# Wykrywanie i śledzenie komórek glistopodobnych w programie ImageJ

(Worm-like cell detection and tracking in ImageJ)

Artur Rosa

Praca magisterska

**Promotor:** dr Andrzej Łukaszewski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

27 lipca 2020

**Streszczenie**

Polskie streszczenie

---

English abstract

# Spis treści

# Rozdział 1.

# Wprowadzenie

## 1.1. Wstęp

...

## 1.2. Background

### 1.2.1. Jakich danych i do czego potrzebują biolodzy

...

### 1.2.2. Pozyskiwanie danych

...

# Rozdział 2.

# Wykrywanie i śledzenie komórek

## 2.1. Opis problemu

### 2.1.1. Obrazy wejściowe

...

### 2.1.2. Pożądany efekt

...

## 2.2. Powiązane prace

...

## 2.3. Wykrywanie komórek

### 2.3.1. Wstęp

Problem opisany w sekcji 2.1. zdefiniowany jest dla nagrań spod mikroskopu. Postanowiłem jednak najpierw rozwiązać podobny problem, ale zdefiniowany dla pojedynczego obrazu. Rozwiązanie tego problemu mogłoby z łatwością zostać uogólnione na stos obrazów (nagranie). W tym rozdziale opiszę rozwiązanie uproszczonego problemu: oznaczanie komórek widocznych na pojedynczym obrazie. Przez „oznaczenie komórki" mam na myśli odnalezienie łamanej przechodzącej przez środek komórki, a więc jej kręgosłupa.

### 2.3.2. Dane wejściowe

Oznaczenie wszystkich komórek widocznych na obrazie można rozłożyć na dwa osobne problemy:

1. Określenie liczby oraz lokalizacji poszczególnych komórek
2. Odnalezienie kształtu poszczególnych komórek.

W niniejszej pracy zdecydowałem się nie rozwiązywać automatycznie pierwszego problemu. Zamiast tego użytkownik zobowiązany jest ręcznie zaznaczyć dokładnie jeden punkt wewnątrz każdej komórki widocznej na obrazie. Wymóg ten dotyczy tylko pierwszej klatki nagrania, co opiszę dokładniej w dalszej części pracy (2.4.).

Danymi wejściowymi są zatem obraz $\mathbf{I}$ oraz zbiór punktów $\mathbf{P}$ lokalizujących komórki.

### 2.3.3. Wybór kanału i wstępne przetwarzanie obrazu

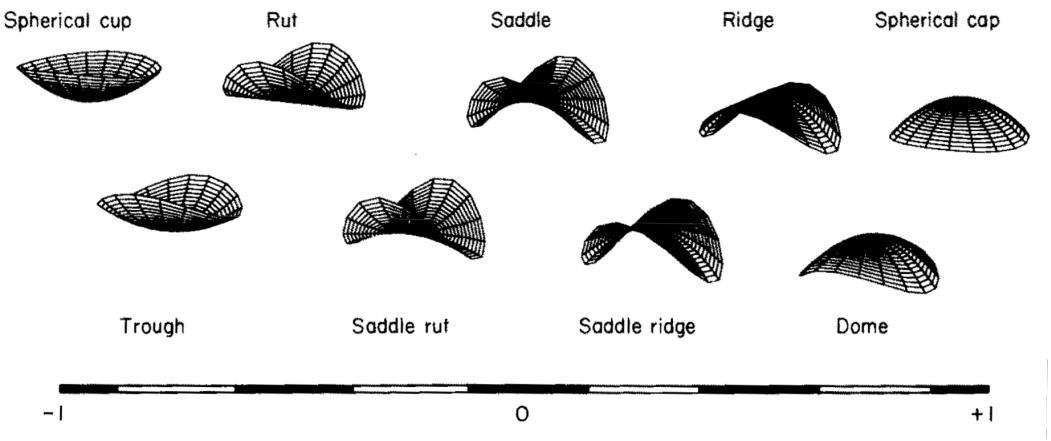
Obraz wejściowy składa się z dwóch podstawowych kanałów (2.1.1.). Chcąc jak najdokładniej oznaczyć początek i koniec komórki, a także miejsca ich podziału, postanowiłem wybrać kanał, który zawiera wyraźną informację o krawędziach w tych miejscach. O ile kanał z fluorescencją mógłby bardzo dobrze sprawdzić się do określenie liczby oraz lokalizacji poszczególnych komórek (w przypadku ich niewielkiej liczby), o tyle drugi kanał zawiera dużo dokładniejszą informację na temat krawędzi komórek.

