen.

Das eigentliche Script ist mit fast 80 Zeilen Code wesentlich umfangreicher, da der Benutzer einen *GenericDialog* präsentiert bekommt, um die zu verarbeitenden Bilder, sowie die zu verwendende Methode zur Detektion der Drift zu wählen. Außerdem wird die Darstellung der Jump-Ratio-Map unter Berücksichtigung von Mittelwert und Standardabweichung optimiert. Wichtig ist dabei, dass nur die Darstellung der Gleitkommazahlen als 256 Graustufen angepasst wird. Die eigentlichen Werte werden von dieser Optimierung nicht verändert.

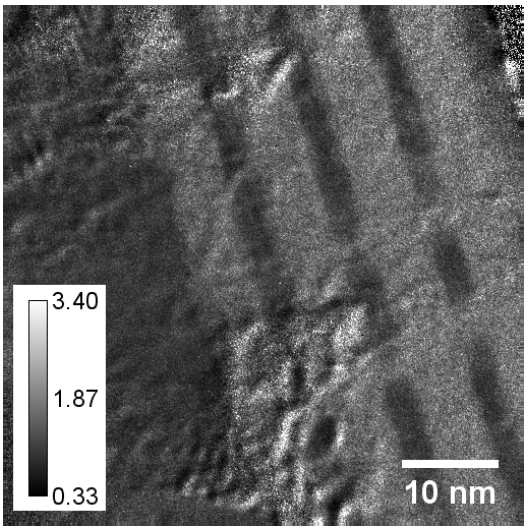
Die relative Dicke der Probe bestimmen

Analog zur Berechnung einer Jump-Ratio-Map lässt sich die relative Dicke einer Probe bestimmen. Dazu werden zwei Bilder benötigt: Das erste Bild wird ohne Spaltblende in der energiedispersiven Ebene aufgenommen. Somit tragen alle Elektronen zum Bild bei (I_0). Zum zweiten Bild tragen nur die Elektronen des ZLP bei (I_G). Es wird entsprechend eine Spaltblende in der energiedispersiven Ebene verwendet. Mit Hilfe der Gleichung

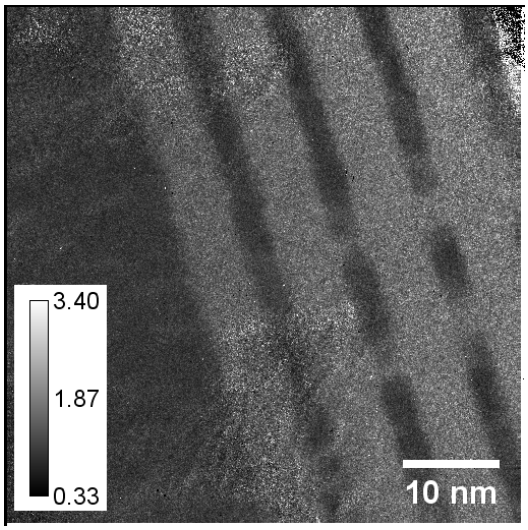
$$\frac{t}{\lambda} = \log\left(\frac{I_\mathrm{G}}{I_0}\right) \tag{1}$$

berechnet man anschließend Pixel für Pixel die relative Dicke ${}^{t}/_{\lambda}$, wobei t die absolute Dicke darstellt und λ die freie Weglänge der Elektronen in der Probe ist. Eine Herleitung dieser Gleichung ist zum Beispiel in *Transmission Electron Microscopy and*

[19] `https://imagej.nih.gov/ij/docs/guide/146-29.html#toc-Subsection-29.13`



(a) Jump-ratio-Map, erstellt ohne Korrektur der Drift.



(b) Jump-ratio-Map, erstellt nach Korrektur der Drift mit Hilfe von SIFT.

Abbildung 4: Eine Korrektur der Drift ist essenziell um richtige Ergebnisse zu erhalten. (a) zeigt welche Auswirkung eine Drift von 0,9 nm auf die Jump-rati-Map hat. (b) zeigt im Vergleich dazu das Ergebnis nach erfolgreicher Korrektur der Drift.

Listing 2: Die wichtigsten drei Zeilen Code bei der Berechnung der Jump-Ratio-Map.

```
from EFTEMj_pyLib import CorrectDrift
from EFTEMj_pyLib import Tools
from ij.plugin import ImageCalculator

corrected_stack = CorrectDrift.get_corrected_stack((img1_in, img2_in),
mode='SIFT'
)
img1, img2 = Tools.stack_to_list_of_imp(corrected_stack)
img_ratio = ImageCalculator().run('Divide create', img2, img1)
img_ratio.show()

```

Listing 3: Die wichtigsten drei Zeilen Code bei der Berechnung der relativen Dicke.

```
1  from EFTEMj_pyLib import CorrectDrift
2  from EFTEMj_pyLib import Tools
3  from ij.plugin import ImageCalculator
4
5  corrected_stack = CorrectDrift.get_corrected_stack((img_G_in, img_0_in),
6  mode='SIFT'
7  )
8  img_G, img_0 = Tools.stack_to_list_of_imp(corrected_stack)
9  img_ratio = ImageCalculator().run('Divide create', img_G, img_0)
10 IJ.run(img_ratio, "Log", "")
11 img_ratio.show()
12
```

