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

[chapter:ch1] 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 dostarcza reklamodawcy kolejne informacje dotyczące sposobu tworzenia reklam, by rozszerzyć zasięg osób, do których one trafiają.

Wraz z rozwojem okulografii, wzrosło zapotrzebowanie na co raz dokładniejsze urządzenia służące 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, liczby wykrytych fiksacji. Badany również będzie wpływ parametrów zewnętrznych na poprawność i dokładność algorytmów. W pracy zaprezentowano również 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 charakterystyki utworzonej aplikacji, wraz z opisem wykorzystanych narzędzi. Po tym zaprezentowano specyfikację zewnętrzną aplikacji, w tym parametry uruchomieniowe dla programu, 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: implementacja obsługi bazy danych, opis przygotowania danych do analizy za pomocą wybranych algorytmów. Sekcje [3.4.5](#ssec:implementidt), [3.4.4](#ssec:implementivt), [3.4.6](#ssec:machinelearningalg) prezentują wdrożone algorytmy wykrywania fiksacji. Ostatnie 3 sekcje opisują w jaki sposób zaprezentowano wyniki, odczyt i zapis plików, 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, danymi pomiarowymi. Następna sekcja została poświęcona sprawdzeniu wpływu parametrów wejściowych algorytmów na wyniki końcowe. Analiza ta odbywa się za pomocą porównania czasów trwania algorytmów, sprawdzenia wykorzystania pamięci przez algorytmy oraz analizę liczby otrzymanych fiksacji. Ostatni fragment w tym rozdziale przyrównuje te wyniki do siebie, oraz przeprowadza dokładniejszą analizę tych wyników. 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 .

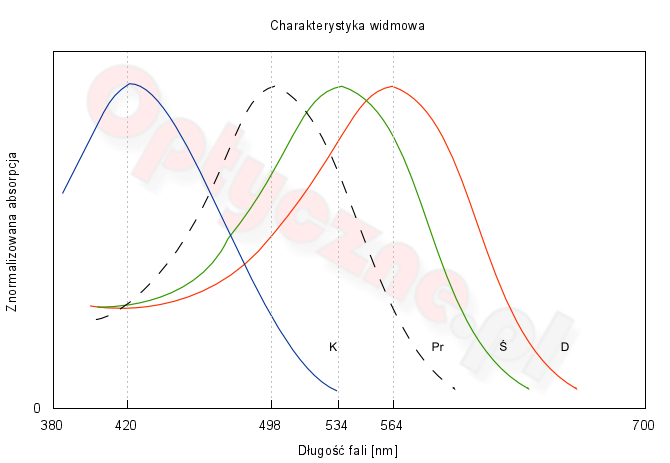
Jednym z najważniejszych organów w ludzkim ciele jest oko. Służy ono do dostarczania większości informacji dotyczących otoczenia w jakim się znajdujemy, oraz poszerzania wiedzy o świecie. Około 10% komórek mózgowych jest zaangażowanych przy interpretacji oraz analizie sygnałów dostarczanych z tego narządu.

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:

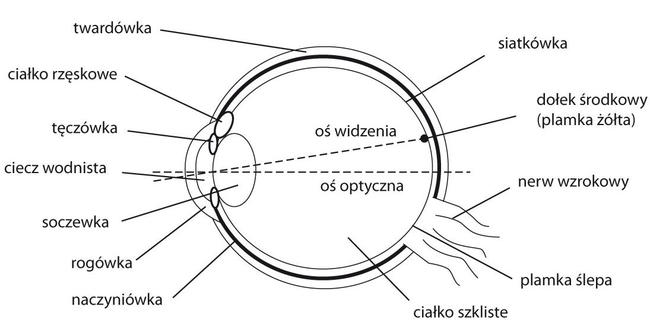
* 6 milionów czopków
* 100 milionów pręcików

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: [dostęp 26.10.2019])

Na siatkówce pozyskany obraz 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: [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.