Celem wstępnego przetwarzania obrazu wejściowego jest w tym przypadku oddzielenie poszczególnych komórek od otoczenia, zachowując przy tym informację na temat ich krawędzi. W ramach pracy przeprowadziłem wiele testów mających na celu odnalezienie narzędzia spełniającego ten cel. Krawędzie otrzymywane za pomocą filtrów czy algorytmów do ich wyszukiwania często były nieciągłe. Dobierając inne parametry często pojawiały się niechciane zaznaczenia wewnątrz komórek.

Ostatecznie zdecydowałem się użyć narzędzia które nie jest związane z wykrywaniem krawędzi. Zaobserwowałem, że zdjęcia zrobione tą techniką mają pewne specyficzne właściwości. Komórki są na nich dość równomiernie oświetlone, przez co dobrze widać ich „tubowaty” kształt. Każdą komórkę otacza też ciemne obramowanie.

#### Mapa indeksów kształtu

Postanowiłem spróbować wyliczyć mapę indeksów kształtu (ang. shape index map) dla obrazu wejściowego, na którym wcześniej zastosowałem rozmycie gaussowskie. Indeks kształtu to liczba z przedziału $[-1, 1]$ przyporządkowana na podstawie




Rysunek 2.1: Skala indeksu kształtu z podziałem na dziewięć kategorii. [1]

„lokalnego kształtu" powierzchni. Jest to skaloniezminnicza (ang. scale invariant) miara, która dzieli powierzchnię na obszary wypukłe, wklęsłe i hiperboliczne (rysunek 2.1)[1]. Obraz wejściowy interpretowany jest tutaj jako mapa wysokości. Zastosowanie rozmycia gaussowskiego w pierwszym kroku jest niezbędne w tym przypadku ze względu na nieodporność tej miary na szum obecny na obrazie wejściowym. Do wyliczenia mapy indeksów skorzystałem z wtyczki dla programu ImageJ autorstwa Johannesa Schindelina[2].

Okazało się, że na wynikowej mapie otoczenie komórek interpretowane jest jako obszar wklęsły, w przeciwieństwie do samych komórek, których wnętrze oznaczane jest indeksami kształtów wypukłych. Własność ta zachodzi na większości testowych obrazów nawet przy stosunkowo dużym zagęszczeniu komórek.

### 2.3.4. Szkieletyzacja i wstępna detekcja komórek

Zgodnie z powyższą obserwacją, możemy łatwo oddzielić komórkę od jej otoczenia na obrazie ustalając pewien próg $t \approx 0$. Załóżmy przez chwilę, że binaryzując w ten sposób mapę indeksów kształtu otrzymamy obraz $\mathbf{I}_{bin}$, na którym każdy piksel leżący wewnątrz dowolnej komórki będzie miał wartość 1, natomiast każdy piksel należący do zewnętrznego obrysu dowolnej komórki (nie należący do komórki, lecz sąsiadujący z pikselem należącym do niej) będzie miał wartość 0. Przy takim założeniu każda komórka jest niezależną „wyspą" na binarnym obrazie $\mathbf{I}_{bin}$. Chcąc odnaleźć „kręgosłup" komórki (łamaną przechodzącą przez jej środek) chcemy tak naprawdę znaleźć łamaną, która jest równoodległa od jej krawędzi. Bardzo podobny problem rozwiązują algorytmy do wyznaczania szkieletu. Ich celem jest odnalezienie zbioru punktów równoodległych od co najmniej dwóch brzegów.