Diffractometry of Materials von B. Fultz und J. M. Howe zu finden [FH08, S. 175ff]. Die wichtigsten Befehle sind in Listing 3 aufgeführt. Im Vergleich zu Listing 2 gibt es eine entschiedene Änderung: In Zeile 10 wird für jeden Pixel des Bildes `img_ratio` der Logarithmus berechnet. Analog zur Jump-Ratio-Map gibt das Listingen nur die wichtigsten Befehle wieder. Das Vollständige Script umfasst ein Vielfaches an Code, um dem Benutzer die Auswahl der Bilder, sowie der zur Driftkorrektur verwendeten Methode mit Hilfe einer grafischen Oberfläche zu ermöglichen.

In Abbildung 5 ist die Map der relativen Dicke an einem Beispiel gezeigt. Wie schon bei der Jump-Ratio-Map führt eine erfolgreiche Korrektur der Drift zu einer deutlichen Verbesserung des Ergebnisses.

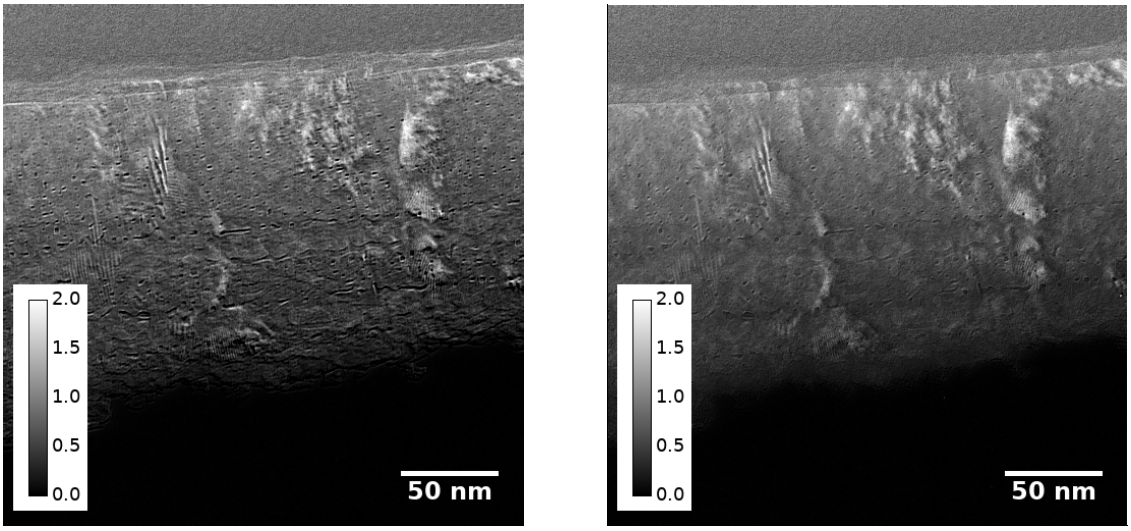
5 Weitere Beispiel-Scripte

Ein EEL Spektrum auslesen

In Abschnitt 1.3 der Dissertation wird beschrieben, wie bei einem modernen TEM das EEL Spektrum aus der Parrallel-EELS-Aufnahme extrahiert wird. Dieser Schritt der Datenverarbeitung wird normalerweise von der Software durchgeführt, mit der man die Kamera des Mikroskops ansteuert. Das aufgenommene Bild wird somit als eindimensionaler Datensatz abgespeichert, der die Intensität in Abhängigkeit vom Energieverlust enthält.

Das aufgezeichnete Bild lässt sich auch speichern und mit ImageJ verarbeiten.

Elemental-Mapping mit der 3-Fenster-Methode



(a) Relative Dicke, erstellt ohne Korrektur der Drift.

(b) relative Dicke, erstellt nach Korrektur der Drift mit Hilfe von SIFT.

Abbildung 5: Eine Korrektur der Drift ist essenziell um richtige Ergebnisse zu erhalten. (a) zeigt welche Auswirkung eine Drift von 1,1 nm auf die Map der relativen Dicke hat. (b) zeigt im Vergleich dazu das Ergebnis nach erfolgreicher Korrektur der Drift.

Literatur

[Col07] Tony J. Collins, *ImageJ for microscopy*, BioTechniques **43** (2007), 25–30, http://dx.doi.org/10.2144/000112517.

[FH08] Brent Fultz and James M. Howe, *Transmission Electron Microscopy and Diffractometry of Materials*, Springer-Verlag, Berlin, Heidelberg, 2008.

[Mer16] Henriette Merkel, *Merkmalsbasierte Driftkorrektur in der Transmissionselektronenmikroskopie*, Westfälischen Wilhelms-Universität Münster, Bachelorarbeit, 2016.

[Ras04] Wayne Rasband, *ImageJ Disclaimer*, https://imagej.nih.gov/ij/disclaimer.html, 2004.