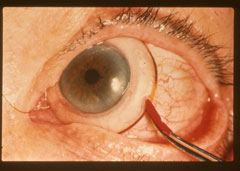
Najważniejszym dla nas elementem siatkówki jest plamka żółta, posiadająca największą rozdzielczość otrzymywanego obrazu, ze względu na najwyższe stężenie czopków w ludzkim oku. Umożliwia ona ludziom np. czytanie tekstu. Im dalej od plamki żółtej, tym zwiększa się ilość pręcików, a przez to zmniejsza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



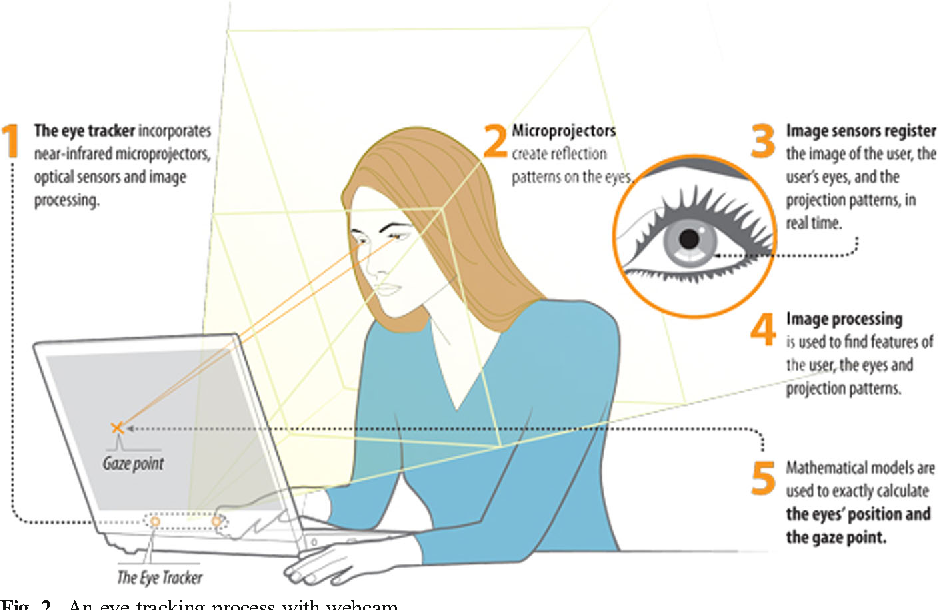
Soczewki do wykrywania ruchu oka.  
(źródło: [dostęp 10.09.2019])

#### Elektrookulografia

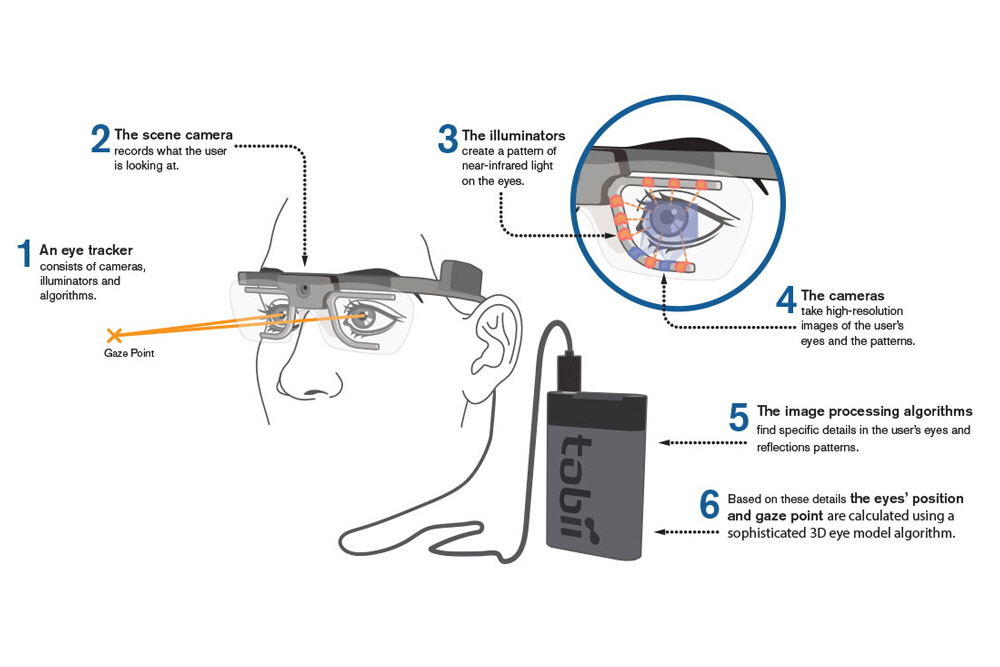
Elektrookulografia jest typem rozwiązań bazującym na różnicy potencjałów pomiędzy elektrodami przymocowanymi blisko oczu. Ze względu na dużą ilość połączeń nerwowych w gałce ocznej, można wykryć zmianę napięcia w samej gałce, gdzie wynosi ona około 1mV, jak również zauważalna jest zmiana pól elektrycznych 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



