

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

29 lipca 2020

Streszczenie

Polskie streszczenie

English abstract

Spis treści

1. Wprowadzenie	7
1.1. Wstęp	7
1.2. Background	7
1.2.1. Jakich danych i do czego potrzebują biolodzy	7
1.2.2. Pozyskiwanie danych	7
2. Wykrywanie i śledzenie komórek	9
2.1. Opis problemu	9
2.1.1. Obrazy wejściowe	9
2.1.2. Pożądany efekt	9
2.2. Powiązane prace	9
2.3. Wykrywanie komórek	9
2.3.1. Wstęp	9
2.3.2. Dane wejściowe	10
2.3.3. Wybór kanału i wstępne przetwarzanie obrazu	10
2.3.4. Szkieletyzacja i wstępna detekcja komórek	11
2.3.5. Wybór krawędzi w węzłach	12
2.3.6. Rozwiązywanie konfliktów	13
2.3.7. Uzyskiwanie linii łamanej korekta końcowa	15
2.3.8. Korekta końcówek	16
2.4. Śledzenie komórek w czasie	16
2.5. Interakcja ze strony użytkownika	17

3. Opis implementacji	19
3.1. Kod źródłowy	19
3.2. Kompilacja i uruchomienie	19
3.3. Obsługa pluginu	20
3.3.1. Lokalizacja i wstępna detekcja komórek	21
3.3.2. Śledzenie komórek w czasie	21
3.3.3. Analiza danych	22
3.3.4. Manualna modyfikacja komórek	23
3.3.5. Nazewnictwo komórek oraz ich orientacja	25
3.3.6. Importowanie i eksportowanie komórek	26
3.4. Dokumentacja techniczna	26
3.4.1. Struktura projektu	26
3.4.2. Główne struktury danych	27
3.4.3. Architektura umożliwiająca dalszy rozwój	28
3.4.4. Wykorzystywane API ImageJ	29
3.4.5. Brakujące lub niespójne API ImageJ	30
4. Zakończenie	33
4.1. Podsumowanie	33
4.2. Ograniczenia wynikające z zastosowanych metod	33
4.3. Dalszy rozwój	33
Bibliografia	35

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

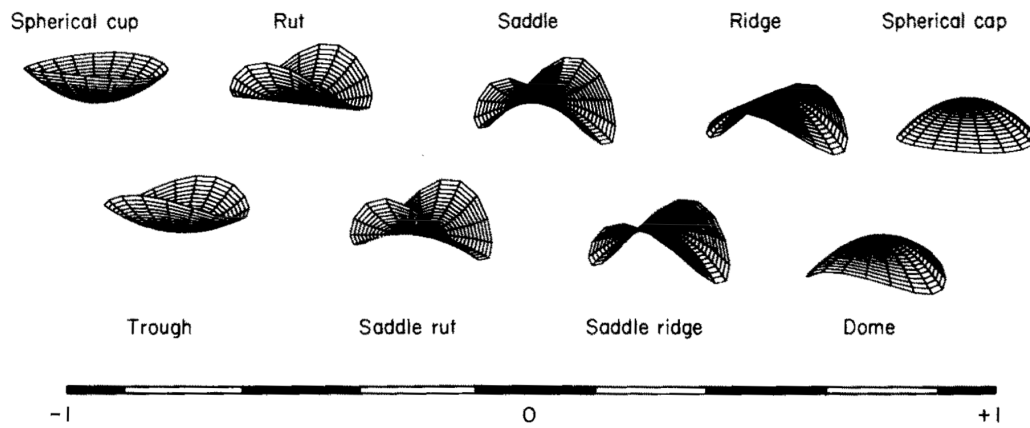
Obraz wejściowy składa się z dwóch podstawowych kanałów (2.1.1.). Chcąc jak najdokładniej oznaczyć początek i koniec komórki, a także miejsca ich podziału, postanowiłem wybrać kanał, który zawiera wyraźną informację o krawędziach w tych miejscach. O ile kanał z fluorescencją mógłby bardzo dobrze sprawdzić się do określenia 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 Johanna 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 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 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} krawędzi 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
     $E_v \leftarrow$  podzbiór krawędzi grafu  $E(\mathbf{G}) - E(\mathbf{S}_n)$  incydentnych do  $v$ ,
      które nie tworzą cyklu w grafie  $\mathbf{S}_n$ 
    if  $E_v = \emptyset$  then
      break
    end if
     $\{v, u\} \leftarrow \operatorname{argmax}_{e \in E_v} q_v(e)$ 
     $\mathbf{S}_{n+1} \leftarrow \mathbf{S}_n \cup (\{u\}, \{\{v, u\}\})$ 
     $v \leftarrow u$ 
  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 Σ_0 będzie zbiorem wszystkich znalezionych kręgosłupów, natomiast $X(\Sigma)$ zbiorem par kręgosłupów nachodzących na siebie należących do zbioru Σ . Rozwiązywanie konfliktów przebiega w następujący sposób:

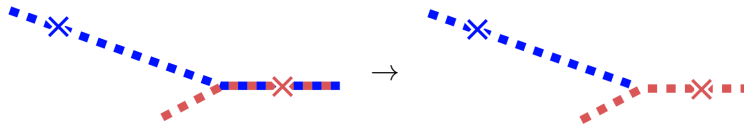
```

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

```

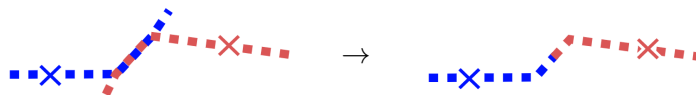
Sposób w jaki rozwiązywany jest konflikt pomiędzy dwoma kręgosłupami, zależy od tego w jaki sposób nachodzą one na siebie. Kluczową rolę odgrywają tutaj także punkty lokalizujące komórki. W dalszej części pisząc, o punkcie charakterystycznym komórki, będę miał na myśli punkt leżący na kręgosłupie będący 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).

