Analiza wybranych algorytmów wykrywania składowych ruchu oka

inż. Paweł Kucia

Gliwice, 10 listopada 2019

załącznik nr 2 do zarz. nr 97/08/09

**Oświadczenie**

~~Wyrażam zgodę~~ / Nie wyrażam zgody\* na udostępnienie mojej pracy dyplomowej / ~~rozprawy doktorskiej\*~~

(podpis)

(poświadczenie wiarygodności podpisu przez Dziekanat)

Gliwice, 10 listopada 2019

**Oświadczenie promotora**

Oświadczam, że praca „Analiza wybranych algorytmów wykrywania składowych ruchu oka” spełnia wymagania formalne pracy dyplomowej magisterskiej

(podpis)

## Wstęp

Dziedzina okulografii *(z ang. eye-tracking)* na przestrzeni ostatnich kilkudziesięciu lat bardzo się rozpowszechniła. Jej zastosowanie możemy znaleźć w wielu aspektach życia oraz nauki, nawet nie zdając sobie z tego sprawy. Wraz z jej rozwojem umożliwiane jest coraz dokładniejsze prześledzenie ruchu gałek ocznych badanego podmiotu, co pozwala na dalszą analizę tych danych. Wynikiem takiej analizy może być na przykład mapa cieplna zmierzonego ruchu oka na banerze reklamowym, co w rezultacie może powodować, iż reklamodawcy otrzymują kolejne informacje dotyczące sposobu tworzenia reklam, by rozszerzyć zasięg osób, do których one trafiają.

Wraz z rozwojem metod analizy, zostało stworzonych wiele urządzeń służących do wykonywania dokładnych pomiarów ruchu, takich jak kamery, elektrody. W większości przypadków te dane są analizowane pod kątem wykrywania fiksacji lub sakad ruchu oka.

Celem poniższej pracy jest zaprezentowanie analizy wybranych algorytmów wykrywania fiksacji oraz sposobu ich implementacji. Analiza jest wykonywana pod kątem zużycia pamięciowego, czasu trwania algorytmów, ilości zmierzonych fiksacji. Badany również będzie wpływ parametrów zewnętrznych na poprawność i dokładność algorytmów. W pracy również zaprezentowano jeden przykład rozwiązania bazującego na coraz to popularniejszej technologii uczenia maszynowego. W tym algorytmie będzie można zbadać dokładność wyników końcowych względem tradycyjnych algorytmów.

Opis wykonanej pracy został podzielony na kilka rozdziałów. Drugi rozdział pracy przedstawia opis teoretyczny dziedziny okulografii oraz związanej z nią tematyki budowy ludzkiej gałki ocznej. W tym rozdziale scharakteryzowano również metody prezentacji danych a także opisano różne algorytmy wykrywania fiksacji, w tym te wybrane do dalszej analizy. Kolejny rozdział stanowi opis stworzonego projektu badawczego. Ten rozdział rozpoczęto od deskrypcji utworzonej aplikacji, wraz z opisem wykorzystanych narzędzi. Po tym zaprezentowano specyfikację zewnętrzną aplikacji, w tym jakie są parametry uruchomieniowe dla programu, przedstawiono format danych wejściowych oraz wyjściowych. Pod koniec tej sekcji przedstawiono metodę wyświetlania danych końcowych. Kolejnym podrozdziałem jest opis specyfikacji wewnętrznej programu. Rozpoczyna się on od zaprezentowania wymagań aplikacji, czyli koniecznych modułów oraz środowisk, celem poprawnego jej uruchomienia. Kolejnymi zademonstrowanymi funkcjonalnościami są: metodyka kalibracji danych wejściowych, w jaki sposób zaimplementowano obsługę bazy danych, opis przygotowania danych do analizy za pomocą algorytmów. Sekcje [3.4.6](#ssec:implementidt), [3.4.5](#ssec:implementivt), [3.4.7](#ssec:machinelearningalg) prezentują wdrożone algorytmy wykrywania fiksacji. Ostatnie 3 sekcje opisują w jaki sposób zaprezentowano wyniki, jak wykonano pomiary czasu oraz przedstawiono bibliotekę dotyczącą analizy wykorzystania pamięci w języku Python 3.  
Głównym zadaniem czwartego rozdziału jest przedstawienie wyników algorytmów oraz sposobu ich analizy. Pierwsze dwa podrozdziały opisują parametry maszyny, na której przeprowadzano badania oraz parametry związane ze stanowiskiem pomiarowym oraz danymi pomiarowymi. W sekcji [4.3.1](#ssec:regression) zaprezentowano sposoby implementacji regresji dla kalibracji danych wejściowych, jak również otrzymane skalibrowane punkty. Następna sekcja została poświęcona na sprawdzenie wpływu parametrów wejściowych algorytmów na wyniki końcowe. W kolejnych sekcjach pokazano porównanie czasów trwania algorytmów, analizę obciążenia pamięci przez algorytmy oraz wpływ metod odczytu danych na działanie aplikacji. Ostatni podrozdział demonstruje dokładność algorytmu uczenia maszynowego oraz porównanie punktów końcowych dla każdego algorytmu. Ostatni rozdział przeznaczono na podsumowanie oraz wnioski z przeprowadzonej analizy.

## Analiza dziedziny przedmiotowej

[chapter:ch2] W poniższym rozdziale zawarty został krótki opis oka, analiza teoretyczna dziedziny okulografii, jak również zaprezentowano typy algorytmów wykrywania fiksacji wykorzystanych do przeprowadzenia badań. Na końcu tego rozdziału przedstawiono metody wyświetlania danych. Ten rozdział oparto głównie o prace , , .

### Opis ludzkiej gałki ocznej

Zadaniem poniższej sekcji jest zaprezentowanie podstawowych terminów związanych z działaniem ludzkiego oka oraz jego budową. Krótka analiza oka została przygotowana zgodnie z pracą numer .

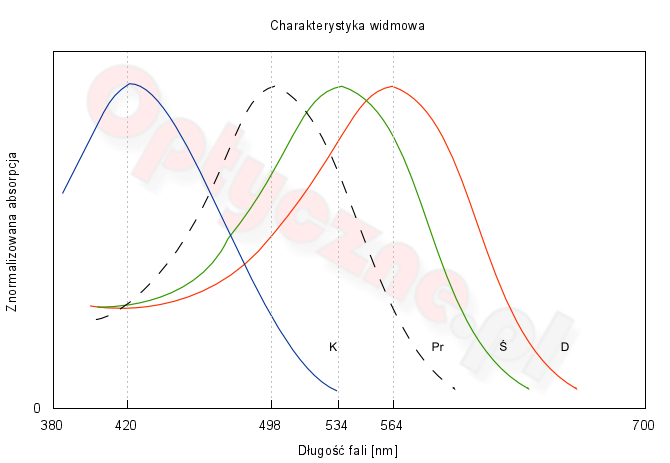
Jednym z najważniejszych organów w ludzkim ciele jest oko. Służy ono do dostarczania większości informacji dotyczących otoczenia w jakim się znajdujemy, oraz poszerzania wiedzy o świecie. Około 10% komórek mózgowych jest zaangażowanych przy interpretacji oraz analizie sygnałów dostarczanych z tego narządu.

Prosty schemat oka został zaprezentowany na rysunku [2.2](#fig:budowaoka).

Działanie oka polega na projekcji światła wpadającego do rogówki, najbardziej zewnętrznej część oka, poprzez źrenicę, która może być regulowana tęczówką w zależności od ilości promieni świetlnych. Następnie światło przechodzi przez soczewkę, która załamuje promienie świetlne, ciało szkliste, kończąc na wewnętrznej warstwie oka, nazywanej siatkówką. Składa się ona z dwóch typów fotoreceptorów:

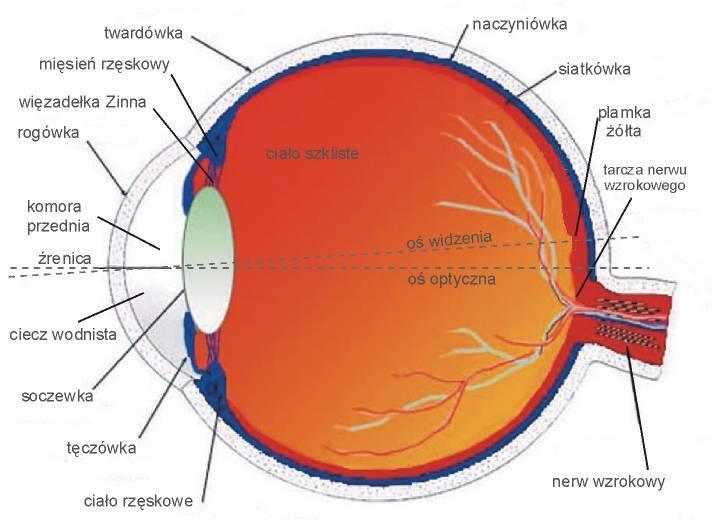
* 6 milionów czopków
* 100 milionów pręcików

Czopki odpowiadają za widzenie fotopowe, pręciki za widzenie skotopowe. Widzenie fotopowe pozwala na rozpoznawanie kolorów (czerwony, zielony, niebieski) przy dobrym oświetleniu, które nie może być zbyt intensywne, ponieważ czopki mogą ulec przesyceniu. Widzenie skotopowe to możliwość obserwacji czarno-białego obrazu przy słabszym oświetleniu, jednak obraz nie jest tak dokładny jak przy widzeniu fotopowym. Przykład zakresu absorbcji fal świetlnych można zaobserwować na rysunku [2.1](#fig:czopki).



Względna absorbcja światła przez czopki (K, Ś, D) oraz pręciki (Pr).  
(źródło: <https://bit.ly/2WmM9qn> [dostęp 26.10.2019])

Na siatkówce obraz, który powstał jest obrócony względem rzeczywistego. Nerw wzrokowy transmituje dalej ten obraz jako sygnał nerwowy do ośrodków wzrokowych kory mózgowej. Ważnymi elementami przy poprawnie funkcjonującym oku są również ciało rzęskowe, posiadające promieniście ułożone fałdy, wydzielające wodnistą ciecz, odpowiadającą za sztywność gałki ocznej, jak również mięsień rzęskowy, który zmienia krzywiznę soczewki, co powoduje modyfikację jej ogniskowej, przez co występuje zjawisko akomodacji oka.



Uproszczony schemat gałki ocznej.  
(źródło: <https://eszkola.pl/fizyka/schemat-budowy-oka-i-wady-wzroku-4225.html> [dostęp 01.10.2019])

Akomodacja oka to zjawisko pozwalające oku na dostosowanie się do oglądania przedmiotów znajdujących się na różnych odległościach. Jak wspomniano wcześniej, fotoreceptory znajdujące się w oku pozwalają nam rozpoznać obraz ze światła wpadającego do oka.