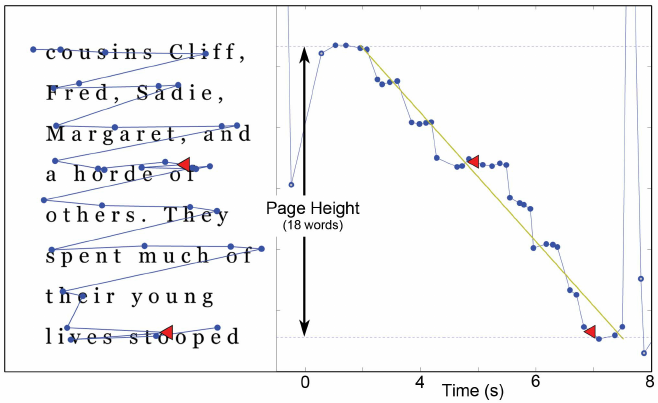
Kamera w okularach

### 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: [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-43). 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-46) 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-50). 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.

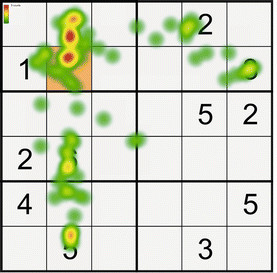
##### Uczenie maszynowe

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-52). 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.

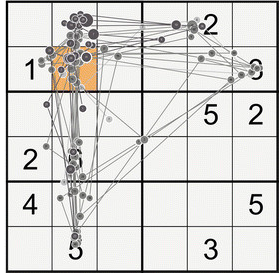
Opis działania algorytmu, jak również sposób analizy danych został zaprezentowany w sekcji [3.4.6](#ssec:machinelearningalg).

### 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, prezentacja 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. 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, 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-60), 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. Uruchom aplikację podając odpowiednie parametry wejściowe.
2. Pobierz dane wejściowe z wybranego źródła.
3. Dla grupy punktów o typie ’R’ oddzielonej punktem posiadającym parametr ’SS’ uruchom wybrany algorytm
4. Zaprezentuj wynik algorytmu na rysunku dla użytkownika
5. 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 ułatwiłoby rozwiązanie problemu stworzenia algorytmu wykorzystującego uczenie maszynowe zaprezentowane w sekcji [3.4.6](#ssec:machinelearningalg), ze względu na prostą metodę integracji go z modułem *ML.NET*[[6]](#footnote-62). Język R w budowie jest bardzo podobny do języka F#, ale jego kod źródłowy jest otwarty, w przeciwieństwie do F#[[7]](#footnote-63) jednak podstawowa znajomość języka Python zadecydowała o wykorzystaniu tej platformy. Wszystkie wymagane moduły oraz sposób użycia opisano w sekcji [3.4.1](#ssec:apprequirements).

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-64), 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 konsolowa, 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 momencie uruchomienia skryptu była wartość *C:/Użytkownicy/nazwaużytkownika*. Ostatnia linia skryptu wymusza na użytkowniku wciśnięcie klawisza żeby zamknąć okno aplikacji.

Wszystkie wymagania do uruchomienia skryptu 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:

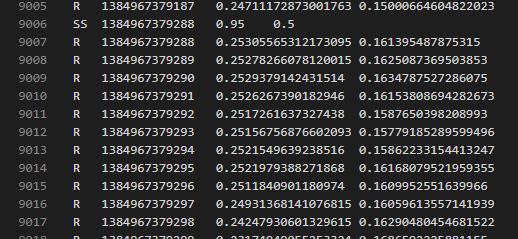
* *’ML’*, algorytm wykorzystujący uczenie maszynowe.
* *’I-DT’*, algorytm I-DT,
* *’I-VT’*, algorytm I-VT

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 wszystkie 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 współrzędne punktów wejściowych, odpowiednio X i Y.