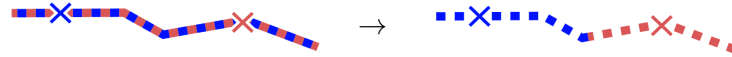
Nakładanie się częściowe



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

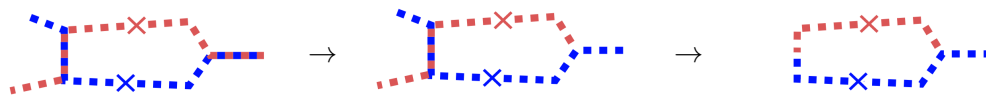
lezeniu najsłabszego punktu części wspólnej (mającego najniższy indeks kształtu), a następnie odpowiednim skróceniu obu kręgosłupów do tego punktu.

Pełne pokrycie



Sposób rozwiązywania tego konfliktu jest bardzo podobny do sposobu rozwiązywania 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 łamanej. 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 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 σ , 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  $n \in \{0, 1, \dots, n-1\}$  do
   $\Sigma_{n+1} \leftarrow \emptyset$ 
  for each  $\sigma \in C_n$  do
     $\Sigma \leftarrow \{S_{n+1}(\phi_\sigma(d)) \mid d \in D\}$ 
     $S \leftarrow \operatorname{argmax}_{S \in \Sigma} q_\sigma(S)$ 
     $\Sigma_{n+1} \leftarrow \Sigma_{n+1} \cup \{S\}$ 
    if  $|S| < 0.9 \cdot |\sigma|$  then
       $\Sigma' \leftarrow \{S_{n+1}(\phi_\sigma(d')) \mid d' \in D'\}$ 
       $S' \leftarrow \operatorname{argmax}_{S' \in \Sigma} q_\sigma(S')$ 
       $\Sigma_{n+1} \leftarrow \Sigma_{n+1} \cup \{S'\}$ 
    end if
  end for
   $\hat{\Sigma}_{n+1} \leftarrow$  wynik rozwiązania konfliktów w zbiorze  $\Sigma_{n+1}$ 
   $C_{n+1} \leftarrow$  wynik konwersji na łamane i korekcie końcówek elementów zbioru  $\hat{\Sigma}_{n+1}$ 
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. Z pewnością można próbować rozwiązać te problemy automatycznie, ale na pewno pojawiają 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



Rysunek 3.1: Menu programu ImageJ rozszerzone o wtyczki zaimplementowane na potrzeby tej pracy.

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

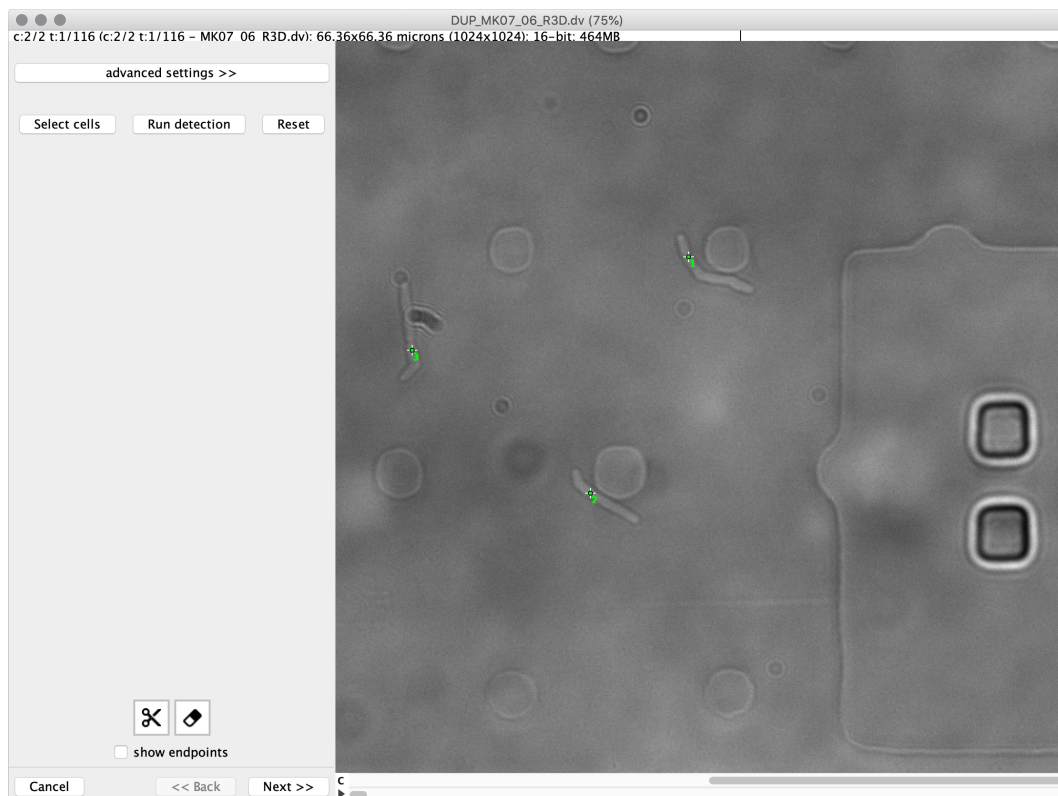
Aby uruchomić plugin w ramach własnej niezależnej instancji ImageJ, po zbudowaniu projektu można uruchomić główną metodę programu komendą `mvn exec:java -Dexec.mainClass="dev.mtbt.Main"`.