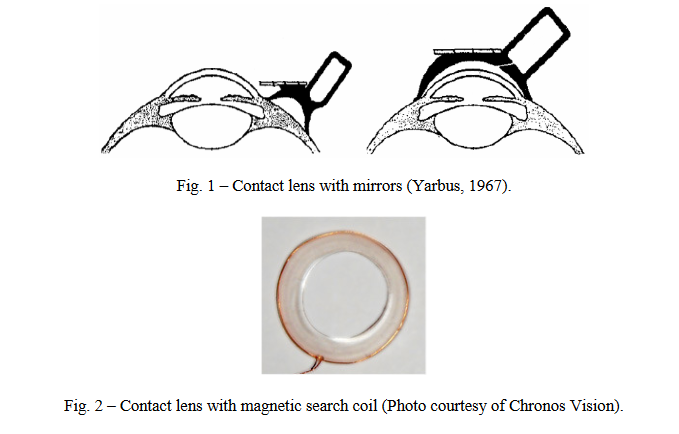
Najważniejszym dla nas elementem siatkówki jest plamka żółta, posiadająca największą rozdzielczość otrzymywanego obrazu, ze względu na najwyższe stężenie czopków w ludzkim oku. Umożliwia ona ludziom np. czytanie tekstu. Im dalej od plamki żółtej, tym zwiększa się ilość pręcików, a przez to zmniejsza się dokładność obrazu.

### Metody wykrywania ruchu gałek ocznych

Celem poniższej sekcji jest zademonstrowanie metod wykrywania ruchu gałek ocznych. Jednym z głównych kryteriów dla tych mechanizmów jest ich inwazyjność. Może ona powodować dyskomfort u badanego podmiotu, co w rezultacie może dać przekłamane wyniki, przez to, że użytkownik może zachowywać się mniej naturalnie. Wiele badań również wykazuje to, iż preferowane jest, żeby badano jak największą liczbę osób z małą znajomością tematyki śledzenia ruchu oka. Jest to spowodowane tym, iż badany może wykonywać inne ruchy niż te które wykonuje naturalnie. Ta sekcja bazuje na danych znalezionych w pracy . Poniżej przedstawiono trzy główne metody mierzenia ruchu oka.

#### Metody bazujące na soczewkach kontaktowych

Ta kategoria sposobów mierzenia ruchu oka wymaga od badanego założenia na gałki oczne soczewek kontaktowych, posiadających lustra, bądź cewki z kablem, które okrążają soczewkę kontaktową. Te cewki są połączone do zewnętrznego systemu cewek magnetycznych. Pomiar odbywa się za pomocą wykrywania zmian sił pola magnetycznego. Pomimo większego rozwoju technologicznego, ze względu na inwazyjność całego rozwiązania, jak również konieczność unieruchomienia głowy w trakcie przeprowadzenia pomiarów, zaniechano korzystania z tego typu rozwiązań. Przykład takich metod pokazano na rysunku [2.3](#fig:soczewki).



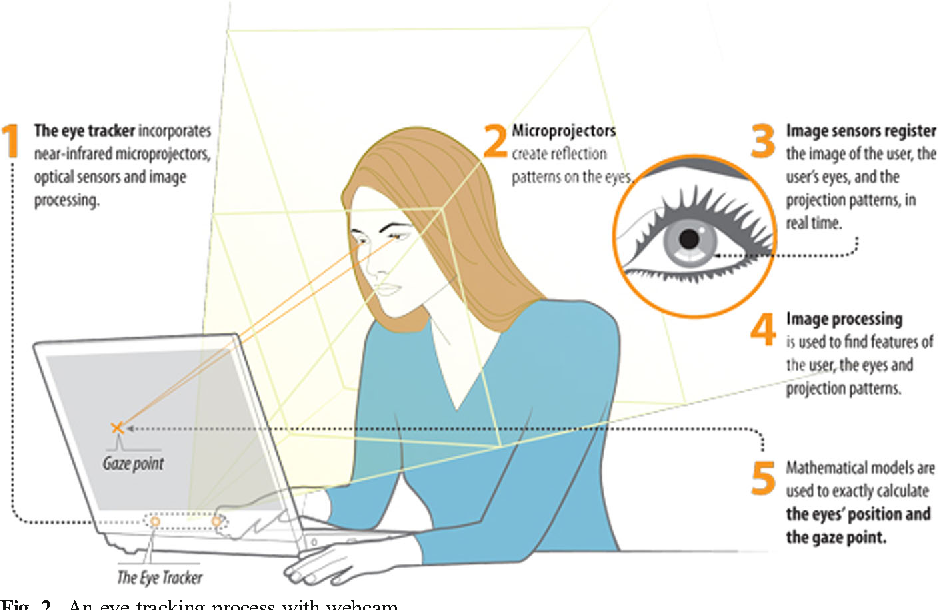
Przykład rozwiązań bazujących na soczewkach kontaktowych.

#### Elektrookulografia

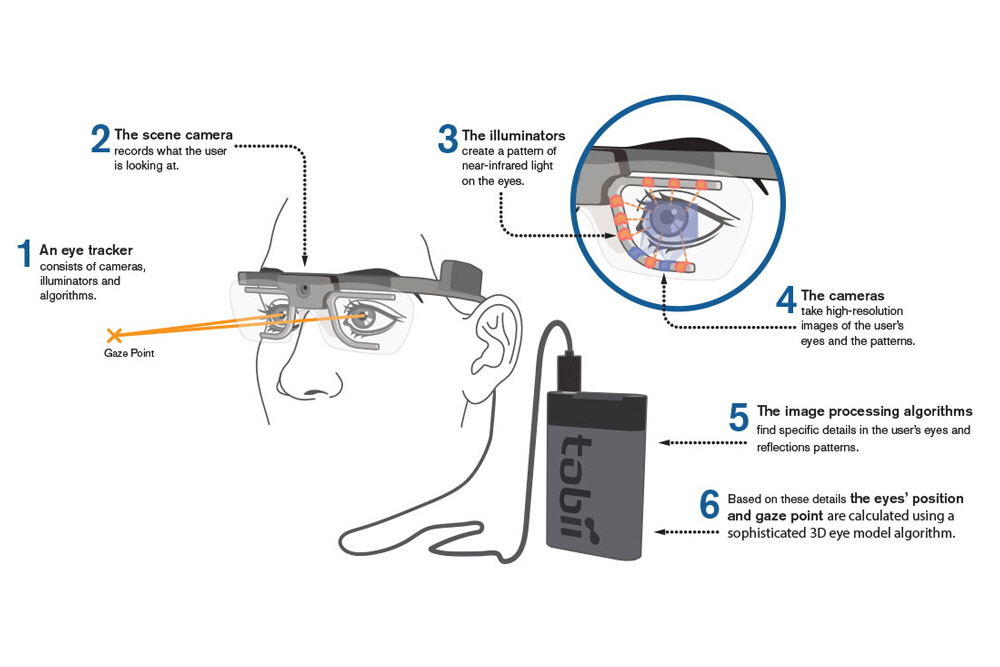
Elektrookulografia jest typem rozwiązań bazującym na różnicy potencjałów pomiędzy elektrodami przymocowanymi blisko oczu. Ze względu na dużą ilość połączeń nerwowych w gałce ocznej, można wykryć zmianę napięcia w samej gałce, gdzie wynosi ona około 1mV, jak również zauważalna jest zmiana pól elektrycznych podczas wykonywania ruchu. Amplituda tych zmian zależy od pozycji oka. Podobnie do rozwiązania prezentowanego w sekcji [2.2.1](#ssec:lenses), to rozwiązanie straciło popularność na rzecz wideookulografii.

#### Kamery

Ostatnia, trzecia metoda pomiarów wykorzystuje kamery do rejestracji kolejnych pozycji ruchu. Ten sposób pozyskiwania danych wykorzystuje dwa typy obrazów: obraz z naturalnego światła oraz z podczerwieni. Obraz z naturalnego światła jest nazywany *podejściem pasywnym*, które pobiera dane z odbitego światła z oka, wynikiem tego obrazu jest zarys ruchu soczewki. Te rozwiązania zaprezentowano poglądowo na rysunkach [2.4](#fig:camerassub1) i [2.5](#fig:camerassub2). Jak można zaobserwować z rysunków, te metody nie wymagaja specjalnego montażu na użytkowniku. Z tego względu one znalazły najszersze zastosowanie w dziedzinie pomiaru ruchu gałek ocznych. Wadą tego rozwiązania jest zależność od źródła światła, gdyż pomiary przeprowadzane w złym świetle mogą powodować przekłamania, a w skrajnych przypadkach brak możliwości odczytu danych. Wykorzystanie odczytu światła podczerwieni eliminuje ten problem. Kolejną zaletą użycia podczerwieni jest zmiana wyniku, z zarysu ruchu soczewki na zarys ruchu źrenicy, poprzez wykorzystanie zjawiska odbicia światła z ekranu. W wypadku spojrzenia na element, odbicie jest rejestrowane jako biały punkt, w przeciwnym wypadku jako czarny punkt.



Stanowisko z kamerą



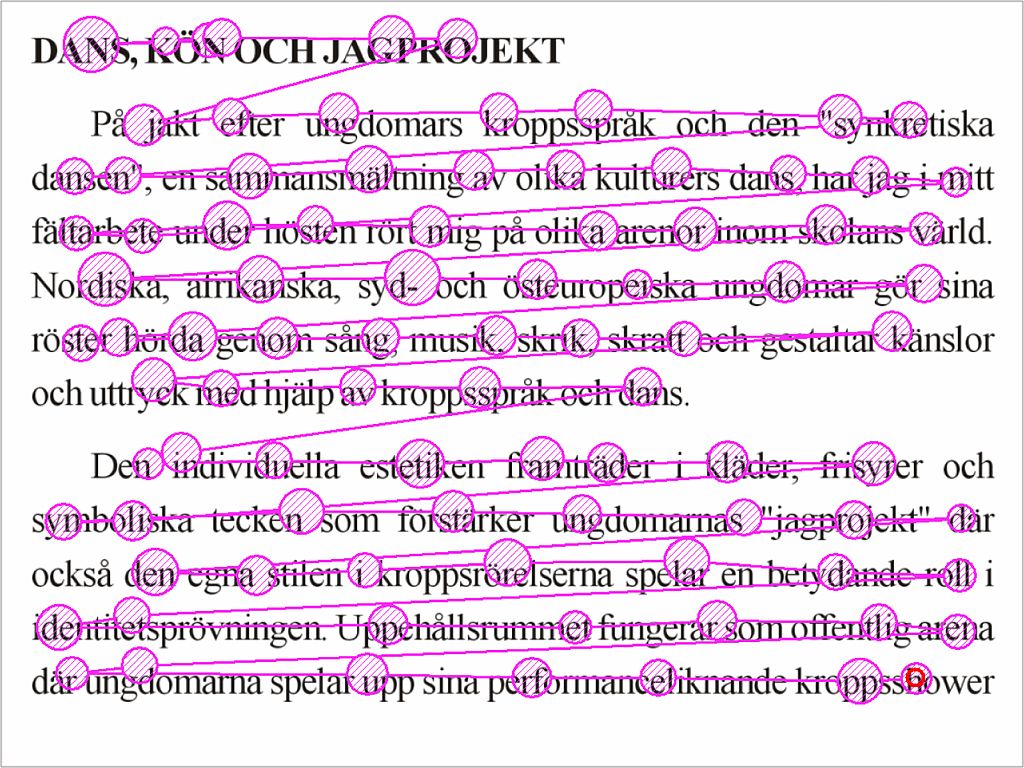
Kamera w okularach

### Eye-tracking

W tej sekcji przybliżono technologię okulografii, oraz jej zastosowania.