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.

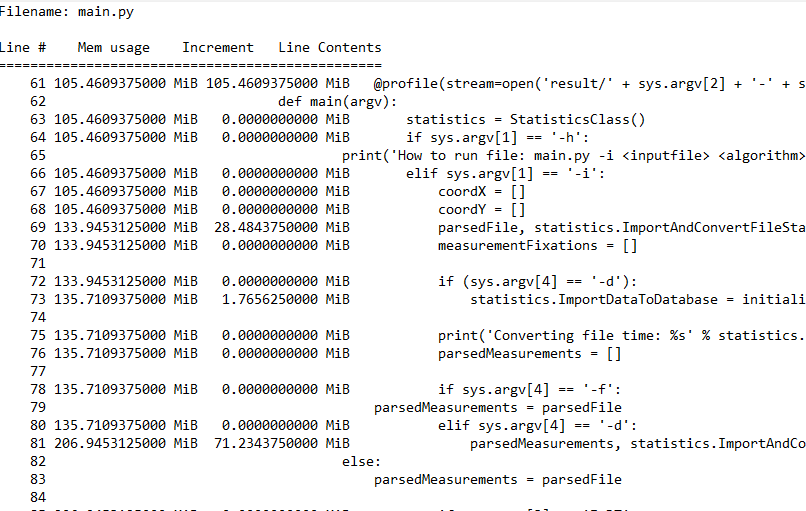
Rezultatem działania programu są trzy pliki, plik *.csv* zawierający obliczone dane, plik *.png* będący reprezentacją graficzną obliczonych fiksacji oraz plik *.log* przechowujący zrzut wykorzystania pamięci w czasie trwania aplikacji. Zapis danych odbywa się w momencie zakończenia obliczeń. Opis pliku graficznego został zaprezentowany w sekcji [3.3.4](#ssec:fixations).  
Plik *.csv* zostaje zapisany w folderze **/result**, posiadając w nazwie tytuł pliku wejściowego, oraz 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

Plik wyjściowy

1. ImportAndConvertFileStatistic
2. SaccadeCount
3. AlgorithmRunTimeStatistic
4. NumberOfFixationsCount
5. ImportDataToDatabase
6. ImportAndConvertDatabaseStatistic
7. MLPrecision

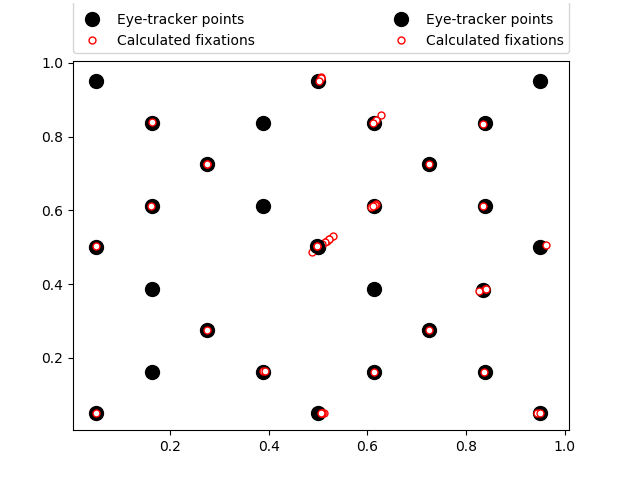
Wszystkie obliczone czasy podano w sekundach z dokładnością do 4 miejsc po przecinku. Pierwsza kolumna prezentuje czas trwania importu danych do pamięci oraz czas trwania konwersji plików do formatu czytelnego dla aplikacji, zgodnie z fragmentem [3.3.2](#ssec:importdata). Następna kolumna przedstawia liczbę punktów należących do sakad. Trzecia kolumna prezentuje czas trwania algorytmu wykrywania fiksacji, a czwarta liczbę 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. Ostatnia kolumna jest również opcjonalna, zawiera ona w sobie wynik zliczonej precyzji algorytmu wykorzystującego uczenie maszynowe.  
Celem obliczenia zużycia przez algorytm pamięci operacyjnej 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