3.3. Obsługa pluginu

Po zainstalowaniu pluginu w programie ImageJ pojawi się nowe menu **Development** (rysunek 3.1) w którym znajdują się narzędzia zaimplementowane na potrzeby tej pracy. Poza głównym pluginem (**Cell detector**) znajdują się tam jeszcze dodatkowe wtyczki służące do importu oraz eksportu komórek, a także narzędzia pomocne w rozwoju programu. Główny plugin podzielony jest na trzy kroki:

1. Lokalizacja i wstępna detekcja komórek
2. Śledzenie komórek w czasie
3. Analiza danych

Przed uruchomieniem pluginu należy otworzyć nagranie które chcemy analizować. Po jego uruchomieniu zobaczymy okno pluginu przedstawiające pierwszy krok. Interfejs pluginu na każdym etapie składa się z kilku stałych i kilku zmiennych (w zależności od aktualnego kroku) części składowych. Po prawej stronie znajduje się zawartość standardowego okna wielokanałowego stosu obrazów ImageJ. Dwa suwaki w dolnej części służą do wyboru kanału oraz klatki nagrania. Wybór kanału ma wpływ na to, na którym kanale odbywa się detekcja. Obok suwaka do wyboru klatki znajduje się przycisk start/stop służący do odtwarzania nagrania. W górnej części panelu znajdującego się po lewej stronie znajdują się kontrolki specyficzne dla danego kroku pluginu. W jego dolnej części znajdują się narzędzia przydatne na każdym etapie działania pluginu, a także przyciski do zmiany kroku.



Rysunek 3.2: Pierwszy krok pluginu służący do wstępnej detekcji komórek.

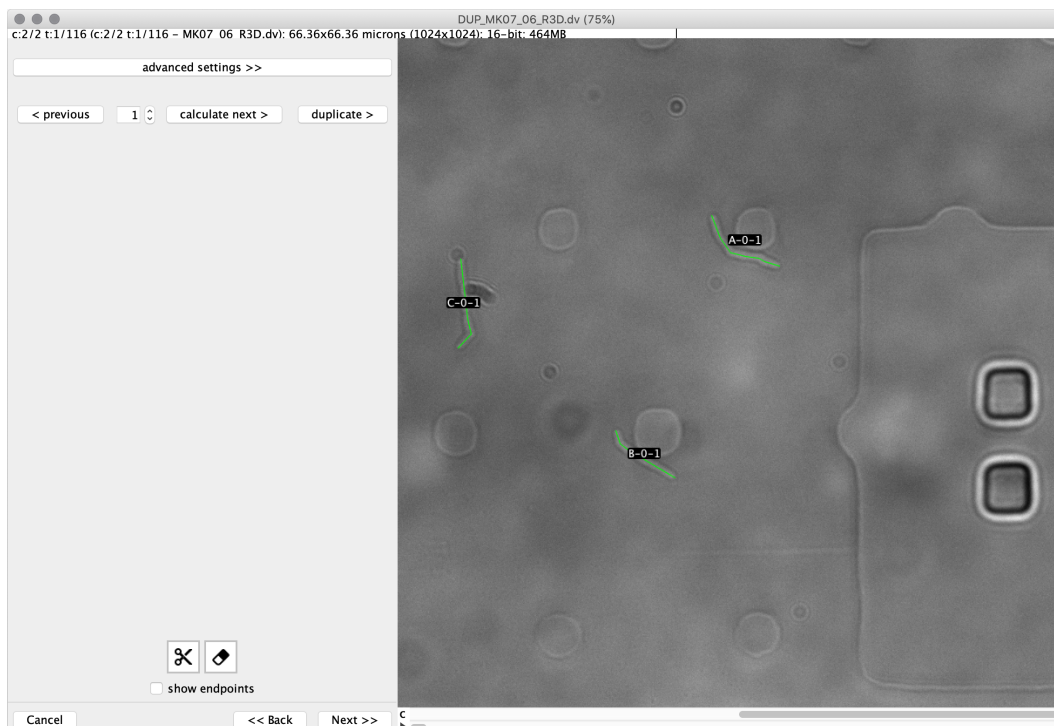
3.3.1. Lokalizacja i wstępna detekcja komórek

Pierwszy etap polega na zaznaczeniu przez użytkownika punktów lokalizujących komórki, a następnie uruchomieniu wstępnej detekcji (rysunek 3.2). W tym celu należy wcisnąć przycisk **Select cells**, zaznaczyć każdą komórkę widoczną na pierwszej klatce nagrania klikając w nią kursorem, a następnie wcisnąć przycisk **Run detection**. W celu poprawienia wstępnej selekcji można skorzystać z przycisku **Reset**, który usuwa cały uzyskany do tej pory efekt.

W dodatkowym panelu **advanced settings** ukryte zostały dodatkowe parametry sterujące procesem detekcji komórek (2.3.), takie jak siła rozmycia gaussowskiego czy wartość progowa użyte na etapie wstępnego przetwarzania obrazu. Można tam również włączyć podgląd mapy indeksów kształtu, a także szkieletu na podstawie których przebiega proces detekcji.