Na potrzeby tej pracy do szkieletonizacji użyta została implementacja algorytmu „3D thinning algorithm"[3] w formie pluginu dla programu ImageJ[4]. Mimo

iż wtyczka pozwala na szkieletyzację obrazów 3D, w tym przypadku została użyta do przetworzenia pojedynczego obrazu 2D. Wynikiem szkieletyzacji jest binarny obraz o pewnych właściwościach. Każdy aktywny piksel można przyporządkować do trzech grup:

- końcówki – mają mniej niż 2 sąsiadujące aktywne piksele
- węzły – mają więcej niż 2 sąsiadujące aktywne piksele
- połączenia – mają dokładnie 2 sąsiadujące aktywne piksele.

Przedstawiając szkielet jako graf końcówki tworzyłyby wierzchołki o stopniu równym 1 lub 0, wierzchołki o większych stopniach przedstawiałyby zbiory sąsiadujących ze sobą węzłów, natomiast krawędzie reprezentowałyby zbiory sąsiadujących ze sobą połączeń (zakończonych zbiorem węzłów lub końcówką). Taką reprezentację grafową można uzyskać za pomocą kolejnej wtyczki dla programu ImageJ tego samego autora[5]. Poza standardowymi informacjami wierzchołki i krawędzie utworzonego za jej pomocą grafu przechowują zbiory pikseli które reprezentują.

Ze względu na specyficzny kształt komórki można zaobserwować następującą właściwość: po przeprowadzeniu szkieletyzacji obrazu $\mathbf{I}_{bin}$, o ile przyjęte wcześniej założenie jest spełnione, szkielet każdej z komórek składa się dokładnie z dwóch końcówek i połączeń między nimi. Łamaną opisującą zbiór połączeń można z powodzeniem nazwać „kręgosłupem komórki". Mając do dyspozycji zbiór punktów $\mathbf{P}$ lokalizujących komórki można teraz w łatwy sposób odnaleźć dla każdej z nich graf ją opisujący (zawierający dwa wierzchołki i jedną krawędź). Jednym ze sposobów może być wyszukanie dla każdego punktu ze zbioru $\mathbf{P}$ krawędź która znajduje się najbliżej tego punktu, gdzie odległość między punktem a krawędzią zdefiniowana jest jako odległość między punktem, a najbliższym pikselem, który należy do zbioru opisywanego przez tę krawędź.

### 2.3.5. Wybór krawędzi w węzłach

Niestety przyjęte założenie o tym, że każdy piksel należący do zewnętrznego obrysu dowolnej komórki będzie miał wartość 0, nie zawsze jest spełnione. W realistycznym scenariuszu zdarza się, że jedna z końcówek komórki znajdują się na tyle blisko innej komórki, że wstępne przetwarzanie i progowanie obrazu nie powoduje ich rozdzielenia na obrazie binarnym. Czasem artefakty widoczne na obrazie wejściowym powodują, że wyspa na obrazie binarnym zawiera nie tylko komórkę, ale także fragment innego kształtu. W zdecydowanej większości takich przypadków kręgosłup komórki można opisać spójnym podgrafem o stopniu 2 grafu reprezentującego szkielet wyspy zawierającej komórki.