Jak wspomniano we wstępie do pracy, rozwój omawianej dziedziny w przeciągu ostatnich kilkudziesięciu lat pozwala nam przeanalizować sposób w jakim operują ludzkie procesy obserwacji oraz sposobu rozpoznawania obrazu. Analiza tych procesów umożliwia badającym na wykorzystywanie wyników badań w zastosowaniach komercyjnych, np. badanie sposobu patrzenia na jezdnię, oraz deskę rozdzielczą w trakcie poruszania się pojazdem samochodowym , analiza psychologiczna czy też w przygotowaniach do tworzenia reklam .

W celu przetworzenia danych z urządzenia pomiarowego, których przykłady zaprezentowano w podrozdziale [2.2](#sec:movement), do dalszej analizy stosuje się typowo dwie wartości: fiksacje, czyli miejsca, na których badana osoba się skupiła, oraz sakady, szybkie ruchy pomiędzy fiksacjami. Podział ten został odkryty w XIX wieku we Francji za pomocą obserwacji fizycznych, ponieważ zauważono, że ludzkie oko podczas czytania nie porusza się płynnie, a wykonuje "skoki" pomiędzy obszarami tekstu. Przykład takiego ruchu oraz punktów skupienia można zaobserwować na rysunku [2.6](#fig:fiksacje).



Przykład fiskacji i sakad na tle czytanego tekstu.  
(źródło: <https://upload.wikimedia.org/wikipedia/commons/e/ef/Reading_Fixations_Saccades.jpg> [dostęp 10.09.2019])

Analiza danych odbywa się między innymi poprzez translacje danych ruchu oka z urządzenia wejściowego na fiksacje, zezwalając na wykonanie podziału danych. Daje to możliwość pozbycia się mniej interesujących danych z próbki, takich jak sakad, pomniejszych ruchów oka, które mogły nastąpić przy niedokładnym pomiarze, czy przy mikroskopijnym ruchu oka. Cytując *"w większości badań naukowych, dane z sakad nie stanowią aż takiej przydatności"*. Zezwala to nam na zmniejszenie rozmiaru danych, poprzez zbijanie rzeczywistych fiksacji do jednego, większego punktu danych. Najczęściej otrzymane wartości są wykorzystywane do metryk pomiaru typu czas fiksacji, prędkości i amplitudy sakad, jak również miary pomiędzy fiksacjami a sakadami.

Wyniki algorytmów wykrywania fiksacji są wynikami typowo statystycznymi, tzn. możemy określić ile wystąpiło fiksacji, a przez to ile elementów jest sakadami, ale dalsza analiza danych należy do badającego. Stwarza to problem interpretacji danych, zgodnie z *"jednym ze sposobów walidacji tych algorytmów jest porównanie wynikowych fiksacji z wrażeniami wizualnymi obserwującego"*.

### Algorytmy wykrywania fiksacji

Wyróżniamy trzy typy algorytmów wykrywania fiksacji ze względu na badany obszar:

* prędkościowe
* dyspersyjne
* powierzchniowe

Algorytmem prędkościowym możemy nazwać algorytm, który analizuje punkty pod kątem różnicy prędkości pomiędzy nimi, biorąc pod uwagę, iż fiksacje posiadają niską prędkość pomiędzy swoimi punktami, a sakady wysoką. Algorytmy dyspersyjne bazują na odległościach pomiędzy punktami, zakładając, iż fiksacje posiadają małe odległości międzypunktowe. Algorytmy powierzchniowe to algorytmy, których zadaniem jest identyfikacja punktów w wybranych powierzchniach zainteresowania (AOI)[[1]](#footnote-45). Algorytmy tego rodzaju posiadają, w przeciwieństwie do innych algorytmów, możliwość identyfikacji zarówno nisko- jak i wysoko-poziomowej, przez to, iż parametrem algorytmów AOI może być fiksacja. Identyfikacja niskopoziomowa polega na pomiarze powierzchni wewnątrz jednej fiksacji, celem dokładniejszego podziału na fiksację i sakady.  
Można także podzielić algorytmy ze względu na ich charakterystykę czasową, wyróżniamy dwa główne rodzaje:

* czułe na czas trwania
* adaptujące się lokalnie

Ten podział został stworzony dlatego, iż fiksacje bardzo rzadko trwają mniej niż 100 ms, a regularny czas ich trwania potrafi wynosić od 200 ms do 400 ms. Implementacja adaptacji lokalnych umożliwia na dokładniejszy pomiar fiksacji, co znalazło zastosowanie w bardziej skomplikowanych algorytmach bazujących na międzypunktowych prędkościach ruchu oka i wyznaczonych obszarach zainteresowań

#### Wybrane algorytmy

Celem poniższego podrozdziału jest opis teoretyczny algorytmów wykrywania fiksacji zastosowanych w pracy. Opis teoretyczny wybranych algorytmów bazuje na pracy oraz .

##### Algorytm I-VT

Algorytm I-VT[[2]](#footnote-48) jest przykładem algorytmu z grupy prędkościowych. Jak wspomniano w podrozdziale [2.4](#sec:fixations) te algorytmy bazują na różnicach prędkości międzypunktowych. Przykładem tych różnic mogą być wartości mniej niż 150 stopni/sekundę dla fiksacji, a więcej niż 300-400 stopni/sekundę dla sakad. Ze względu na proste wymagania algorytmu, nie jest on skomplikowany w implementacji. Pseudokod algorytmu I-VT zaprezentowano w kodzie [[lst:ivtpseudocode]](#lst:ivtpseudocode).

def ivt:  
 Oblicz prędkości pomiędzy punktami dla wszystkich punktów w protokole  
 Określ punkty poniżej progu jako fiksacje, a powyżej jako sakady  
 Połącz wszystkie punkty fiksacji w grupy fiksacji, usuń wszystkie sakady  
 Zmapuj każdą grupę fiksacji do punktu znajdującego się w środku każdej grupy  
 return zmapowane punkty

Pierwszym krokiem algorytmu I-VT jest obliczenie prędkości między każdym punktem w badanym obszarze. Prędkość ta jest mierzona jako odległość między obecnym punktem a następnym (lub poprzednim) punktem. Następnie każdy punkt jest klasyfikowany jako fiksacja lub sakada w zależności od spełnienia warunku progu, którym w tym wypadku jest prędkość. Zgodnie z zasadami tego typu algorytmów, wszystkie elementy ponad granicą zaliczamy jako sakady, a resztę jako fiksacje. Kolejnym krokiem jest pozbycie się danych niepotrzebnych - sakad, i połączenie pozostawionych danych w grupy fiksacji. Ostatnim krokiem algorytmu jest wyznaczenie środka masy każdej grupy fiksacji, co pozwala nam zdefiniować fiksację jako punkt.

Według specyfikacji, ten algorytm posiada jeden parametr wewnętrzny, próg prędkości.

##### Algorytm I-DT

def idt:  
 while istnieją punkty do zbadania:  
 Zainicjalizuj okno na pierwszych punktach, celem pokrycia progu czasowego.  
 if dyspersja punktów w oknie >= próg:  
 Dodaj dodatkowe punkty do okna aż dyspersja > próg  
 Zanotuj fiksacje jako centroid punktów w oknie  
 Usuń punkty w oknie z listy punktów  
 else:  
 Usuń pierwszy punkt z listy punktów  
   
 return fiksacje

Następnym badanym algorytmem jest algorytm uwzględniający rozproszenie punktów pomiaru ruchu okularach I-DT[[3]](#footnote-52). W przeciwieństwie do wcześniej omówionego algorytmu I-VT, algorytmu I-DT wykorzystuje fakt, iż fiksacje ze względu na swoją małą prędkość mają tendencje do grupowania się. Algorytm I-DT identyfikuje te fiksacje za pomocą okien (grup punktów) o określonej dyspersji (*maksymalna separacja*). Pseudokod algorytmu można znaleźć w kodzie [[lst:idtpseudocode]](#lst:idtpseudocode).

Algorytm I-DT wykorzystuje ruchome okno, do którego należą punkty będące potencjalnymi fiksacjami. Zgodnie z powyższym pseudokodem, inicjalizacja okna następuje na podanym przez użytkownika progu czasowym. Następnie sprawdzana jest dyspersja pomiędzy punktami, którą można obliczyć za pomocą wzoru: . Jest to wzór dla płaszczyzny dwuwymiarowej, jednak można zastosować inne wzory w wypadku użycia innych płaszczyzn przy wykonywaniu pomiaru. W wypadku gdy obliczona dyspersja przekroczyła zadany próg, nie znaleziono fiksacji, okno zostaje przesunięte na kolejny punkt. W przeciwnym wypadku okno należy zanotować jako fiksację. W tym oknie należy dodawać elementy aż obliczona dyspersja przekroczy próg dyspersji. Algorytm się kończy w momencie gdy wszystkie punkty zostaną przeanalizowane.

Algorytm I-DT wymaga podania dwóch parametrów wejściowych: progu dyspersji oraz progu czasowego. Ponieważ fiksacje zazwyczaj trwają mniej niż 100 ms, można określić próg czasowy na poziomie 150-200 ms. W wypadku znajomości kąta pomiędzy okiem a ekranem, możemy określić próg dyspersji w granicy do do znanej wartości. W innym wypadku należy określić przybliżoną wartość na podstawie eksploracji danych pomiarowych.

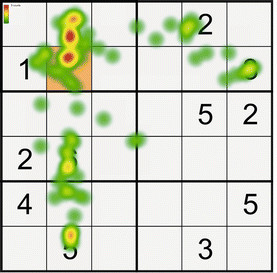
##### Uczenie maszynowe

W przeciągu ostatniej dekady można zauważyć znaczący wzrost zainteresowania technologiami powiązanymi z zagadnieniem sztucznej inteligencji, czy też uczenia maszynowego. Dzięki temu, iż dane wydobyte przez urządzenie śledzące ruch oka można łatwo sparametryzować, czy to przez podanie prędkości międzypunktowych, czy przez obliczanie dyspersji pomiędzy każdą parą punktów, również w dziedzinie eye-trackingu można zaobserwować wzrost wykorzystania technologii ML[[4]](#footnote-54). Dla okulografii głównym wykorzystaniem tego typu rozwiązań jest możliwość automatycznej analizy danych, bez konieczności podawania parametrów wejściowych, takich jak w tradycyjnych algorytmach zaprezentowanych we wcześniejszych akapitach. Możliwość analizy danych bez parametrów eliminuje błędy powiązane z ich niepoprawnymi wartościami. Użycie tego sposobu wymaga od użytkownika podania już scharakteryzowanych danych, w wypadku wykrywania fiksacji jest to informacja, czy dany punkt należy do fiksacji, czy jest sakadą.

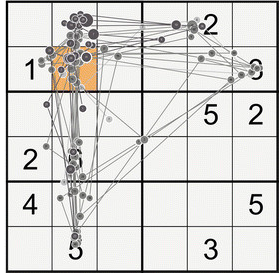
Opis działania algorytmu, jak również sposób analizy danych został zaprezentowany w sekcji [3.4.7](#ssec:machinelearningalg).