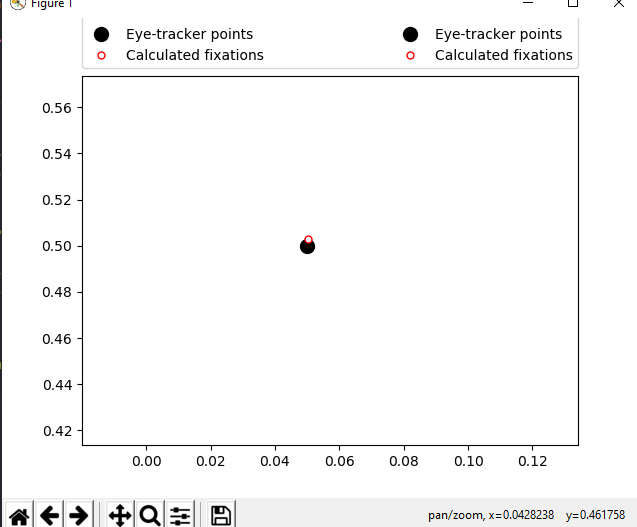
Pierwsza kolumna prezentuje linię w pliku, druga wykorzystanie pamięci, trzecia różnicę w pomiarach pomiędzy poprzednią linią a obecną a ostatnia treść linii. Czwarta kolumna pozwala w łatwy sposób zinterpretować, jak dużo pamięci pobiera algorytm obliczania fiksacji.

#### 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).

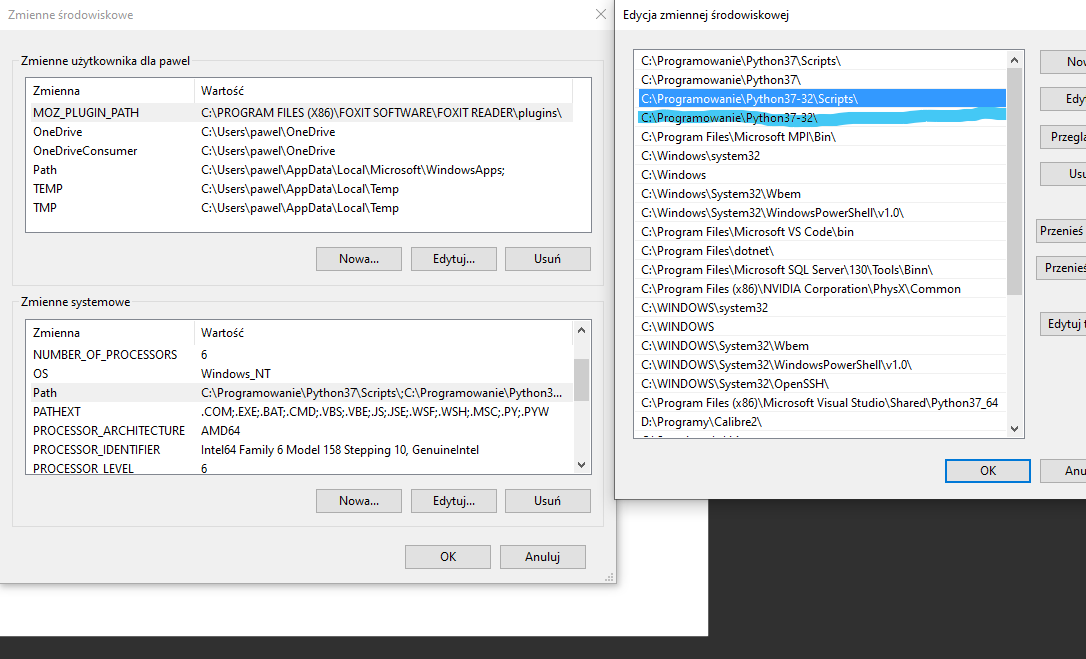
### Specyfikacja wewnętrzna aplikacji

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 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. przy tworzeniu modelu dla danych związanych z uczeniem maszynowym. Wykorzystane rozwiązanie zaprezentowano w sekcji [3.4.6](#ssec:machinelearningalg). Kolejną biblioteką jest **Matplotlib.pyplot**, umożliwiająca wyświetlanie danych wyjściowych. Moduł zaprezentowano w podrozdziale [3.4.7](#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**, którego celem jest wykonywanie zrzutu pamięciowego funkcji w ciągu działania aplikacji. Ostatnim modułem, który wykorzystuje się w sekcji [3.4.6](#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).