Wstępna detekcja kręgosłupa danej komórki polega, tak jak w scenariuszu optymistycznym, na odnalezieniu krawędzi $\{v_0, u_0\}$ w grafie $\mathbf{G}$ leżącej najbliżej punktu

opisującego komórkę, a następnie na stworzeniu z niej i jej wierzchołków nowego grafu $\mathbf{S}_0$. Tak utworzony kręgosłup rozszerzany jest później zgodnie z następującym algorytmem:

**for each** $v \in \{v_0, u_0\}$ **do**
  **loop**
    $\mathrm{E}_v \leftarrow$ podzbiór krawędzi grafu $E(\mathbf{G}) - E(\mathbf{S}_n)$ incydentnych do $v$,
        które nie tworzą cyklu w grafie $\mathbf{S}_n$
    **if** $\mathrm{E}_v = \emptyset$ **then**
      **break**
    **end if**
    $\{v, u\} \leftarrow \mathbf{argmax}_{e \in \mathrm{E}_v}\ q_v(e)$
    $\mathbf{S}_{n+1} \leftarrow \mathbf{S}_n \cup (\{u\}, \{\{v, u\}\})$
    $v \leftarrow u$
  **end loop**
**end for**

Funkcja $q_v(e)$ jest tutaj funkcją oceny, która służy do wyboru najmocniej związanej krawędzi. Siła wiązania zdefiniowana jest tutaj jako najniższa wartość indeksu kształtu (oryginalnego obrazu) dla piksela leżącego na krawędzi $e$, w okolicy wierzchołka $v$ (w odległości nie większej niż pewna stała $d$ od środka masy pikseli tworzących węzeł $v$).

Takie rozwiązanie sprawdza się dobrze dla komórek, których kręgosłup znajduje się w grafie $\mathbf{G}$, oraz jego zakończenia w tym grafie mają stopień 1. Pomijam na ten moment przypadek, gdy szukany kręgosłup komórki nie istnieje w grafie $\mathbf{G}$. W pozostałych przypadkach przynajmniej jedno z zakończeń szukanego kręgosłupa $\mathbf{S}$ ma w grafie $\mathbf{G}$ stopień większy niż 1. Zatem, o ile funkcja oceny $q_v(e)$ sprawdzi się dobrze jeśli chodzi o dobór krawędzi należących do szukanego kręgosłupa komórki, uzyskany na końcu algorytmu kręgosłup $\mathbf{S}_n$ będzie nadgrafem szukanego kręgosłupa $\mathbf{S}$. W takim przypadku „nadmiarowa" część kręgosłupa należy do szkieletu innej komórki lub jest wynikiem artefaktu widocznego na oryginalnym obrazie. Drugi przypadek pozostawiam do ręcznego rozwiązania użytkownikowi. Pierwszy natomiast, nachodzące na siebie kręgosłupy komórek, rozwiązuję automatycznie.

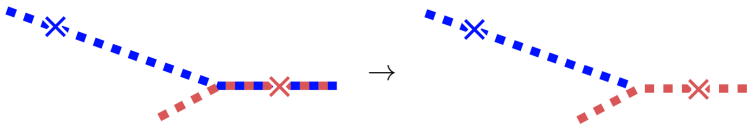
### 2.3.6. Rozwiązywanie konfliktów

Rozdzielanie nachodzących na siebie komórek nazwałem „rozwiązywaniem konfliktów". Niech $\mathbf{\Sigma}_0$ będzie zbiorem wszystkich znalezionych kręgosłupów, natomiast $X(\mathbf{\Sigma})$ zbiorem par kręgosłupów nachodzących na siebie należących do zbioru $\mathbf{\Sigma}$. Rozwiązywanie konfliktów przebiega w następujący sposób:

**while** $X(\mathbf{\Sigma}_n) \neq \emptyset$ **do**
  $\{S_x, S_y\} \leftarrow$ dowolna para ze zbioru $X(\mathbf{\Sigma})$
  $\{S'_x, S'_y\} \leftarrow$ wynik rozwiązania konfliktu pomiędzy $S_x$ i $S_y$
  $\mathbf{\Sigma}_{n+1} \leftarrow (\mathbf{\Sigma}_n - \{S_x, S_y\}) \cup \{S'_x, S'_y\}$
**end while**