### Metody prezentacji danych

W oparciu o obliczone fiksacje, istnieje wiele rodzajów wizualnej analizy danych pochodzących z urządzenia mierzącego, na podstawie pracy , możemy wykonywać tą analizę poprzez mapy cieplne, których przykład pokazano na rysunku [2.7](#fig:heatmap). Mapy cieplne to obszary reprezentujące zwiększoną aktywność oka w danym miejscu, czy też punkty skupienia. Podobną metodą analizy danych są ścieżki skanowania, których zadaniem jest pokazanie linii, w jakich przebiega obserwacja na obrazie. Rysunek [2.8](#fig:scanpaths) ukazuje przykład ścieżek skanowania wraz z numeracją, jak przebiegał ruch oka. Metody te posiadają jednak pewne wady, mapy cieplne ukazują nam tylko pewną reprezentacje skupienia obiektu, nie ukazując nam kolejności w jakiej obraz został przeglądany. Ścieżki skanowania, jak zaprezentowano na rysunku [2.8](#fig:scanpaths) pozwalają zaprezentować dokładny przebieg badanej próbki na rysunku, jednak w wypadku zbyt dużego nagromadzenia danych, taki rysunek może być nieczytelny.



Przykład mapy cieplnej



Przykład ścieżki skanowania

## Opis projektu badawczego

Wewnątrz tego rozdziału znajduje się krótki opis aplikacji służącej do przeprowadzenia analizy punktów, deskrypcja wykorzystanych narzędzi, jak również specyfikacje wewnętrzna i zewnętrzna aplikacji. W specyfikacji zewnętrznej znajdziemy opis parametrów uruchomieniowych programu, sposób formatowania danych wejściowych oraz wyjściowych oraz podgląd danych wyjściowych. Dla specyfikacji wewnętrznej przygotowano opis wymagań aplikacji, sposób kalibracji danych wejściowych. Zaprezentowano również fragmenty kodu przedstawiające obsługę silnika MongoDB dla platformy Python, metodę przygotowania danych do analizy za pomocą algorytmów I-VT oraz I-DT. Następnie przedstawiono implementację trzech algorytmów - I-VT, I-DT oraz algorytmu korzystającego z uczenia maszynowego. Ostatnie trzy sekcje prezentują metody odpowiadające za prezentację wyników, pomiar czasu oraz pomiar wykorzystania pamięci.

### Krótki opis aplikacji

Program, który został wykorzystany do analizy punktów pomiarowych został przygotowany jako aplikacja konsolowa. Tego typu aplikacja pozwala na implementację algorytmów, jak również różnych sposobów wykonywania pomiarów, bez obciążania maszyny o zbędne GUI[[5]](#footnote-62), który może powodować przekłamania względem analizy czasowej i pamięciowej. Aplikacja wykorzystuje bazę danych MongoDB, będącej bazą typu NoSQL. Zastosowano to rozwiązanie celem weryfikacji różnicy w łącznym czasie trwania aplikacji w stosunku do pobierania danych z pliku. W tabeli [[tab:nosqlschema]](#tab:nosqlschema) zeprezentowano schemat tabeli z elementami.

Schemat tabeli z elementami

|  |  |
| --- | --- |
| **Nazwa** | **Typ danych** |
| \_id | ObjectId *Unique PK* |
| Type | String |
| CoordX | Double |
| CoordY | Double |
| TimeStamp | String |

Jak można zaobserwować na schemacie, identyfikator tabeli jest kluczem głównym, unikalnym, generowanym automatycznie przez silnik bazodanowy. Następne pola odpowiadają wartościom zaprezentowanym przy analizie danych wejściowych opisanej w sekcji [3.3.2](#ssec:importdata).

W założeniu aplikacja działa w następujący sposób.

1. Podając odpowiednie parametry wejściowe uruchom aplikację.
2. Pobierz dane wejściowe z wybranego źródła.
3. Skalibruj te dane do formatu danych odpowiadających punktom ’SS’
4. Dla grupy punktów o typie ’R’ oddzielonej punktem posiadającym parametr ’SS’ uruchom wybrany algorytm
5. Zaprezentuj wynik algorytmu na rysunku dla użytkownika
6. Zapisz dane wyjściowe do plików

Dokładny opis działania każdego z powyższych punktów zaprezentowano w sekcji [3.4](#sec:internal).

### Wykorzystane narzędzia

Projekt został stworzony wykorzystując język programowania *Python* w wersji 3.7.3. Ten język został wybrany ze względu na jego czytelność oraz modułowość. Kolejnym czynnikiem przeważającym w wyborze, była prostota w implementacji algorytmów, ze względu na wykorzystanie gotowych funkcjonalności języka Python. Rozważanymi alternatywami był język *F#* oraz język *R*. Język F# wykorzystuje platformę .NET, co na pewno ułatwiło by rozwiązanie problemu stworzenia algorytmu wykorzystującego uczenie maszynowe zaprezentowane w sekcji [3.4.7](#ssec:machinelearningalg), ze względu na prostą metodę integracji go z modułem *ML.NET*[[6]](#footnote-64). Język R w budowie jest bardzo podobny do języka F#, ale jego kod źródłowy jest otwarty, w przeciwieństwie do F#[[7]](#footnote-65) jednak podstawowa znajomość języka Python zadecydowała o wykorzystaniu tej platformy. Wszystkie wymagane moduły oraz sposób użycia opisano w sekcji [3.4.1](#ssec:apprequirements).

Praca magisterska została napisana z wykorzystaniem narzędzia LaTeX. Głównym powodem wybrania tej technologii jest to, iż prace badawcze tworzy się łatwo w tej technologii, istnieje dużo rozszerzeń ułatwiających np. wklejanie fragmentów kodu, tworzenie pseudokodu oraz umieszczanie obrazków w dowolnym miejscu oraz formacie. Kolejną zaletą jest darmowość tego pakietu, w przeciwieństwie do pakietu Office firmy Microsoft. Istnieją również rozwiązania darmowe typu OpenOffice, LibreOffice, jednak nie posiadają one takich ułatwień dla prac naukowych, oraz pierwsze próby tworzenia sprawiały dużą trudność. Użyto edytora tekstu Visual Studio Code.

Jak wspomniano w sekcji [3.1](#sec:shortdesc) do stworzenia bazy danych wykorzystano silnik bazodanowy *MongoDB* typu NoSQL. NoSQL jest nierelacyjną bazą danych wykorzystującą typ **document** jako model danych. Oznacza to, iż dane są zapisywane w formacie BSON[[8]](#footnote-66), co umożliwia łatwy odczyt przez większość języków programowania. Ze względu na wykorzystany typ danych wejściowych, pozwalający na umieszczenie go w jednej tabeli bez relacji, postanowiono użyć tego rozwiązania. Język Python umożliwia w bardzo prosty sposób integrację z bazą danych MongoDB, jak zaprezentowano w sekcji [[ssec:db]](#ssec:db).

### Specyfikacja zewnętrzna aplikacji

Celem poniższego podrozdziału jest zaprezentowanie sposobu działania aplikacji, metody jej uruchomienia, opisanie formatu danych wejściowych oraz wyjściowych, jak również pokazano sposób wyświetlania wyników algorytmów.

#### Parametry wejściowe

Zadaniem tej sekcji jest przedstawienie dostępnych parametrów wejściowych dla aplikacji.  
Ze względu na to iż projekt został przygotowany jako aplikacja konsolowy, nie posiadająca interfejsu użytkownika, konieczne było zaprojektowanie odpowiedniego systemu wprowadzania danych do programu, wraz z wyborem odpowiedniego algorytmu. W celu ułatwienia uruchomienia aplikacji, został przygotowany skrypt uruchomieniowy napisany w języku Powershell. Znajduje się on w katalogu głównym aplikacji. Przykładowa treść tego skryptu została przedstawiona w kodzie [[lst:runapp]](#lst:runapp).

Set-Location $PSScriptRoot  
 python main.py -i '1\_01\_1311201811.cal' 'ML' -d  
 pause

Jak można zauważyć, druga linia kodu [[lst:runapp]](#lst:runapp) odpowiada za uruchomienie aplikacji. Pierwsza linia ustawia lokalizację środowiska okna w głównym folderze aplikacji, gdyż bez tego domyślną wartością w momecie uruchomienia skryptu była wartość *C:/Użytkownicy/nazwaużytkownika*. Ostatnia linia skryptu wymusza na użytkowniku wciśnięcie klawisza żeby zamknąć okno aplikacji.

Wszystkie wymagania, żeby uruchomić skrypt zostały podane w sekcji [3.4.1](#ssec:apprequirements).

Po nazwie pliku głównego wykorzystano przełącznik, który wymaga podania jednej z trzech wartości:

* *-i*, odpowiadającą za dalsze działanie aplikacji,
* *-h*, wyświetlającą pomoc z aplikacji,
* *-a*, pokazującą dostępne algorytmy

Następnym parametrem wejściowym jest nazwa pliku umieszczona w katalogu */data* w głównym folderze aplikacji. Czwartym parametrem jest wybór algorytmu, który ma za zadanie przebadanie podanego parametr wcześniej pliku pomiarowego. Wyróżnia się trzy wartości:

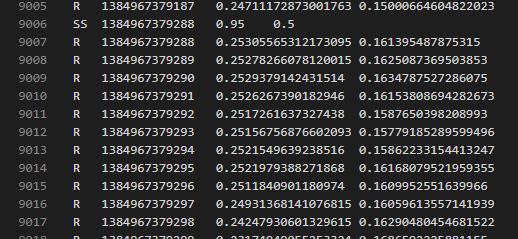
* *’ML’*, algorytm wykorzystujący uczenie maszynowe.
* *’I-DT’*, algorytm I-DT,
* *’I-VT’*, algorytm I-VT

Ostatni, *nieobowiązkowy* parametr odpowiada za wybór miejsca, z którego mają być wczytywane dane, ***-d*** wykonuje połączenie z bazą danych, a ***-f*** z pliku. W wypadku braku parametru, wykonywana jest ta druga akcja.

#### Format danych wejściowych

W tej sekcji zaprezentowano dane wejściowe, otrzymane w wyniku pomiarów z kamery.

Wszystkie dane wejściowe zostały umieszczone w folderze ***/data*** znajdującym się w katalogu głównym aplikacji. Przechowywane one są w formacie *.cal*, który można otworzyć za pomocą dowolnego edytora tekstowego. Zachowaniem przypomina on format *.csv*, z tą różnicą, iż zamiast znaków *;* lub *,* rozdzielających elementy w jednej wartości występuje znak specjalny **/t** odpowiadający jednemu wciśnięciu przycisku TAB na klawiaturze. Przykład takiego pliku wejściowego umieszczono na rysunku [3.1](#fig:plikwejsciowy).



Fragment pliku wejściowego

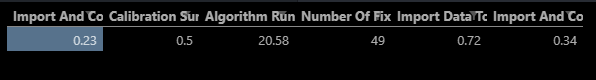
Pierwsza kolumna reprezentuje typ odczytanych danych, zawiera ona dwie wartości: **SS** oraz **R**. SS oznacza zmianę mierzonego punktu, a wszyskie elementy R oznaczają wykonany pomiar. Kolejna kolumna przechowuje czas wykonania pomiaru w formacie *UNIX* obliczanym w milisekundach. Jak można zauważyć, pomiar wykonywany jest z częstotliwością 1 ms, przez co odczytane wyniki powinny być dokładne. Trzeci oraz ostatni parametr to nieskalibrowane współrzędne punktów wejściowych, odpowiednio X i Y. Kalibrację danych opisano w sekcji [[ssec:calibration]](#ssec:calibration).