3.3.2. Śledzenie komórek w czasie

Kolejnym krokiem jest śledzenie komórek w czasie (rysunek 3.3). Użytkownik na tym etapie może zdecydować w jaki sposób chce dalej pracować. Może skorzystać z opcji automatycznego wyliczenia jednej bądź kilku kolejnych klatek (przycisk **calculate next**) lub zduplikować aktualnie zaznaczone komórki do kolejnej klatki a



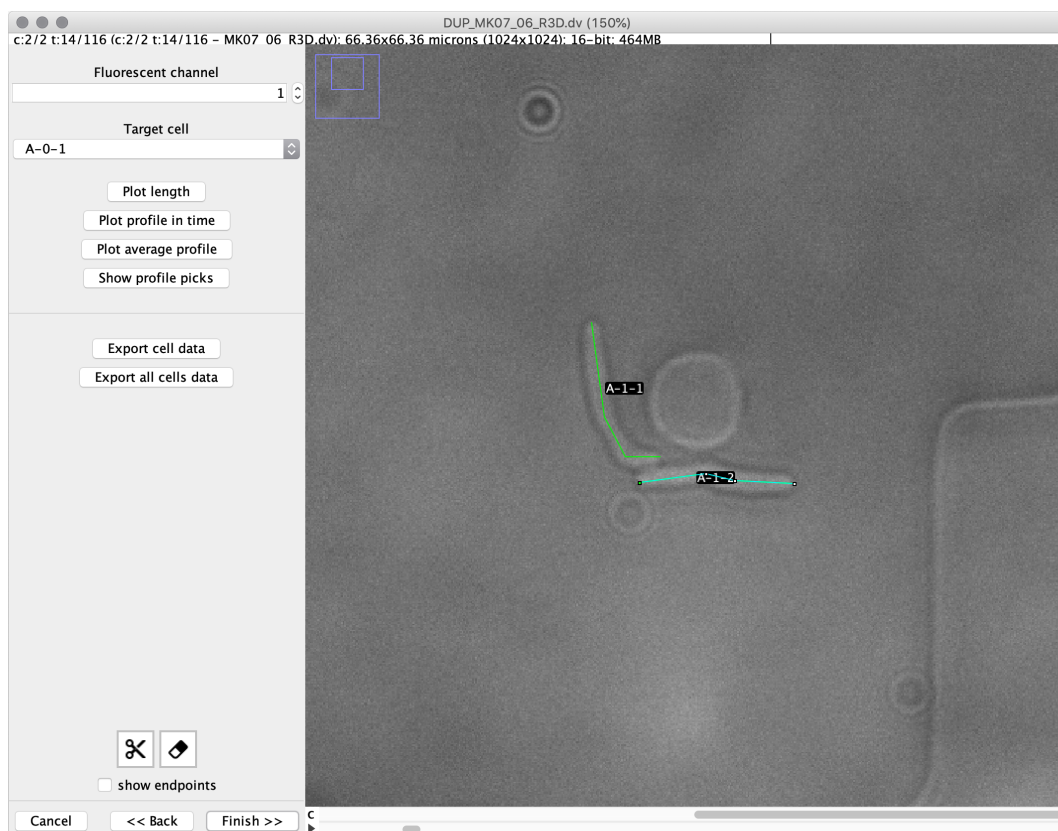
Rysunek 3.3: Drugi krok pluginu służący do wstępnej śledzenia komórek w czasie.

następnie skorzystać z dostępnych narzędzi i manualnie je dostosować (*duplicate*). Cofając się w historii do poprzednich klatek można poprawiać zaznaczenia. Można też powtórnie wyliczyć zaznaczenia na danym obrazie, jednak należy wtedy pamiętać, że zaznaczenia w kolejnych klatkach zostaną zresetowane.

3.3.3. Analiza danych

Ostatni krok umożliwia analizę danych wyliczonych na podstawie zaznaczonych komórek (rysunek 3.4). W celu zapewnienia poprawności danych należy najpierw wskazać numer kanału z fluorescencją. Kolejnym krokiem jest wybór komórki do analizy. Umieszczona niżej sekcja przycisków pozwala na wyświetlenie kilku rodzajów wykresów (rysunek 3.5):

- wykresu przedstawiającego zmianę długości w czasie
- stosu profili intensywności pod łamaną opisującą komórkę na kanale z fluorescencją (dla każdej klatki życia komórki)
- średniego profilu intensywności od łamaną opisującą komórkę na kanale z fluorescencją (oś X każdego profilu jest przed uśrednieniem normalizowana do przedziału $[0, 1]$)
- umiejscowienia „ognisk” na kanale z fluorescencją w czasie.



Rysunek 3.4: Trzeci krok pluginu służący do analizy zaznaczonych komórek.

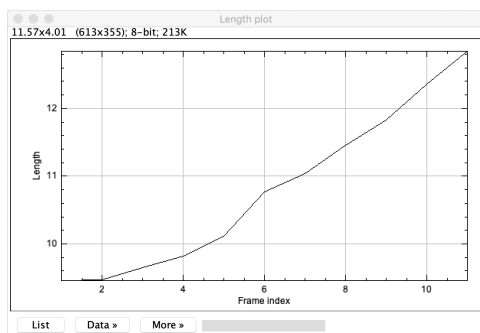
W celu analizy profili i długości komórek poza programem można, za pomocą kolejnych przycisków, wyeksportować dane pojedynczej lub wszystkich komórek do formatu CSV (ang. comma-separated values).