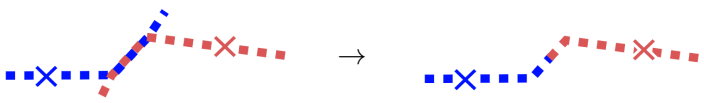
Sposób w jaki rozwiązywany jest konflikt pomiędzy dwoma kręgosłupami, zależy od tego w jaki sposób nachodzą one na siebie. Kluczową rolę odgrywają tutaj także punkty lokalizujące komórki. W dalszej części pisząc, o punkcie charakterystycznym komórki, będę miał na myśli punkt leżący na kręgosłupie będący najbliżej punktu lokalizującego komórkę. Wyróżniłem 4 typy nakładania się kręgosłupów.

### Nakładanie się częściowe z końcówką



W tym przypadku część wspólna przydzielana jest do jednego z kręgosłupów w całości. Są dwie możliwości: oba punkty charakterystyczne komórki leżą poza częścią wspólną lub dokładnie jeden z nich znajduje się w części wspólnej. Gdyby oba punkty leżały w części wspólnej, to ze względu na sposób konstrukcji szkieletów, podczas procesu rozszerzania dobierane byłyby te same krawędzie, a więc końcowo oba kręgosłupy byłyby dokładnie takie same. W drugim przypadku sprawa jest prosta – część wspólna zostaje przydzielona do kręgosłupa którego punkt charakterystyczny leży w części wspólnej. W pierwszym przypadku część wspólna przydzielana jest do tego kręgosłupa z którym jest mocniej związana (siłę wiązania określam w podobny sposób jak w algorytmie konstrukcji kręgosłupa).

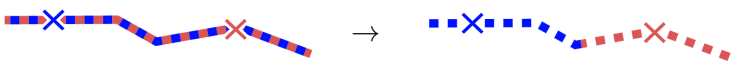
### Nakładanie się częściowe



W przypadku takiego rodzaju nakładania oba punkty charakterystyczne komórki znajdują się poza częścią wspólną. Rozwiązanie konfliktu polega tutaj na zna-

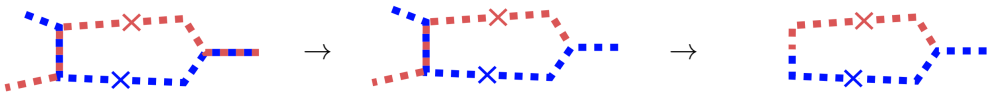
lezieniu najsłabszego punktu części wspólnej (mającego najniższy indeks kształtu), a następnie odpowiednim skróceniu obu kręgosłupów do tego punktu.

#### Pełne pokrycie



Sposób rozwiązania tego konfliktu jest bardzo podobny do sposobu rozwiązania nakładania się częściowego. Najsłabszy punkt wyszukiwany jest w tym przypadku pomiędzy punktami charakterystycznymi komórek.

#### Nakładanie się wielokrotne



Dwa kręgosłupy mogą mieć więcej niż jeden konflikt równocześnie. Jest to rzadkie, ale realistyczne. W tym przypadku w każdej iteracji rozwiązywany jest jeden z takich konfliktów za pomocą sposobów opisanych powyżej.

### 2.3.7. Uzyskiwanie linii łamanej korekta końcowa

Po rozwiązaniu wszystkich konfliktów każdy kręgosłup opisuje szkielet pewnej komórki. Kolejnym krokiem jest konwersja reprezentacji grafowej (która zawiera zbiór pikseli opisujących szkielet) do linii łamanaj. W takiej formie użytkownik będzie mógł ręcznie modyfikować zaznaczenia komórek korzystając ze standardowego interfejsu programu ImageJ. Pierwsza wersja linii łamanej powstaje ze wszystkich punktów tworzących krawędzie i końcówki kręgosłupa, a także ze środków masy jego węzłów. Następnie geometria łamanej jest upraszczana za pomocą algorytmu Ramera–Douglasa–Peuckera[6]. Algorytm ten eliminuje punkty, których usunięcie nie wpływa znacząco na kształt łamanej tj. po ich usunięciu odległość uproszczonej łamanej od oryginalnej jest nie większa niż pewna przyjęta stała.