Podobny format danych można znaleźć w bazie danych, zgodnie z podrozdziałem [3.1](#sec:shortdesc), wraz z ich formatem.

#### Format danych wyjściowych

Ten podrozdział opisuje format danych wyjściowych oraz pliki w jakich one są zapisywane.

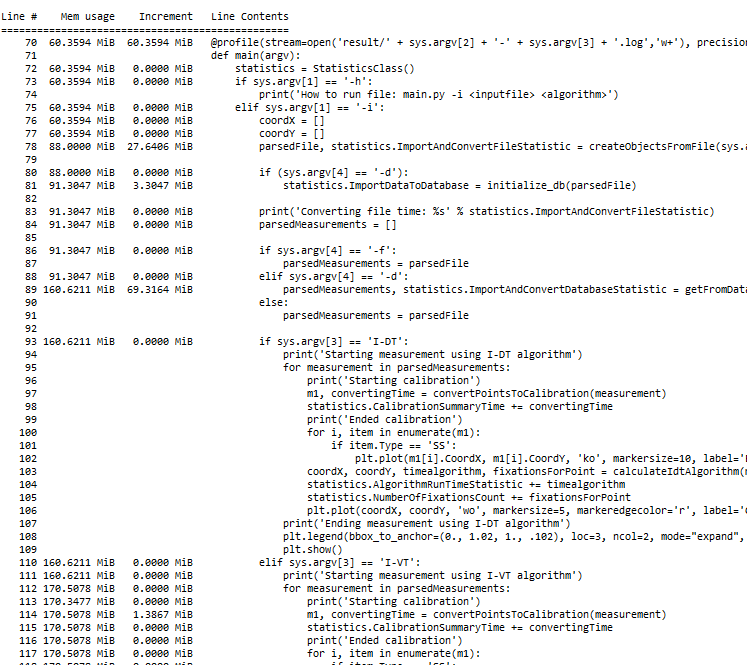
Rezultatem działania programu jest trójka plików, plik *.csv* zawierający obliczone dane, plik *.png* będący reprezentacją graficzną obliczonych fiksacji oraz plik *.log* przechowujący zrzut wykorzystania pamięci w czasie trwania aplikacji. Zapis danych odbywa się w momencie zakończenia obliczeń. Opis pliku graficznego został zaprezentowany w sekcji [3.3.4](#ssec:fixations).  
Plik *.csv* zostaje zapisany w folderze **/result**, posiadając w nazwie tytuł pliku wejściowego, dołączając do niego datę wykonania programu, np. 1\_01\_1311201811.cal23102019173857.csv. Plik ten został zbudowany w sposób zaprezentowany na rysunku [3.2](#fig:exportfile). Posiada on 6 parametrów, każdy odpowiadający pewnej obliczonej statystyce, bądź pomiarze zmierzonym w trakcie działania programu. Są to w kolejności od lewej do prawej:



Plik wyjściowy

1. ImportAndConvertFileStatistic
2. CalibrationSummaryTime
3. AlgorithmRunTimeStatistic
4. NumberOfFixationsCount
5. ImportDataToDatabase
6. ImportAndConvertDatabaseStatistic

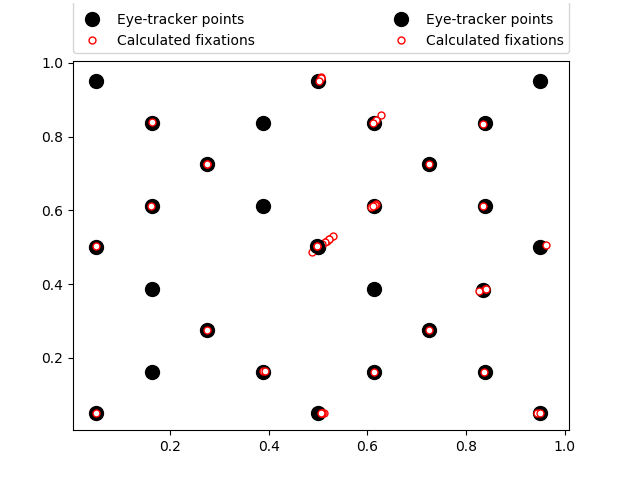
Wszystkie obliczone czasy podano w sekundach z dokładnością do 4 miejsc po przecinku. Pierwsza kolumna prezentuje czas trwania importu danych do pamięci oraz czas trwania konwersji plików do formatu czytelnego dla aplikacji, zgodnie z fragmentem [3.3.2](#ssec:importdata). Następna kolumna przedstawia łączny czas trwania kalibracji współrzędnych wejściowych do formy układu współrzędnych odpowiadającej mierzonym punktom ’SS’. Trzecia kolumna prezentuje czas trwania algorytmu wykrywania fiksacji, a czwarta ilość odnalezionych fiksacji. Piąta i szósta kolumna mogą być puste, gdyż wykazują czas trwania importu danych do bazy danych, oraz odpowiednik importu i konwersji z pierwszej kolumny dla bazy danych dla dalszej analizy.  
Celem obliczenia zużycia pamięci przez algorytm tworzony jest kolejny plik, tym razem z profilem pamięciowym aplikacji, czyli wykorzystaniem pamięci przez każdą linię utworzonego kodu. Fragment takiego pliku wynikowego zaprezentowano na rysunku [3.3](#fig:memoryfile).



Fragment pliku ze zrzutem pamięci

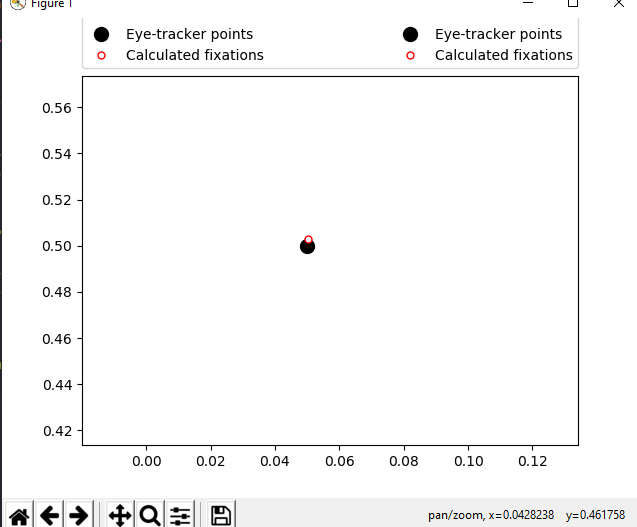
Pierwsza kolumna prezentuje linię w pliku, druga wykorzystanie pamięci, trzecia różnicę w pomiarach pomiędzy poprzednią linią a obecną a ostatnia treść linii. Czwarta kolumna pozwala w łatwy sposób zinterpretować, jak dużo pamięci pobiera algorytm obliczania fiksacji.

#### Prezentacja fiksacji



Przykład zaprezentowanych fiksacji

Rysunek [3.4](#fig:presentationfixation) pokazuje sposób wyświetlania danych po wykonaniu konkretnego algorytmu. Czarne punkty przedstawiają punkty zaprezentowane użytkownikowi w trakcie przeprowadzenia badań, czyli punkty ’SS’, a czerwone punkty prezentują wszystkie wykryte fiksacje. W celu zaprezentowania wszystkich elementów wykorzystano moduł matplotlib języka Python, który umożliwia takie rozwiązanie. Zezwala on również na przybliżanie otrzymanego wykresu, co pozwala na dokładniejszą analizę danych. Przykład ten zaprezentowano na rysunku [3.5](#fig:zoomedfixation).



Przybliżony podgląd fiksacji

W trakcie analizy danych, zaobserwowano iż w danych wejściowych istnieją dwa punkty ’SS’ o wartościach co powoduje zwiększenie ilości mierzonych fiksacji w okolicach tego punktu, jak również brak pomiaru dla wartości . Można to zaobserować na rysunku [3.4](#fig:presentationfixation).

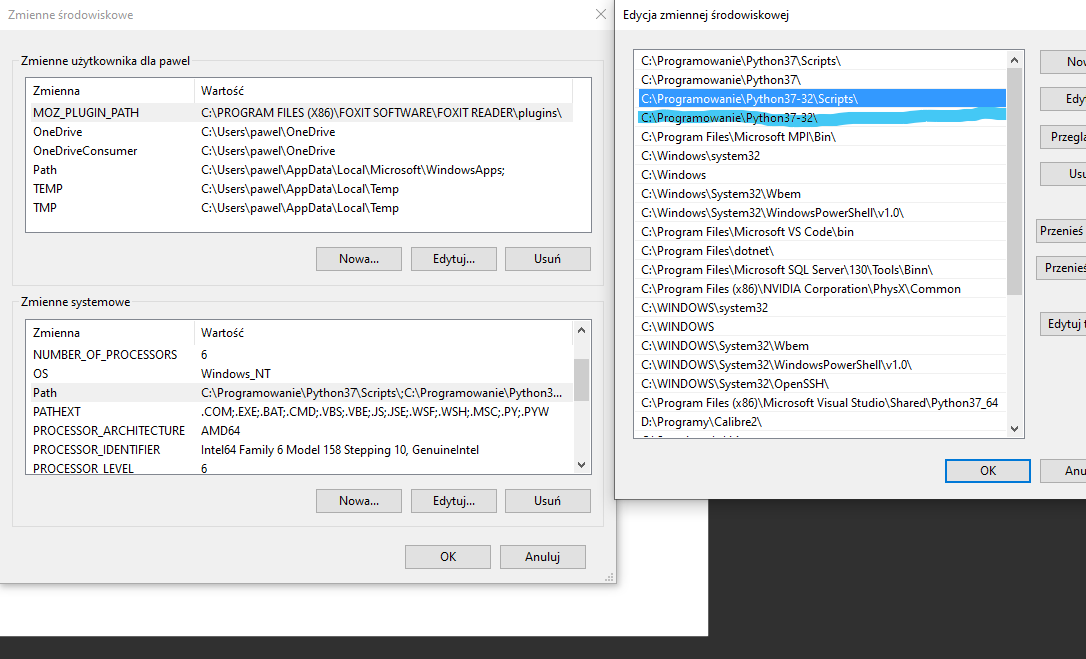
### Specyfikacja wewnętrzna aplikacji

W tym podrozdziale przedstawiono informacje dotyczące sposobu implementacji algorytmów opisanych w rozdziale [2.4.1](#ssec:algorithms). Przed tym, przedstawiono moduły języka Python wymagane do uruchomienia aplikacji, sposób ich wykorzystania, jak również opisano metodykę konwersji danych wejściowych na format czytelny dla algorytmów. Zaprezentowano także sposób połączenia z bazą danych MongoDB. Celem zwięzłości pokazanych fragmentów kodu, pominięto wyświetlanie inicjalizacji tablic, list oraz elementów nieistotnych dla działania algorytmów, typu pomiar czasu i pamięci. Ostatnie sekcje prezentują kod odpowiedzialny za pomiar czasu, profil pamięciowy oraz wyświetlanie danych.