3.3.4. Manualna modyfikacja komórek

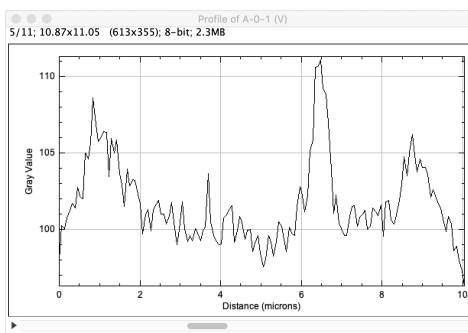
W celu manualnej modyfikacji komórki należy wybrać ją w oknie **ROI Manager**, które otwierane jest automatycznie podczas pracy z komórkami (rysunek 3.6). Komórka na obrazie będzie wyświetlona w postaci standardowego zaznaczenia wykonanego narzędziem do zaznaczania linii łamanej (ang. Segmented Line Selection Tool)[7]. Przesuwanie, usuwanie lub dodawanie wierzchołków takiego zaznaczenia zapisywane jest także dla komórki. Komórki usunięte z listy okna **ROI Manager** zostaną usunięte również z kolekcji wraz z zaznaczeniami w kolejnych klatkach, jak i całym potomstwem. Kolekcja zaznaczeń dla poprzednich klatek pozostanie bez zmian.

Dodatkowo okno pluginu wyposażone jest w dwa dodatkowe narzędzia mające ułatwić manualną edycję komórek. Narzędzia można włączyć za pomocą przycisków widocznych w lewym dolnym rogu okna pluginu.

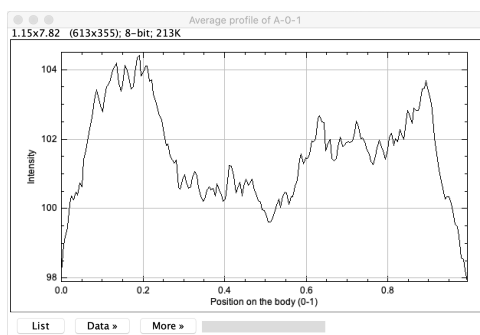
Tryb rozcinania komórek można aktywować za pomocą przycisku z ikoną w



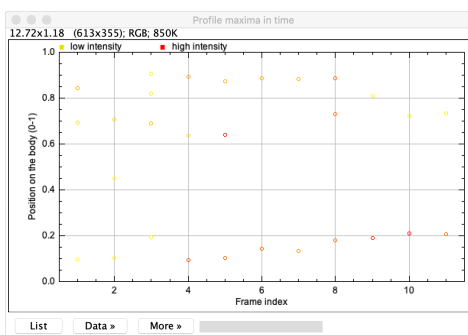
(a) Wykresu przedstawiający zmianę długości w czasie.



(b) Stos profili intensywności pod łamaną opisującą komórkę na kanale z fluorescencją.

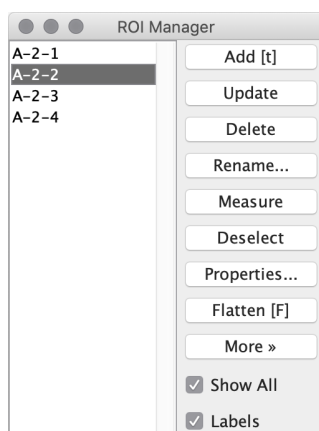


(c) Średni profil intensywności od łamaną opisującą komórkę na kanale z fluorescencją.



(d) Umiejscowienie „ognisk” na kanale z fluorescencją w czasie.

Rysunek 3.5: Różne rodzaje wykresów.



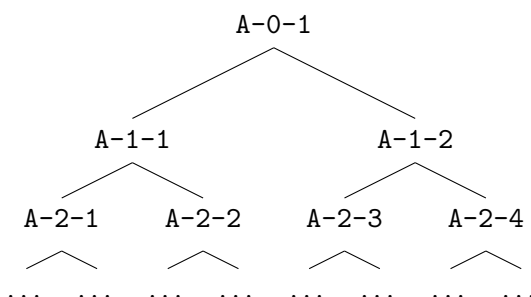
Rysunek 3.6: Okno ROI Manager pozwalające na wybór komórki.

kształcie nożyczek (✂). Służy on do ręcznego oznaczania punktów podziału komórek. Po jego uruchomieniu użytkownik może „przeciąć” komórkę rysując symboliczną linię na obrazie. Przecięta komórka zostanie podzielona na dwie komórki, które zostaną dodane do kolekcji jako „dzieci” komórki sprzed podziału (jeśli występuje ona w poprzedniej klatce). Ze względu na cykl życia opisywanych komórek, nie jest możliwe doprowadzenie do stanu w którym jedna komórka dzieli się na więcej niż dwie części. Należy pamiętać, że rozcięcie komórki, która została oznaczona w późniejszych klatkach, wiąże się z utratą jej zaznaczeń w tych klatkach oraz usunięciem jej potomstwa.

Drugim dodatkowym narzędziem jest tryb skracania końcówek, uruchamiany za pomocą przycisku z ikoną w kształcie gumki do mazania (◆). Działa on podobnie do narzędzia „gumki” obecnego w popularnych programach graficznych. Za jego pomocą nie można podzielić komórki, ale można ją skrócić lub całkowicie usunąć. Jeśli cała komórka zostanie „wymazana” to, analogicznie jak przy usuwaniu komórki z poziomu okna *ROI Manager*, usunięte zostaną również zaznaczenia w kolejnych klatkach oraz jej potomstwo.