### 2.3.8. Korekta końcówek

Ostatnim krokiem oznaczania komórek jest korekta końcówek. Potrzeba korekty wynika z algorytmu szkieletyzacji. Proces ten polega na erodowaniu binarnego obrazu — z tego powodu końcówki szkieletu komórki często oddalone są od faktycznej krawędzi na oryginalnym obrazie. Dotyczy to tylko zakończeń które reprezentowane są przez wierzchołki mające w oryginalnym szkielecie obrazu stopień równy 1. Nie dzieje się tak w przypadku, gdy binarny obraz komórki łączy się w tym miejscu z inną komórką, a zakończenie powstało na skutek rozwiązania konfliktu.

Właściwej lokalizacji zakończenia komórki szukam na półprostej tworzonej przez ostatni odcinek łamanej, która ją opisuje. Mając do dyspozycji obraz binarny na podstawie którego powstał szkielet, szukam zakończenia wyspy leżącego najbliżej początku półprostej. Jeśli punkt ten leży nie dalej niż pewna przyjęta stała, staje się on nowym zakończeniem łamanej. Stała którą przyjąłem była nie większa niż przewidywana szerokość komórki.

Proces ten powtarzany jest dla każdego zakończenia komórki wymagającego poprawienia.

## 2.4. Śledzenie komórek w czasie

Do tej pory opisywałem sposób na wykrywanie i oznaczanie komórek na pojedynczym obrazie, mając do dyspozycji punkty lokalizujące komórki wprowadzone przez użytkownika. Oryginalny problem dotyczył jednak stosu obrazów przedstawiającego proces rozwoju komórek w czasie. Uogólnienie opisanego sposobu polega na automatycznym wyznaczaniu dla każdej komórki kilku potencjalnych punktów ją lokalizujących, na podstawie łamanej opisującej tę komórkę w poprzedniej klatce nagrania naniesionej na aktualną klatkę. Następnie wybierane są te punkty (i stworzone przez nie kręgosłupy), które dały najlepsze efekty względem pewnej funkcji oceny.

Jeśli nowy kręgosłup komórki jest znacząco krótszy niż łamana opisująca ją na poprzednim obrazie, prawdopodobnie oznacza to, że nastąpił podział komórki. W takim przypadku wyszukiwany jest kolejny kręgosłup. Tym razem kandydatów na punkty lokalizujące poszukuję tylko po jednej stronie łamanej. Jeśli pierwszy kręgosłup powstał na podstawie punktu leżącego bliżej końca łamanej, jego bliźniaczy kręgosłup prawdopodobnie znajduje się bliżej jej początku i vice versa.

Po odnalezieniu wszystkich nowych kręgosłupów zgodnie z powyższą procedurą, następuje etap rozwiązywania konfliktów, konwersji na linie łamane i ostatecznej korekty końcówek (2.3.6.–2.3.8.).

Niech $\mathbf{C}_n$ będzie zbiorem łamanych opisujących komórki na $n$-tym obrazie stosu. Niech $\phi_\sigma(d)$ będzie funkcją która liczbą z przedziału $[0, 1]$ przyporządkowuje punkty

leżące na łamanej $\sigma$, proporcjonalnie do odległości od jej początku (przyjmijmy, że linia łamana ma zdefiniowany „kierunek", tzn. ma początek i koniec). Niech $S_n(p)$ będzie szkieletem komórki wyznaczonym zgodnie z metodą opisaną wcześniej (2.3.2.–2.3.5.) dla punktu lokalizującego $p$ i $n$-tej klatki nagrania. Cały proces można zobrazować następującym algorytmem:

**for each** $n \in \{0, 1, \ldots, n-1\}$ **do**
  $\mathbf{\Sigma}_{n+1} \leftarrow \emptyset$
  **for each** $\sigma \in \mathbf{C}_n$ **do**
    $\Sigma \leftarrow \{S_{n+1}(\phi_\sigma(d)) \mid d \in \mathrm{D}\}$
    $S \leftarrow \mathbf{argmax}_{S \in \Sigma}\ q_\sigma(S)$
    $\mathbf{\Sigma}_{n+1} \leftarrow \mathbf{\Sigma}_{n+1} \cup \{S\}$
    **if** $|S| < 0.9 \cdot |\sigma|$ **then**
      $\Sigma' \leftarrow \{S_{n+1}(\phi_\sigma(d')) \mid d' \in \mathrm{D}'\}$
      $S' \leftarrow \mathbf{argmax}_{S' \in \Sigma}\ q_\sigma(S')$
      $\mathbf{\Sigma}_{n+1} \leftarrow \mathbf{\Sigma}_{n+1} \cup \{S'\}$
    **end if**
  **end for**
  $\hat{\mathbf{\Sigma}}_{n+1} \leftarrow$ wynik rozwiązania konfliktów w zbiorze $\mathbf{\Sigma}_{n+1}$
  $\mathbf{C}_{n+1} \leftarrow$ wynik konwersji na łamane i korekcie końcówek elementów zbioru $\hat{\mathbf{\Sigma}}_{n+1}$
**end for**

W algorytmie pojawia się zbiór $D \subseteq [0, 1]$ którego elementy dobrałem empirycznie jako $\{0.2, 0.4, 0.6, 0.8\}$. Zbiór $D' \subset D$ to zbiór, który zależy od wcześniejszego wyboru najlepszego kandydata na kręgosłup komórki. Jeśli w danej iteracji kręgosłup $S$ powstał na podstawie punktu lokalizującego leżącego na pierwszej połowie łamanej, będzie to zbiór $\{d \in D \mid d > 0.5\}$ w przeciwnym wypadku będzie to pozostała część zbioru $D$.

## 2.5. Interakcja ze strony użytkownika

Niestety powyższe działania nie sprawdzą się dobrze w każdym przypadku. Jest wiele czynników które utrudniają poprawną detekcję komórek. Większość z nich związana jest ze słabą jakością nagrań. Obrazy są mocno zaszumione, często występują na nich artefakty, czasem płytka na której poruszają się komórki minimalnie się przesuwa, a innym razem mikroskop traci ostrość na kilka klatek nagrania. Zapewne można próbować rozwiązać te problemy automatycznie, ale na pewno pojawią się nowe, nie rozważane wcześniej przypadki. Właśnie z tego powodu postanowiłem zapewnić użytkownikowi możliwość interaktywnego poprawiania automatycznie wyznaczonych komórek, ale także całkowicie manualne kontynuowanie detekcji, zachowując przy tym informacje o historii komórek.

Komórki przedstawione są w oknie programu jako standardowe zaznaczenia programu ImageJ i można je edytować za pomocą standardowego interfejsu. Poza tym plugin stworzyłem dwa dodatkowe narzędzia dla użytkownika. Pierwsze z nich służy do rozcinania komórek w miejscach, w których się one dzielą. Mimo że algorytm śledzenia ma mechanizm odpowiedzialny za zauważenie takich zmian, zdarza się że dzieje się to zbyt późno (komórki jeszcze przez pewien czas po podziale znajdują się bardzo blisko siebie).

Drugie narzędzie służy do skracania końcówek zaznaczeń. Zauważyłem, że częstym problemem jest to, że zaznaczenie wybiega poza krawędzie komórki. Najczęściej dzieje się to ze względu na występujące na obrazie artefakty lub punkty stałe na płytkach znajdujące się blisko zakończeń komórek. Żeby przyspieszyć proces poprawiania takich zaznaczeń można skorzystać z narzędzia do skracania końcówek, które działa podobnie do „gumki" w popularnych programach graficznych.