#### Wymagania aplikacji

Żeby uruchomić skrypt *run.ps1* opisany w sekcji [3.3.1](#ssec:parameters) na platformie Windows, należy ustawić parametr ***ExecutionPolicy*** na wartość ***Unrestricted***. Można to wykonać za pomocą polecenia ***Set-ExecutionPolicy Unrestricted*** w oknie konsoli PowerShell.

Zgodnie z fragmentem [3.2](#sec:tools), aplikacja korzysta z języka *Python*. Celem uruchomienia programu należy dodać do zmiennych środowiskowych ścieżki do środowiska Python. Przykład takich elementów pokazano na rysunku [3.6](#fig:path).



Konfiguracja zmiennych środowiskowych

Projekt wymaga zainstalowania poniższych modułów języka Python, celem poprawnego działania konkretnych elementów aplikacji. Poniżej zaprezentowano te moduły, wraz z ich przeznaczeniem.

Wszystkie moduły można pobrać za pomocą pakietu **pip**, domyślnie instalowanego wraz z każdą instancją Pythona. Pierwszym modułem jest **NumPy**, które jest wiodącym pakietem przeznaczonym do bardziej skomplikowanych operacji na danych naukowych. W projekcie znalazł on zastosowanie m.in. w kalibracji danych wejściowych oraz przy danych dla modelu uczenia maszynowego. Pierwsze rozwiązanie przedstawiono w sekcj [[ssec:calibration]](#ssec:calibration), gdzie wykorzystano metodę najmniejszych kwadratów celem obliczenia regresji liniowej punktów. Kolejną biblioteką jest **Matplotlib.pyplot**, umożliwiająca wyświetlanie danych wyjściowych. Moduł zaprezentowano w podrozdziale [3.4.8](#ssec:datashow). Celem połączenia z bazą danych oraz prawidłową konwersją danych z tej bazy należało zainstalować moduły **pymongo** oraz **bson**. Zaprezentowano ich wykorzystanie w podsekcji [[ssec:db]](#ssec:db). Kolejną ważną funkcjonalnością jest moduł **memory\_profiler**, któego celem jest wykonywanie zrzutu pamięciowego funkcji w ciągu działania aplikacji. Ostatnim modułem, który wykorzystuje się w sekcji [3.4.7](#ssec:machinelearningalg) jest podmoduł dla **NumPy**, mianowicie **scikit-learn**. Zawiera on wszystkie potrzebne narzędzia do tworzenia modeli oraz wykonywania przewidywań za pomocą uczenia maszynowego.

Domyślnie zainstalowanymi pakietami, które również znalazły swoje zastosowanie w pracy są pakiety **time**, **os**, **sys**. Odpowiadają one za pomiar czasu, obsługę plików oraz komunikację z systemem.

Ostatnim wymaganym elementem jest posiadanie bazy **MongoDB**. W tym celu należy zainstalować wybraną wersję pakietu ze strony <https://www.mongodb.com/>. Podczas instalacji automatycznie tworzona jest sesja pod wybranym przez użytkownika portem. Domyślną wartością jest **27017**. Metoda połączenia z bazą danych dla tego projektu pokazana jest w kodzie [[lst:connectDB]](#lst:connectDB).

#### Kalibracja danych wejściowych

Ta sekcja ma za zadanie prezentację metodyki kalibracji danych wejściowych, jak również opis teoretyczny wykorzystanego zastosowania. Kod algorytmu znajduje się na listingu [[lst:calibration]](#lst:calibration).

Kalibracja danych ma za zadanie konwersję danych z czujnika ruchu do formatu danych na liniowym układzie współrzędnych odpowiadającym punktom. Jak można zauważyć na rysunku [3.1](#fig:plikwejsciowy), współrzędne X i Y, dla danych typu *’R’* w pliku wejściowym posiadają nietypowe wartości, co powoduje błędne działanie algorytmów wykrywania fiksacji. Rozwiązaniem problemu kalibracji jest zastosowanie metody najmniejszych kwadratów. Jak wykazano w sekcji [4.3.1](#ssec:regression), dla obecnych danych należało wykorzystać metodę liniową, a nie wielomianową.

Zadaniem liniowej metody najmniejszych kwadratów jest znalezienie współczynników funkcji liniowej, przez którą przebiegają wszystkie punkty. W tej metodzie zakładamy iż nasze równanie wynosi . Każda akcja wykonywana jest dwa razy, dla punktów osi X oraz Y.

Funkcja jest wywoływana osobno dla każdego punktu, ze względu na wstępną kalibrację przeprowadzoną przez urządzenie pomiarowe. [ssec:calibration]

import numpy as np  
def calibrate(xList, yList):  
 i = 1  
 ...  
 while i < len(xList):  
 X.append([xList[i]\*\*2,xList[i],yList[i]\*\*2,yList[i]])  
 Y.append([yList[i]\*\*2,yList[i],xList[i]\*\*2,xList[i]])  
 ssxArr.append(xList[0])  
 ssyArr.append(yList[0])  
 i+=1  
   
 x = np.linalg.lstsq(X,ssxArr,rcond=None)  
 y = np.linalg.lstsq(Y,ssyArr,rcond=None)  
  
 i = 1  
 ...  
 while i < len(xList):  
 retX.append(x[0][0]\*xList[i]\*\*2 + x[0][1]\*xList[i] + x[0][2]\*yList[i]\*\*2 + x[0][3]\*yList[i])  
 retY.append(y[0][0]\*yList[i]\*\*2 + y[0][1]\*yList[i] + y[0][2]\*xList[i]\*\*2 + y[0][3]\*xList[i])  
 i += 1  
 return retX, retY

W pierwszym kroku należy przygotować dane do dalszych obliczeń, co przedstawiono w liniach od 4 do 9. Tablice **ssxArr** oraz **ssyArr** są tablicami jednowymiarowymi o długości ilości pomiarów. Zawierają one wartość punktu ’SS’, który jest skalibrowaną poprawną wartością. Dla równania liniowego jest on wektorem . Tablice dwuwymiarowe **X** oraz **Y** odpowiadają macierzom współczynników . Po umieszczeniu wszystkich elementów następuje obliczenie wartości za pomocą dostępnej metody *np.linalg.lstsq*. Jako parametry przyjmuje ona wartości (A, b). Trzecim, opcjonalnym parametrem jest parametr odcięcia miejsc po przecinku dla małych wartości w macierzy . Zauważono, iż dla dostępnych danych należy ustawić go na *None*, żeby uzyskać dokładniejsze rezultaty.

Ostatnim krokiem algorytmu jest zapisanie obliczonych wartości punktów x oraz y. Następuje to za pomocą przyrównania ich do wartości z macierzy współczynników oraz zsumowania obliczonych wartości. Rezultatem algorytmu są skalibrowane punkty X oraz Y.

#### Obsługa bazy danych

W tym podrozdziale pokazano metody odpowiadające za obsługę bazy danych, w tym połączenie z nią oraz umieszczenie i wyciąganie danych. [ssec:db]

##### Połączenie z bazą danych

import pymongo  
 ...  
 LOCALHOST = "mongodb://localhost:27017/"  
 ...  
 myclient = pymongo.MongoClient(LOCALHOST)  
 mydb = myclient["mydatabase"]

Jak zaobserwowano w kodzie [[lst:connectDB]](#lst:connectDB) pakiet **PyMongo** umożliwia w bardzo prosty sposób połączenie z bazą danych. Trzecia linia kodu przedstawia zmienną przechowującą ścieżkę połączenia z bazą danych, a ostatnie dwie odpowiadają za połączenie oraz inicjalizację lub wybór instancji bazy danych. Te linie są wykorzystywane we fragmentach [[lst:insertDB]](#lst:insertDB) oraz [[lst:getFromDB]](#lst:getFromDB) celem inicjalizacji zmiennych lokalnych odpowiadających za komunikację.

##### Umieszczenie danych w bazie danych

Fragment dotyczący umieszczenia danych w bazie danych został zaprezentowany w kodzie [[lst:insertDB]](#lst:insertDB).

def initialize\_db(pointsList):  
 ...  
 myclient.drop\_database("mydatabase")  
 col = mydb["elements"]  
 for element in list(pointsList):  
 for value in element:  
 doc = collections.OrderedDict()  
 doc['Type'] = value.Type  
 doc['CoordX'] = value.CoordX  
 doc['CoordY'] = value.CoordY  
 doc['TimeStamp'] = value.TimeStamp  
 odbcarr.append(doc)  
 col.insert\_many(odbcarr)  
 ...

Parametrem wejściowym funkcji jest lista wszystkich istniejących punktów. Fragment kodu w liniach 3 oraz 4 przedstawia inicjalizację elementów w bazie danych. W pierwszym kroku następuje wyczyszczenie istniejących elementów wraz z usunięciem instancji bazy danych, a potem utworzenie bazy danych *"mydatabase"* z tabelą *"elements"*. Jest to wykonywane za każdym razem, celem usunięcia poprzednich danych. Linie kodu od 6 do 12 przedstawiają umieszczenie elementów typu **OrderedDict** w tablicy. Tego typu obiekty zgodnie z dokumentacją, są najprostszym sposobem realizacji operacji INSERT w bazie danych dla większej ilości elementów. W linii 13 następuje umieszczenie elementów w tabeli.

##### Wydobycie danych z bazy danych

def getFromDatabase():  
 ...  
 elements = mydb["elements"].find()  
 for element in elements:  
 item = Data(element['Type'], element['TimeStamp'], element['CoordX'], element['CoordY'])  
 dataArray.append(item)  
 i = 0  
 loopFlag = True  
 while loopFlag:  
 for element in dataArray:  
 if i + 1 == len(dataArray):  
 retList.append(dataArray[i])  
 loopFlag = False  
 break  
 if dataArray[i + 1].Type == 'SS':  
 retList.append(dataArray[i])  
 break  
 retList.append(dataArray[i])  
 i += 1  
 i += 1  
 returnList.append(retList)  
 retList = []  
 ...

Listing kodu [[lst:getFromDB]](#lst:getFromDB) przedstawia rozwiązanie problemu wydobycia danych z tabeli w bazie danych. Linia 3 wykorzystując bibliotekę **MongoDB** pobiera wszystkie elementy za pomocą funkcji *find()*. Można ją przyrównać do wykonania zapytania

SELECT \* FROM mydatabase.elements;

Następnym krokiem funkcji jest przetworzenie danych na dane czytelne dla algorytmów wykrywania fiksacji, implementację przedstawiono w liniach 4-6. Linie 9-22 odpowiadają za podział wszystkich punktów w celu umieszczenia ich w pamięci aplikacji do tablicy jednowymiarowej, której elementami są punkty przedzielone wartościami ’SS’.