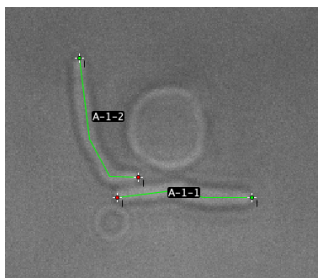
3.3.5. Nazewnictwo komórek oraz ich orientacja

Aby ułatwić identyfikację komórek, przyjąłem stałą konwencję ich nazewnictwa. Nazwa składa się z trzech członów: identyfikatora komórki źródłowej, indeksu pokolenia oraz indeksu w ramach pokolenia. Komórkom źródłowym (nie mającym rodzica) zostaje przydzielony unikalny identyfikator w postaci wielkiej litery alfabety. Indeks pokolenia jak i indeks w ramach pokolenia wynikają z natury kolekcji komórek. Każda komórka dzieli się w pewnym momencie na dwie nowe komórki (dzieci) nowego pokolenia. „Drzewo genealogiczne” komórek będzie w takim przypadku drzewem binarnym, a indeksy pokolenia i w ramach pokolenia będą odpowiednio odległością od korzenia i liczbą porządkową w ramach tego samego indeksu pokolenia. Przykładowe drzewo pokrewieństwa opisane nazwami komórek przedstawia rysunek 3.7.



Rysunek 3.7: Drzewo genealogiczne opisane zgodnie z konwencją nazewnictwa.

Dzięki tak skonstruowanej konwencji nazewnictwa można łatwo zidentyfikować pochodzenie komórki, a także stopień pokrewieństwa z innymi komórkami.



Rysunek 3.8: Komórki wraz ze znacznikami identyfikującymi końcówki.

W celu zapewnienia spójności analizowanych na końcu danych, ustaliłem również stałą orientację komórek. Aby móc rozróżnić końcówki komórek, wystarczy zaznaczyć opcję `show endpoints` w dolnej części okna pluginu. Kolorem czerwonym oznaczone zostaną końcówki powstałe w wyniku podziału komórki, a zielonym te które były wcześniej końcówkami komórki rodzica (rysunek 3.8). W przypadku komórki źródłowej końcówki oznaczone są losowo.

3.3.6. Importowanie i eksportowanie komórek

Manualne oznaczanie i poprawianie komórek może być czasochłonne i nie zawsze można zakończyć pracę w ciągu jednej sesji. Aby nie utracić efektów swojej pracy, użytkownik może wyeksportować oznaczone komórki za pomocą menu **Development** > **Export cells**. Stworzony w ten sposób plik XML jest wynikiem serializacji kolekcji komórek. Podczas kolejnej sesji, po otwarciu tego samego nagrania, zapisane komórki można wczytać korzystając z menu **Development** > **Import cells**.

3.4. Dokumentacja techniczna

3.4.1. Struktura projektu

Główna struktura plików projektu podzielona jest zgodnie z konwencją zaproponowaną przez zespół Apache Maven[8]. Główne foldery to:

- `src/main/java` – kod źródłowy aplikacji
- `src/main/resources` – zasoby aplikacji np. obrazki
- `src/test/java` – kod źródłowy testów automatycznych.

Kod źródłowy aplikacji podzielony jest na kilka folderów, ze względu na aspekt którego dotyczy:

- **cells** – kod zawierający logikę dotyczącą domeny pracy, m.in. kod pluginu, interfejsy poszczególnych kroków, główne algorytmy itp.; wewnątrz wydzielono jeszcze dodatkowo foldery:
 - **serialization** – kod dotyczący importu i eksportu komórek
 - **skeleton** – implementacja krótków detekcji i śledzenia komórek opisana w tej pracy (oparta na szkieletyzacji)
 - **measurements** – implementacja etapu analizy danych
- **graph** – implementacja grafu
- **gui** – implementacje generycznych okien i komponentów interfejsu
- **imagej** – klasy ułatwiające pracę i komunikację z programem–hostem ImageJ
- **util** – inne pomocnicze klasy
- **vendor** – publicznie dostępne wtyczki do programu ImageJ lub opakowania ułatwiające korzystanie z zależności zadeklarowanych w `pom.xml`.

Konwencja nazewnictwa

Nazwy interfejsów rozpoczynają się od wielkiej litery „I” np. `ICellsPluginStep`, natomiast nazwy klas abstrakcyjnych zaczynają się od prefiksu „Abstract” np. `AbstractCellCollection`.

3.4.2. Główne struktury danych

`AbstractCellCollection`

Ta abstrakcyjna klasa służy do opisywania drzewiastej struktury komórek. Dziedziczą po niej dwie klasy: `CellCollection` oraz `Cell`. Ta pierwsza służy do przechowywania kolekcji komórek reprezentowanych przez klasę `Cell`. Druga natomiast oprócz reprezentowania komórki, może zawierać także referencje do dwóch komórek potomnych, tworząc w ten sposób binarne „drzewo genealogiczne” komórki.

Klasa deklaruje m.in. metody dodawania i usuwania komórek z kolekcji, ale także metody do wydobywania wszystkich komórek „żyjących” w danej klatce nagrania.

Klasy dziedziczące po `AbstractCellCollection` muszą być serializowalne. Właśność ta jest wykorzystywana podczas importowania i eksportowania komórek do pliku XML.

Cell

Jest to klasa opisująca komórkę i cały jej cykl życia – od pojawienia się, do momentu podziału na dwie nowe komórki (może zawierać także referencje do tych komórek potomnych). Obiekt tej klasy zawiera informacje o każdej klatce w której występuje komórka w postaci referencji do instancji **AbstractCellFrame**. Pozwala też określić i odczytać poszczególne części nazwy komórki takie jak identyfikator komórki źródłowej, indeks pokolenia czy indeks w ramach pokolenia (3.3.5.).

Poza wyżej wymienionymi oraz dziedziczonymi po klasie **AbstractCellCollection** funkcjonalnościami, obiekt tej klasy zawiera również metody potrzebne do uzyskania reprezentacji komórki w postaci standardowego zaznaczenia w programie ImageJ.