Chcąc ręcznie oznaczyć komórki w kolejnej klatce (zamiast wyliczać je automatycznie na podstawie poprzedniej), użytkownik może skorzystać z opcji duplikowania klatki. Dzięki temu w aktualnej klatce pojawią się wszystkie zaznaczenia z klatki poprzedniej, które następnie można ręcznie dopasować do ich nowej pozycji.

# Rozdział 3.

# Opis implementacji

## 3.1. Kod źródłowy

Kod źródłowy projektu zamieszczony został w publicznie dostępnym repozytorium w serwisie GitHub. Repozytorium można znaleźć pod adresem `https://github.com/rossinek/cell-detector-imagej-plugin`.

Program zaimplementowany jest jako wtyczka do programu ImageJ, napisana w języku Java. Do automatyzacji procesu budowy wykorzystany został Apache Maven. Wszystkie zależności zostały uwzględnione w pliku konfiguracyjnym dla Mavena, który odpowiada również za ich pobranie.

Plugin był rozwijany głównie na systemie macOS Catalina, ale został również przetestowany na systemie Windows 10.

## 3.2. Kompilacja i uruchomienie

Do poprawnego zbudowania pluginu z kodu źródłowego należy upewnić się że zainstalowane są wymagane zależności. Poniższa instrukcja zakłada, że system wyposażony jest w:

- Apache Maven w wersji większej lub równej `3.3.9`
- Java w wersji 8.

Aby zbudować projekt należy skorzystać z komendy `mvn`, która spowoduje pobranie wszystkich zależności, kompilację kodu źródłowego i wywołanie testów automatycznych.

Zbudowany plugin dla programu ImageJ można znaleźć w folderze `target` pod nazwą `Mtbt_Plugin-{wersja}.jar`. Tak spakowany plugin można zainstalować w

programie ImageJ (lub Fiji) wklejając go do folderu `jars` w miejscu gdzie zainstalowany jest program.

Aby uruchomić plugin w ramach własnej niezależnej instancji ImageJ, po zbudowaniu projektu można uruchomić główną metodę programu komendą `mvn exec:java -Dexec.mainClass="dev.mtbt.Main"`.

## 3.3. Obsługa pluginu

...

## 3.4. Struktury danych

...

## 3.5. Architektura

...

## 3.6. Opis wykorzystanego API ImageJ

...

## 3.7. Błędy w implementacji ImageJ i sposoby na ich obejście

...

## 3.8. Opis sposobu dalszego rozwoju, interfejsy

...

# Rozdział 4.

# Zakończenie

## 4.1. Podsumowanie

...

## 4.2. Ograniczenia wynikające z zastosowanych metod

...

## 4.3. Dalszy rozwój

...

# Bibliografia


[1] Jan J Koenderink, Andrea J van Doorn, *Surface shape and curvature scales*, Image and Vision Computing, 10:557–565, 1992.

[2] Johannes Schindelin, *Shape Index Map*, 2010, `https://imagej.net/Shape_Index_Map`.

[3] Ta-Chih Lee, Rangasami L. Kashyap, Chong-Nam Chu, *Building skeleton models via 3-D medial surface/axis thinning algorithms*, Computer Vision, Graphics, and Image Processing, 56(6):462–478, 1994.

[4] Ignacio Arganda-Carreras, *Skeletonize3D*, 2.1.1, 2017, `https://imagej.net/Skeletonize3D`.

[5] Ignacio Arganda-Carreras, *AnalyzeSkeleton*, 3.3.0, 2018, `https://imagej.net/AnalyzeSkeleton`.

[6] Wikipedia, *Ramer–Douglas–Peucker algorithm*, `https://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm`.