#### 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.

##### Odczyt 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:  
 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 wybranych algorytmów wykrywania fiksacji. 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’. Po wykonaniu podziału na indywidualne pomiary następuje zmierzenie wartości wynikowych algorytmu.

#### 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 [3.4.3](#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 przypadku 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 [3.4.3](#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 liczby elementów w liście wejściowej pętla główna jest przerywana. Po dodaniu wszystkich elementów zgodnych z warunkiem pętli należy obliczyć wielkość rozproszenia punktów 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 przypadku 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 liczbę 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-101), 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 zaprezentowano 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 pozwalają 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 do modelu danych tylko wartości ’VEL’, ’DBP’ oraz ’FIX’. 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 jako swój parametr listę obiektów iterowalnych. 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 obiektów na dane treningowe i testowe. Zdecydowano się przykładowo na podzielenie zbioru danych na zbiory: 20% danych treningowych, 80% danych testowych. Należało te dane połączyć, co przedstawia funkcja **np.concatenate[X,Y]**, przyjmująca 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])  
 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')  
 XArr = np.concatenate([XArr,X])  
 YArr = np.concatenate([YArr,Y])  
 x\_learn, x\_test, y\_learn, y\_test = model\_selection.train\_test\_split(XArr, YArr, test\_size=0.8)  
 y\_learn=y\_learn.astype('int')  
 y\_test=y\_test.astype('int')  
 model = LogisticRegression()  
 model.fit(x\_learn[1:,1:], y\_learn[1:])  
   
 prediction = model.predict(x\_test[1:,1:])  
   
 ite = model.score(x\_test[1:,1:],y\_test[1:])  
 print(ite)  
 endXArr = np.concatenate([x\_learn[1:,:],x\_test[1:,:]])  
 endYArr = np.concatenate([y\_learn[1:],prediction])  
 endAll = np.c\_[endXArr,endYArr]  
   
 retX = []  
 retY = []  
 measurementFixations = []  
 for item in endAll:  
 if item[3] == 1:  
 retX.append(item[0].CoordX)  
 retY.append(item[0].CoordY)  
 measurementFixations = item[0]  
 end = time.process\_time()

#### 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 w 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, ...

#### Obsługa plików

Celem tej sekcji jest zaprezentowanie sposobu odczytu danych z pliku cal, oraz wykorzystania biblioteki *csv* służącej do zapisywania plików typu .csv. [ssec:filehandling]

##### Odczyt danych z pliku

W kodzie [[lst:readfile]](#lst:readfile) przedstawiono metodykę odczytu danych z pliku *.cal* oraz sposób konwersji tych danych na obiekt czytelny dla użytkownika, wraz z oddzieleniem każdego pomiaru za pomocą elementu ’SS’. Pierwsza pętla **for** od linii 3 do linii 9 bezpośrednio odpowiada za odczyt danych z pliku. W pierwszym kroku odczytuje ona wszystkie pliki za pomocą funkcji *os.listdir()*, następnie sprawdza czy plik posiada nazwę zgodną z parametrem wejściowym *fileName*. W linii 5 opisano wykorzystanie dyrektywy *with*, odpowiadającej dyrektywie *using* z języka C#, tzn. usuwającej z pamięci operacyjnej obiekt po wyjściu z wcięcia w języku Python.  
Od linii szóstej w pętli sprawdzana jest każda linia pliku, następnie tworzona jest tablica przechowująca elementy typu *string*, które tworzone są za pomocą podzielenia linii przez znak specjalny TAB. Dodawane są one następnie jako specjalny obiekt do tymczasowej listy.  
Po skończeniu operacji w pierwszej pętli od linii 10 do 23 przedstawiono pętle rozdzielającą listę wszystkich pomiarów na tablicę list pomiarów. Elementem rodzielającym listę jest pomiar typu ’SS’. Funkcja zwraca tablicę list pomiarów.

def createObjectsFromFile(fileName):  
 for file in os.listdir('./data'):  
 if file.endswith(fileName):  
 with open('./data/' + file, newline='') as inputfile:  
 for line in inputfile:  
 splittedLine = line.strip('\r\n').split('\t') # co linię, oddziel o znak tab  
 results.append(Data(splittedLine[0], splittedLine[1], splittedLine[2], splittedLine[3]))  
 i = 0  
 loopFlag = True  
 while loopFlag:  
 for element in results:  
 if i + 1 == len(results):  
 retList.append(results[i])  
 loopFlag = False  
 break  
 if results[i + 1].Type == 'SS':  
 retList.append(results[i])  
 break  
 retList.append(results[i])  
 i += 1  
 i += 1  
 returnList.append(retList)  
 retList = []  
 return returnList