AbstractCellFrame

Klasa ta deklaruje metody służące do uzyskania zaznaczenia pojedynczej komórki w konkretnej klatce nagrania, a także do modyfikacji tego zaznaczenia.

Na potrzeby tej pracy powstała jej implementacja **PolylineCellFrame**, która reprezentuje zaznaczenie komórki w postaci linii łamanej.

Graph, Edge, Vertex, Point

Klasy te składają się na implementację grafu, który zawiera dodatkowe informacje o punktach składających się na krawędzie i wierzchołki.

3.4.3. Architektura umożliwiająca dalszy rozwój

Główną klasą projektu jest **CellsPlugin**, która instancjonowana jest przez program ImageJ po otwarciu wtyczki z poziomu menu **Development > Cell Detector**. Podczas uruchomienia tworzona jest kopia aktualnie otwartego stosu obrazów, która wyświetlona zostaje w specjalnym oknie pluginu (jego opis znajduje się w rozdziale 3.3.), a także utworzona zostaje pusta kolekcja komórek.

Podczas inicjalizacji uruchomiony zostaje również nasłuch na różnego rodzaju zdarzenia programu-hosta ImageJ m.in. na:

- zmiany podglądu obrazu – umożliwia to wyświetlenie odpowiednich komórek dla aktualnie oglądanej klatki
- zmiany zaznaczeń na obrazie – dzięki temu zmiany mogą być zapisywane dla komórek opisywanych przez zmodyfikowane zaznaczenia
- zmianę aktywnego narzędzia – może mieć wpływ na zachowanie wybranego narzędzia specjalnego (3.3.4.)

- zmiany na liście okna **Roi Manager** – umożliwia to usuwanie komórek z poziomu tego okna.

Każdy z trzech etapów pluginu (wstępna detekcja, śledzenie komórek i analiza danych) jest obiektem klasy implementującej interfejs `ICellsPluginStep`. Interfejs ten deklaruje zaledwie trzy publiczne metody:

- `JComponent init(ImagePlus imp, CellCollection cells)`

Metoda ta wywoływana jest podczas inicjalizacji danego kroku np. przy starcie pluginu (dla kroku wstępnej detekcji) lub przy wciśnięciu przycisku **next** (dla kolejnego kroku). Argumentami są obraz źródłowy (nagranie) oraz referencje do kolekcji komórek. Metoda odpowiedzialna jest również za stworzenie i zwrócenie komponent interfejsu danego kroku, który zostanie automatycznie umieszczony w odpowiednim miejscu okna wtyczki.

- `void imageUpdated()`

Ta metoda służy do nasłuchiwania na zmiany obrazu i wyświetlanych komórek. Uruchamiana jest przez plugin po każdej aktualizacji podglądu obrazu i komórek na nim wyświetlonych.

- `void cleanup()`

Metoda ta wywoływana jest podczas zmiany aktualnego kroku lub przy zamknięciu pluginu.

Dzięki tak zaprojektowanemu systemowi każdy z etapów programu jest niezależny i może zostać z łatwością wymieniony na inną implementację tego interfejsu. Kod pluginu zawiera wszystkie główne struktury danych (lub ich abstrakcyjne wersje) i dzięki temu jest zupełnie niezależny od implementacji poszczególnych kroków – komunikuje się z nimi jedynie za pomocą opisanych wyżej interfejsów. Funkcjonalności niezależne od implementacji poszczególnych etapów takie jak wyświetlanie, usuwanie i ręczna modyfikacja komórek, a także ich eksportowanie i importowanie, bazują wyłącznie na metodach deklarowanych przez odpowiednie interfejsy lub klasy abstrakcyjne.

3.4.4. Wykorzystywane API ImageJ

ImageJ jest częścią większego ekosystemu SciJava. Projekty tworzone w ramach tego ekosystemu są wytwarzane jako otwarte oprogramowanie z publicznie dostępnym kodem. ImageJ jest nie tylko programem, ale rozszerzalną platformą zbudowaną na rozszerzalnej architekturze opartej o mechanizm pluginów. Stanowi to jeden z największych atutów tej platformy[9].

Plugin został napisany przy wykorzystaniu nowszego API ImageJ2, jednak korzysta również z warstwy zapewniającej kompatybilność z ImageJ 1.x. Spowodowane

jest to głównie brakiem dobrej dokumentacji istotnych funkcjonalności dla nowszej wersji API, dostępnej w trakcie implementacji projektu.

W moim programie zdecydowałem się użyć natywnych dla ImageJ narzędzi do tworzenia zaznaczeń (ROIs, ang. regions of interest), w celu wyświetlania i modyfikacji komórek. Dzięki temu nie musiałem tworzyć od nowa mechanizmów służących do wyświetlania wielu linii łamanych na obrazie, a także do ich modyfikacji. Również narzędzia do przycinania i skracania końcówek są tak naprawdę nakładką na standardowe narzędzia do zaznaczania linii i zaznaczania za pomocą pędzla.

3.4.5. Brakujące lub niespójne API ImageJ

W tak dużym otwartym oprogramowaniu tworzonym przez dużą społeczność nie sposób uniknąć niespójności czy obsłużyć wszystkie warunki brzegowe. Implementując niektóre funkcjonalności zdecydowałem się użyć standardowych narzędzi ImageJ, w celu nieco innym niż ich oryginalne przeznaczenie. Ze względu na to, potrzebowałem niskopoziomowego dostępu do różnych mechanizmów za tym stojących, aby wstrzyknąć tam własną warstwę pośredniczącą. Twórcy ImageJ, mając na uwadze potrzebę rozszerzalności, wyposażyli tę platformę w szereg narzędzi mających w tym pomóc. Mimo wszystko kilka elementów systemu okazało się działać nie do końca zgodnie z intuicją i wymagało nie do końca eleganckiego rozszerzenia.