#### Przygotowanie danych do analizy

if sys.argv[3] == 'I-DT':  
 print('Starting measurement using I-DT algorithm')  
 for measurement in parsedMeasurements:  
 print('Starting calibration')  
 m1, convertingTime = convertPointsToCalibration(measurement)  
 print('Ended calibration')  
 coordX, coordY, timealgorithm, fixationsForPoint = calculateIdtAlgorithm(m1)  
 print('Ending measurement using I-DT algorithm')

Na listingu [[lst:datapreparation]](#lst:datapreparation) zaprezentowano metodykę przygotowania danych do analizy za pomocą algorytmów. Przykład przygotowuje dane dla algorytmu I-DT. Pierwsza linia przedstawia fragment pętli wybierającej odpowiedni algorytm, zgodnie z parametrami uruchomieniowymi zaprezentowanymi w sekcji [3.3.1](#ssec:parameters). W pętli **for** elementem iterowanym jest lista odczytanych pomiarów. Każda z tych list jest oddzielona punktem ’SS’. W pierwszym kroku wykonywana jest kalibracja danych. Zwracanymi wartościami są lista punktów skalibrowanych - *m1* oraz czas trwania konwersji. Kolejnym krokiem jest wykonanie algorytmu wykrywania fiksacji.

#### Algorytm I-VT

def calculateIvtAlgorithm(pointList):  
 i = 0  
 for element in pointList:  
 velocity = 0  
 if pointList[i].Type == 'SS':  
 i += 1  
 continue  
 if i + 1 == len(pointList) - 1:  
 velocity = math.sqrt(math.pow(pointList[i + 1].CoordX - pointList[i].CoordX, 2) + math.pow(pointList[i + 1].CoordY - pointList[i].CoordY, 2))  
 if velocity < constants.FIXATION\_VELOCITY\_THRESHOLD:  
 fixations.append(pointList[i])  
 fixations.append(pointList[i + 1])  
 break  
 velocity = math.sqrt(math.pow(pointList[i + 1].CoordX - pointList[i].CoordX, 2) + math.pow(pointList[i + 1].CoordY - pointList[i].CoordY, 2))  
 if (velocity < constants.FIXATION\_VELOCITY\_THRESHOLD):  
 fixations.append(pointList[i])  
 i += 1  
   
 i = 0  
 while i < len(fixations) - 1:  
 velocity = math.sqrt(math.pow(fixations[i + 1].CoordX - fixations[i].CoordX, 2) + math.pow(fixations[i + 1].CoordY - fixations[i].CoordY, 2))  
 if velocity < constants.FIXATION\_VELOCITY\_THRESHOLD:  
 combineFixationsArray.append(fixations[i])  
 else:  
 if len(combineFixationsArray) != 0:  
 coordX.append(sum(sumX.CoordX for sumX in combineFixationsArray) / len(combineFixationsArray))  
 coordY.append(sum(sumY.CoordY for sumY in combineFixationsArray) / len(combineFixationsArray))  
 i += 1  
 return coordX, coordY, end - start, len(coordX)

W kodzie [[lst:ivt]](#lst:ivt) przedstawiono implementację algorytmu I-VT w języku Python, zgodnie z pseudokodem [[lst:ivtpseudocode]](#lst:ivtpseudocode). Jako parametr wejściowy funkcja otrzymuje podzieloną listę punktów zgodnie z opisem w sekcji [3.4.4](#ssec:Dataanalysis). Główna pętla **for** iteruje po każdym punkcie z listy wejściowej. Zadaniem linii 5-7 jest pominięcie każdego punktu ’SS’, gdyż nie jest on potrzebny do dalszych obliczeń. Wiersz 8 prezentuje warunek, gdy badany jest przedostatni i ostatni punkt z listy. Obliczana jest prędkość pomiędzy tymi punktami, i jeżeli warunek nr 2 zaprezentowany w pseudokodzie [[lst:ivtpseudocode]](#lst:ivtpseudocode) jest spełniony, dodaje się te punkty do listy fiksacji. Linie 14 do 17 prezentują działanie algorytmu dla punktów innych niż przedostatni i ostatni. Prędkość pomiędzy tymi elementami jest obliczana zgodnie ze wzorem:

gdzie *p* jest tablicą elementów, *X* i *Y* są współrzędnymi punktu a *i* jest iteratorem tablicy.  
Wiersze 20-28 odpowiadają za podział elementów na grupy fiksacji, zgodnie z linią 3 pseudokodu. Ten podział odbywa się za pomocą weryfikacji prędkości międzypunktowych. W wypadku znalezienia końca grupy fiksacji, następuje wydzielenie środka grupy fiksacji za pomocą równania:

W tym równaniu P jest punktem X lub punktem Y, a N to liczba elementów w tablicy *combineFixationsArray*.  
Po przeanalizowaniu wszystkich punktów wejściowych, zwracane są następujące elementy: współrzędne X fiksacji, współrzędne Y fiksacji, czas trwania algorytmu oraz ilość fiksacji. Jak można zauważyć, w kodzie algorytmu znalazł się parametr **constants.FIXATION\_ VELOCITY\_ THRESHOLD**. Odpowiada on za próg prędkości opisany we fragmencie [2.4.1.1](#ssec:ivt).

#### Algorytm I-DT

W kodzie [[lst:idt]](#lst:idt) przedstawiono zaimplementowany algorytm I-DT, według pseudokodu opisanego w sekcji [[lst:idtpseudocode]](#lst:idtpseudocode). Jako parametr wejściowy ta funkcja przyjmuje podzieloną listę punktów. Ten podział zaprezentowano w sekcji [3.4.4](#ssec:Dataanalysis). W 4 linii znajduje się główna pętla **while** algorytmu, działająca dopóki nie zostaną zanalizowane wszystkie punkty. Tak jak w algorytmie I-VT [[lst:ivt]](#lst:ivt), punkty ’SS’ są pomijane, co zapisano w liniach 5-7. Następnie należy przygotować wstępne okno do przeprowadzenia obliczeń. Zgodnie z dokumentacją, potrzeba do tego celu parametru czasowego rozmiaru okna, który określono jako **constants.WINDOW\_TIME\_THRESHOLD**. Tworzenie okna pokazano w pętli **while** pomiędzy liniami 8-14. Dwie zmienne lokalne **timeStart** i **currTime** służą do pomiaru czasu. W wypadku przekroczenia ilości elementów w liście wejściowej pętla główna jest przerywana. Po dodaniu wszystkich elementów zgodnych z warunkiem pętli należy obliczyć dyspersję w oknie. Zastosowano do tego następujący wzór:

Kolejnym krokiem, po przygotowaniu wstępnego okna, jest porównanie czy dyspersja spełnia warunek graniczny opisany w kroku 3 pseudokodu. Warunek graniczny przechowywany jest w zmiennej **constants. DISPERSION\_THRESHOLD**. W wypadku gdy dyspersja jest mniejsza, należy dodawać kolejne punkty aż zostanie przekroczony warunek graniczny, co zaprezentowano w pętli **while** pomiędzy wierszami 22 i 27. Po przekroczeniu wartości granicznej zostaje dodany do tablic wyjściowych punkt centralny okna, obliczony za pomocą wzoru

P może być punktem X lub Y. Po tym okno jest czyszczone do momentu spełnienia warunku granicznego, i algorytm jest wykonywany do skończenia punktów.  
Algorytm zwraca listę współrzędnych X i Y, czas trwania algorytmu oraz ilość fiksacji.

def calculateIdtAlgorithm(pointsList):  
 timeStart = int(pointsList[0].TimeStamp)  
 countPoints = len(pointsList)  
 while i < countPoints - 1:  
 if pointsList[i].Type == 'SS':  
 i += 1  
 continue  
 currTime = int(pointsList[i].TimeStamp)  
 while currTime - timeStart <= constants.WINDOW\_TIME\_THRESHOLD:  
 windowList.append(pointsList[i])  
 i += 1  
 if i >= countPoints:  
 break  
 currTime = int(pointsList[i].TimeStamp)  
 if i >= countPoints:  
 break  
 if len(windowList) > 1:  
 Dispersion = (max(maxX.CoordX for maxX in windowList) - min(minX.CoordX for minX in windowList)) + (max(maxY.CoordY for maxY in windowList) - min(minY.CoordY for minY in windowList))  
 while len(windowList) > 1:  
 if Dispersion <= constants.DISPERSION\_THRESHOLD and len(windowList) > 1:  
 while (Dispersion < constants.DISPERSION\_THRESHOLD):  
 windowList.append(pointsList[i])  
 i += 1  
 if i >= countPoints:  
 break  
 Dispersion = (max(maxX.CoordX for maxX in windowList) - min(minX.CoordX for minX in windowList)) + (max(maxY.CoordY for maxY in windowList) - min(minY.CoordY for minY in windowList))  
 if i >= countPoints:  
 break  
 coordXList.append(sum(sumX.CoordX for sumX in windowList) / len(windowList))  
 coordYList.append(sum(sumY.CoordY for sumY in windowList) / len(windowList))  
 else:  
 windowList.pop(0)  
 if i <= countPoints - 1:  
 timeStart = int(pointsList[i].TimeStamp)  
 return coordXList, coordYList, end - start, len(coordXList)

#### Algorytm wykorzystujący uczenie maszynowe

Opis teoretyczny biblioteki przeznaczonej do wykorzystywania uczenia maszynowego bazuje na pracy .

Jak opisano w rozdziale [2.4.1.3](#ssec:machinelearning), uczenie maszynowe ze względu na rozwój technologiczny stało się jedną z najczęściej wykorzystywanych technologii, szczególnie przy analizie danych. W celu użycia tych rozwiązań, w języku Python należy wykorzystać pakiet **scikit-learn**. Zezwala on na stworzenie projektu z wykorzystaniem wielu dostępnych klasyfikatorów uczenia maszynowego np. k-NN[[9]](#footnote-106), SVC, regresji liniowej.  
Celem wykonywania przewidywań w aplikacji wykorzystano model regresji logistycznej. Jest to metoda statystyczna, której zadaniem jest przewidywanie wyników dla klas binarnych, przez co znajduje ona idealne zastosowanie dla algorytmów wykrywania fiksacji. Różni się ona od regresji liniowej danymi wyjściowymi. Podczas gdy regresja liniowa jako wynik prezentuje dane ciągłe, to regresja logistyczna przyjmuje wartości stałe. Oznacza to, iż regresja liniowa może przewidywać więcej wartości. Wyróżnia się jednak dodatkowe typy regresji logistycznych, takie jak binarna regresja, gdzie wynikiem może być podział na to czy wiadomość należy do spamu czy nie oraz wielomianowa regresja, gdzie rezultatem może być przykładowo typ wina.  
Celem uruchomienia algorytmu uczenia maszynowego należy zbudować model danych. Przykład takiego modelu zaprezenotwano w kodzie [[lst:mlhelperclass]](#lst:mlhelperclass). Ten kod przedstawia konstruktor klasy pomocniczej **MLHelper**. Posiada ona cztery elementy: informacje o punkcie, obliczoną prędkość pomiędzy tym punktem oraz następnym, odległość międzypunktową oraz wartość fiksacji w tym punkcie obliczoną za pomocą algorytmu [2.4.1.1](#ssec:ivt). Podczas projektowania aplikacji próbowano dodać dodatkowe parametry takie jak odległości w oknie dla punktu oraz dane statystyczne dotyczące okna, jednak powodowały one przesycenie modelu, co powodowało błędy w obliczeniach oraz brak fiksacji na wyjściu. Drugim powodem braku ich implementacji był bardzo duży przyrost czasu przeznaczanego na przygotowanie modelu danych, ze względu na dużą ilość danych, średnio kilkaset tysięcy na plik.