##### Zapis danych do pliku

Na listingu [[lst:writefile]](#lst:writefile) pokazano sposób tworzenia oraz zapisywania plików wyjściowych do formatu CSV. Wykorzystano w tym celu specjalną bibliotekę *csv*, domyślnie instalowaną z pakietem Python.  
We fragmencie od linii 2 do 8 włącznie utworzono słownik nagłówków, jakie plik csv powinien posiadać. W wypadku braku tego słownika, aplikacja zgłasza błąd. W linii 10 otwierany jest obiekt dla pliku, analogicznie do rozwiązania zaprezentowanego w sekcji [3.4.9.1](#sssec:readfile), tylko wykorzystując parametr **’w’**, odpowiadający za zapis. W liniach 11,12 tworzony jest plik przyjmujący słownik jako swoje pola oraz dodawany jest do niego nagłówek. W ostatniej linii algorytmu tworzony jest jeden rząd, w którym umieszczone są statystyki pomiaru.

def createExitFile(fileName, statisticClass, alg):  
 fieldNames = ['ImportAndConvertFileStatistic',  
 'SaccadeCount',  
 'AlgorithmRunTimeStatistic',  
 'NumberOfFixationsCount',  
 'ImportDataToDatabase',  
 'ImportAndConvertDatabaseStatistic',  
 'MLPrecision']  
 now = datetime.datetime.now()  
 with open('./result/' + alg + fileName + now.strftime("%d-%m-%Y-%H%M%S") + '.csv', 'w', newline='') as csvFile:  
 writer = csv.DictWriter(csvFile, fieldnames=fieldNames)  
 writer.writeheader()  
 writer.writerow({  
 'ImportAndConvertFileStatistic': str(statisticClass.ImportAndConvertFileStatistic),  
 'SaccadeCount': str(statisticClass.SaccadeCount),  
 'AlgorithmRunTimeStatistic': str(statisticClass.AlgorithmRunTimeStatistic),  
 'NumberOfFixationsCount': str(statisticClass.NumberOfFixationsCount),  
 'ImportDataToDatabase': str(statisticClass.ImportDataToDatabase),  
 'ImportAndConvertDatabaseStatistic': str(statisticClass.ImportAndConvertDatabaseStatistic),  
 'MLPrecision': str(statisticClass.MLPrecision)  
 })

#### 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**. Biblioteka ta wypisuje w MiB[[10]](#footnote-116) zużycie pamięci przez każdą linijkę aplikacji, oraz zapisuje ją do pliku poprzez użycie parametru ’stream’. W przypadku aplikacji zastosowano precyzję do 5 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

Celem tego rozdziału jest zaprezentowanie otrzymanych wyników wykrywania fiksacji, oraz dodatkowych danych statystycznych z nimi powiązanych, takich jak czas trwania algorytmu, wykorzystanie pamięci operacyjnej. Dodatkowo dla algorytmu uczenia maszynowego obliczona została jego precyzja. Do każdych z rezultatów została dołączona krótka analiza. Rozdział rozpoczęto od prezentacji stanowiska, na którym zostały przeprowadzone pomiary, a następnie stworzono krótką charakterystykę danych wejściowych. Kolejną część przeznaczono na opis wpływu parametrów wejściowych na otrzymane wyniki. Dalej stworzono krótką analizę wpływu źródła danych na łączny czas trwania algorytmu. W ostatniej sekcji porównano otrzymane wyniki oraz przeprowadzono ich analizę.

### Opis komputera

Stworzenie oraz testy aplikacji wykrywającej fiksacje zostały przeprowadzone na maszynie zaprezentowanej w tabeli [[tab:machine]](#tab:machine). Wymieniono w niej tylko elementy mogące wpłynąć na ostateczne wyniki, tzn. procesor wraz z jego taktowaniem, pamięć RAM, typ dysku twardego, na którym dane były przechowywane, jego prędkość obrotową. Pominięto takie elementy jak ilość rdzeni lub wątków, ze względu na jednowątkowość utworzonego projektu. Obecnie taką maszynę uznaje się za średnio-wydajną maszynę. Również pominięto prędkość odczytu z dysku SSD, pomimo znalezienia się na nim bazy danych, gdyż połącznenie jest wykonywane za pomocą protokołu *http*.