Nasłuchiwanie na zdarzenia modyfikacji ROI

Chcąc rejestrować wydarzenia na temat modyfikacji wyświetlanych aktualnie zaznaczeń, należy zaimplementować interfejs `RoiListener`, a następnie tę implementację zarejestrować przekazując ją do statycznej metody `Roi::addRoiListener`. Słuchacz będzie otrzymywał zgłoszenia na temat różnych wydarzeń związanych z zaznaczeniami m.in. na temat ich utworzenia, przesunięcia, modyfikacji, rozszerzenia, ukończenia czy usunięcia. W moim programie nasłuchiwanie na te wydarzenia odgrywa kluczową rolę. Zdiagnozowałem kilka miejsc w których system notyfikacji działa, moim zdaniem, niezgodnie z intuicją:

- Tworzenie zaznaczenia w kształcie linii polega na oznaczenie kursorem myszy punktu początkowego, wciśnięciu głównego klawisza myszy, przeciągnięciu kursora do miejsca docelowego, a następnie zwolnieniu klawisza myszy. Konstruując zaznaczenie w kształcie linii, słuchacz dostaje informacje na temat wydarzenia związanego z modyfikacją, natomiast po zwolnieniu przycisku myszy nie jest zgłaszane żadne wydarzenie. Intuicja podpowiada mi, że powinno zostać zgłoszone wydarzenie związane z ukończeniem konstrukcji zaznaczenia.
- Analogiczny problem występuje w przypadku zaznaczania za pomocą pędzla. Tutaj proces zaznaczania podobny jest do „malowania” narzędziem pędzla

w popularnych programach graficznych. I w tym przypadku po zwolnieniu przycisku myszy nie jest zgłaszane żadne wydarzenie.

- Zaznaczenie w postaci linii łamanej może być modyfikowane na różne sposoby. Można przeciągać całą linię łamaną lub modyfikować jej wierzchołki poprzez ich przeciąganie, usuwanie lub dodawanie. W przypadku przesunięcia całego zaznaczenia jak i w przypadku zmiany pozycji poszczególnych wierzchołków zgłaszane są poprawne wydarzenia (odpowiednio o przesunięciu i modyfikacji). Jednak w przypadku usuwania i dodawania wierzchołków żadne wydarzenie nie jest zgłaszane.

W celu rozwiązania pierwszych dwóch problemów postanowiłem stworzyć własną warstwę aplikacji (`RoiObserver`) odpowiedzialną za obserwowanie zmian zaznaczeń. Nasłuchuje ona na zmiany zaznaczeń poprzez interfejs `RoiListener`, ale także nasłuchuje na zdarzenia związane z interakcją między kursorem myszy, a otwartymi oknami obrazów. Podczas konstrukcji problematycznych zaznaczeń zapisywane są ich referencje, aby następnie po zgłoszeniu wydarzenia o zwolnieniu przycisku myszy wyemitować brakujące wydarzenie.

Trzeci problem został rozwiązany poprzez utworzenie i korzystanie z nowej klasy rozszerzającej zaznaczenie w kształcie linii łamanej. Nadpisuje ona tylko jedną metodę `PolygonRoi::mouseDownInHandle`, która używana jest do tworzenia i usuwania dodatkowych wierzchołków (odbywa się to przez kliknięcie kursorem myszy w wierzchołek mając przy tym wciśnięty klawisz `alt` lub `control`). Nowa wersja metody wywołuje oryginalną, zachowując przy tym pierwotny sposób działania, a następnie zgłasza wydarzenie związane z modyfikacją.

Usuwanie ROI z poziomu okna ROI Manager

Najbardziej naturalnym sposobem na usunięcie komórki (np. w przypadku gdy została niepoprawnie podzielona), wydaje się zaznaczenie jej na liście okna `ROI Manager` i wciśnięcie przycisku `Delete` na klawiaturze lub z poziomu interfejsu. Bez specjalnej obsługi takiego zachowania, zaznaczenie komórki zniknęłoby, mimo że komórka wciąż pozostawałaby w kolekcji i po odświeżeniu podglądu znów pojawiłaby się na ekranie. Takie zachowanie byłoby nieintuicyjne z perspektywy użytkownika. Postanowiłem więc stworzyć odpowiednią obsługę takiej sytuacji. Okazało się jednak, że nie istnieje żadne narzędzie pozwalające otrzymać powiadomienie na temat usuniętego z poziomu tego okna zaznaczenia.

Problem ten rozwiązałem dostając się do komponentu wyświetlającego listę komórek. Pozwala on na nasłuchiwanie na zmianę zawartości, w szczególności usunięcie jednego lub wielu elementów. Po otrzymaniu takiego powiadomienia, stan listy porównywany jest z kolekcją komórek, które powinny aktualnie być wyświetlane – te których brakuje na liście usuwane są również z kolekcji. Rozwiązanie to nie

jest wyjątkowo stabilne, ponieważ dokonuje konwersji typów bazując na strukturze komponentów z jakich składa się okno `ROI Manager`.

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.
- [7] ImageJ User Guide, *Segmented Line Selection Tool*, <https://imagej.nih.gov/ij/docs/guide/146-19.html#sub:Segmented-Line-Selection>.
- [8] Apache Maven Project, *Introduction to the Standard Directory Layout*, <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>.
- [9] ImageJ User Guide, *Philosophy*, <https://imagej.net/Philosophy>.