def \_\_init\_\_(self, Type, CoordX, CoordY, TimeStamp, \*args, \*\*kwargs):  
 self.Data = Data(Type, TimeStamp, CoordX, CoordY)  
 self.VelocityBetweenPoints = float(0)  
 self.DistancesBetweenPoints = float(0)  
 self.IsFixation = bool

W kodzie [[lst:mlalgorithm]](#lst:mlalgorithm) przedstawiony został algorytm wykorzystujący uczenie maszynowe. Początek algorytmu, czyli linie od 2 do 5 przedstawiają inicjalizację tablic typu **numpy.array**, które umożliwiają na bardziej skomplikowane operacje na tablicach, podobne do tych używanych w języku R oraz MatLAB. Przykładem wykorzystania takich operacji jest linia 14, gdzie zdecydowano na przekazanie tylko wartości ’VEL’, ’DBP’ oraz ’FIX’ do modelu danych. Na początku funkcji inicjalizowane są tablice X1, Y1 dla danych treningowych, oraz X2, Y2 dla danych testowych. Następnie w liniach 6-15 przedstawiono pobranie danych z parametrów wejściowych funkcji oraz ich konwersję na format czytelny dla biblioteki **numpy**. Należało wykonać konwersję z obiektu **class** na słownik, gdyż obiekt **pd.DataFrame** przyjmuje listę obiektów iterowalnych jako swój parametr. Przykład realizacji takiej konwersji pokazano na listingu [[lst:iter]](#lst:iter).

def \_\_iter\_\_(self):  
 yield 'Data', self.Data  
 yield 'VEL', self.VelocityBetweenPoints  
 yield 'DBP', self.DistancesBetweenPoints  
 yield 'FIX', 1 if self.IsFixation == True else 0

W linii 18 postawiono warunek podziału ile obiektów ma służyć jako trening dla modelu, a ile przeznaczyć na testowanie. Zdecydowano się na podzielenie zbioru danych na zbiory: 10% danych treningowych, 90% danych testowych. Należało te dane połączyć, co przedstawia funkcja **np.concatenate[X,Y]**, przyjująca dwie tablice o tych samych wymiarach jako swoje parametry.  
W linii 26 po wyborze typu modelu następuje jego uczenie za pomocą tablic testujących. Następny wiersz przedstawia wykonanie predykcji. Po obliczeniu wszystkich wartości wynikowych należy zbadać dokładność algorytmu, co zostało pokazane w linii 29. **Numpy** udostępnia funkcje sprawdzające dokładność oraz precyzję klasyfikatorów uczenia maszynowego.

def calculateML(pointList):  
 start = time.process\_time()  
 XArr = np.empty([1,3])  
 YArr = np.empty([1])  
 XArr2 = np.empty([1,3])  
 YArr2 = np.empty([1])  
 for i, point in enumerate(pointList):  
 tmpArr = list()  
 for item in point:  
 el = dict(item)  
 tmpArr.append(el)  
 values = pd.DataFrame(tmpArr, columns=['Data', 'VEL', 'DBP', 'FIX'])  
 array = values.values  
 X = array[:,0:3]  
 Y = array[:,3]  
 Y=Y.astype('int')  
 if len(X) != len(Y):  
 print('W punkcie ' + i + ' jest bład')  
 if i < 3:  
 XArr = np.concatenate([XArr,X])  
 YArr = np.concatenate([YArr,Y])  
 else:  
 XArr2 = np.concatenate([XArr2,X])  
 YArr2 = np.concatenate([YArr2,Y])  
 print(XArr[1:,1:])  
 model = LogisticRegression()  
 model.fit(XArr[1:,1:], YArr[1:])  
  
 prediction = model.predict(XArr2[1:,1:])  
  
 ite = accuracy\_score(YArr2[1:], prediction)  
 print(ite)  
 endXArr = np.concatenate([XArr[1:,:],XArr2[1:,:]])  
 endYArr = np.concatenate([YArr[1:],prediction])  
 endAll = np.c\_[endXArr,endYArr]  
  
 retX = []  
 retY = []  
 print(endAll)  
 for item in endAll:  
 if item[3] == 1:  
 retX.append(item[0].CoordX)  
 retY.append(item[0].CoordY)  
 end = time.process\_time()  
 return retX, retY, len(retX), end - start, ite

#### Prezentacja wyników

import pyplot as plt  
 ...  
 plt.plot(m1[i].CoordX, m1[i].CoordY, 'ko', markersize=10, label='Eye-tracker points' if i == 0 else "")  
 ...  
 coordX, coordY, timealgorithm, fixationsForPoint = calculateIdtAlgorithm(m1)  
 plt.plot(coordX, coordY, 'wo', markersize=5, markeredgecolor='r', label='Calculated fixations')  
 ...  
 plt.legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3, ncol=2, mode="expand", borderaxespad=0.)  
 plt.show()

Listing [[lst:plot]](#lst:plot) prezentuje wykorzystanie biblioteki **pyplot** odpowiadającej za tworzenie wykresów oraz wyświetlanie punktów na układzie współrzędnych w języku Python. Funkcja *plot()* przyjmuje jako parametry wejściowe następujące wartości: X, Y, sposób prezentacji, rozmiar linii/punktu, legendę. Celem wyświetlenia wartości należy użyć funkcji **plt.show()**. Otwiera ona nowe okno z naniesionymi wartościami. Przykład takiego okna zaprezentowano na rysunku [3.4](#fig:presentationfixation).

#### Pomiar czasu

W trakcie wykonywania aplikacji wykonywany jest pomiar czasu działania różnych funkcji aplikacji. W tym celu została wykorzystana biblioteka **time**. W kodzie [[lst:time]](#lst:time) zaprezentowano sposób działania biblioteki. Funkcja **time.process\_time** zwraca obecny czas systemowy w sekundach, więc różnica tych wartości prezentuje czas trwania danego kodu.

start = time.process\_time()  
 # wykonywana akcja  
 end = time.process\_time()  
 return ..., end - start, ...

#### Pomiar wykorzystania pamięci

W celu przeprowadzenia pomiaru wykorzystania pamięci w aplikacji należy użyć biblioteki **memoryprofiler**. Sposób implementacji przedstawiono w kodzie [[lst:memory]](#lst:memory). Ten moduł został użyty do całej aplikacji, przez zdefiniowanie go przed funkcją **main**. Ta biblioteka wypisuje zużycie pamięci w MB przez każdą linijkę aplikacji, oraz zapisuje ją do pliku poprzez użycie parametru ’stream’. W wypadku aplikacji zastosowano precyzję do 4 miejsc po przecinku, ze względu na zakładaną możliwość mniejszego zużycia pamięci przez podstawowe algorytmy. Określenie tej precyzji znajduje się w parametrze *precision*.

@profile(stream=open('result/' + sys.argv[2] + '-' + sys.argv[3] + '.log','w+'), precision=4)  
def main(argv):

Na rysunku [3.3](#fig:memoryfile) zaprezentowano fragment zastosowania tej biblioteki.

## Badania

W tym rozdziale przedstawiono przeprowadzone badania, sposób ich przeprowadzenia, przedstawiono sprzęt, na którym przeprowadzono badania.

### Maszyna do badań

Wszystkie testy oraz badania zostały przeprowadzone na maszynie zaprezentowanej w tabeli [[tab:machine]](#tab:machine). W tabeli wymieniono tylko znaczące elementy, tzn. procesor, pamięć RAM oraz jej szybkość, dysk twardy, jego prędkość obrotową oraz typ. Wszystkie te podzespoły mogą wpływać na czas trwania elementów działania aplikacji. Ze względu na jednowątkowość aplikacji, nie podaje się ilości wątków.

Maszyna do badań

|  |  |
| --- | --- |
| Procesor | Intel Core-i5 8600 @ 3.10 GHz |
| Pamięć RAM | 16GB DDR4 @ 3200 MHz |
| Typ dysku twardego | HDD |
| Dysk twardy | Toshiba HDWD110 |
| Prędkość obrotowa | 7200 obr./min |

### Parametry stanowiska pomiarowego

30 pomiarów

### Przeprowadzone badania

#### Wybór metodyki regresji

#### Analiza wpływu parametrów algorytmów

#### Porównanie czasów trwania algorytmów

#### Analiza obciążenia pamięci

#### Wpływ typu danych wejściowych

#### Dokładność algorytmów i porównanie punktów wynikowych

## Podsumowanie i wnioski

99 Dario D. Salvucci, Joseph H. Goldberg (2000), *Identifying Fixations and Saccades in Eye-Tracking Protocols*, Eye Tracking Research and Applications Symposium 2000 pod red. Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, itd. *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research 12 (2011) 2825-2830 Gustav Larsson, *Evaluation Methodology of Eye Movement Classification Algorithms*, Królewski Instytut Technologiczny w Sztokholmie Jian Mou (2017), Effects of social popularity and time scarcity on online consumer behaviour regarding smart healthcare products: An eye-tracking approach. *Computers in Human Behavior, 78, 74–89.*, <https://doi.org/10.1016/j.chb.2017.08.049> Karl Seguin (2011), *The Little MongoDB Book*,  
<https://openmymind.net/mongodb.pdf> Leslie Lamport (1994), *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, Maria Laura Mele, Stefano Frederici (2012) *Gaze and eye-tracking solutions for psychological research*, Cogn Process (2012) 13 (Suppl 1):S261–S265, Mark Lutz (2009) *Python. Wprowadzenie. Wydanie IV*, HELION S.A. Michael Raschke, Tanja Blascheck, Michael Burch (2013), *Visual Analysis of Eye Tracking Data*, Institute for Visualization and Interactive Systems, University of Stuttgart, Germany M. F. Land, D. N. Lee (1994), *Where we look when we steer*, Nature 369(6483):742-744 Raimondas Zemblys, Diederick C. Niehorster, Oleg Komogortsev, Kenneth Holmqvist, *Using machine learning to detect events in eye-tracking data*, Psychonomic Society, Inc. 2017 Robert Gabriel Lupu\*, Florina Ungureanu (2013), *A Survey Of Eye Tracking Methods And Applications*, “Gheorghe Asachi” Technical University of Iaşi, Rozdział 2

# Dodatki

### Zawartość płyty

Do pracy dołączona została płyta z następującą zawartością: oryginał pracy magisterskiej, pliki źródłowe aplikacji, prezentacja, praca magisterska posiadająca tylko tekst.

1. ang. area of interest [↑](#footnote-ref-45)
2. z ang. Identification-Velocity Threshold [↑](#footnote-ref-48)
3. z ang. Identification Dispersion-Threshold [↑](#footnote-ref-52)
4. Machine Learning [↑](#footnote-ref-54)
5. z ang. Graphical User Interface - graficzny interfejs użytkownika [↑](#footnote-ref-62)
6. Machine Learning .NET [↑](#footnote-ref-64)
7. Należy nadmienić, iż chodzi o standard .NET Framework, a nie .NET Core, dla którego F# posiada otwarty kod źródłowy [↑](#footnote-ref-65)
8. binary JSON [↑](#footnote-ref-66)
9. k najbliższych sąsiadów [↑](#footnote-ref-106)