Spis części komputera pomiarowego

|  |  |
| --- | --- |
| 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 danych wejściowych

Dla każdego pliku umieszczonego w folderze *data* w katalogu głównym aplikacji można określić następującą charakterystykę. Wykonane zostało 30 pomiarów punktów, w folderze istnieje 48 plików z pomiarami. Średnią ilością otrzymanych pomiarów, włącznie z punktami ’SS’ jest 99059 pomiarów. Pomiary zostały wykonywane z częstotliwością 1000 Hz, czyli co 1 ms. W tabeli [[tab:measurementstats]](#tab:measurementstats) zaprezentowano indywidualną liczbę pomiarów w każdym pliku.

Statystyki pomiarów

|  |  |
| --- | --- |
| **Kontynuacja** |  |
|  |  |
| 1\_01\_1311201811.cal | 99129 |
| 1\_02\_1310301856.cal | 99172 |
| 1\_03\_1310301904.cal | 99207 |
| 1\_04\_1310301810.cal | 99150 |
| 1\_05\_1310301817.cal | 99210 |
| 1\_06\_1310301837.cal | 99182 |
| 1\_12\_1310301901.cal | 99026 |
| 1\_14\_1310301912.cal | 99176 |
| 1\_15\_1310301840.cal | 99110 |
| 1\_16\_1310301844.cal | 99190 |
| 1\_17\_1310301825.cal | 99209 |
| 1\_20\_1310301822.cal | 99079 |
| 1\_21\_1310301833.cal | 99114 |
| 1\_22\_1310301908.cal | 99116 |
| 1\_23\_1310301722.cal | 98776 |
| 1\_24\_1311201804.cal | 99045 |
| 1\_25\_1311201806.cal | 99291 |
| 1\_26\_1311201835.cal | 98716 |
| 1\_27\_1311201823.cal | 99106 |
| 1\_28\_1310301813.cal | 98983 |
| 1\_29\_1310301914.cal | 99291 |
| 1\_31\_1310301807.cal | 99013 |
| 1\_32\_1310301831.cal | 99104 |
| 1\_34\_1310301804.cal | 99041 |
| 3\_01\_1401151823.cal | 99262 |
| 3\_02\_1401151905.cal | 98957 |
| 3\_03\_1401151858.cal | 99216 |
| 3\_04\_1401151803.cal | 98979 |
| 3\_05\_1401151752.cal | 99048 |
| 3\_06\_1311201818.cal | 98867 |
| 3\_12\_1401151843.cal | 98988 |
| 3\_14\_1401151837.cal | 99119 |
| 3\_15\_1401151641.cal | 98743 |
| 3\_16\_1401151847.cal | 98990 |
| 3\_17\_1401151814.cal | 99299 |
| 3\_20\_1401151759.cal | 98881 |
| 3\_21\_1401151911.cal | 99225 |
| 3\_22\_1401151828.cal | 99040 |
| 3\_23\_1401151854.cal | 99023 |
| 3\_24\_1401151840.cal | 98707 |
| 3\_25\_1401151908.cal | 99054 |
| 3\_26\_1401151826.cal | 99206 |
| 3\_27\_1401151811.cal | 99046 |
| 3\_28\_1401151747.cal | 98689 |
| 3\_29\_1401151833.cal | 98729 |
| 3\_31\_1401151755.cal | 98963 |
| 3\_32\_1401151808.cal | 99244 |
| 3\_34\_1401151805.cal | 99143 |
|  |  |

### Przeprowadzone pomiary

W tym podrozdziale zostały zaprezentowane otrzymane wyniki, których format zaprezentowano w sekcji [3.3.3](#ssec:exitdata). Dodatkowo do każdego wyniku została przygotowana krótka analiza wraz z zaprezentowaniem dopełniających parametrów wyników, umożliwiających lepszą interpretację rezultatów. Pierwszym elementem jest ukazanie wpływu parametrów wewnętrznych algorytmów na wyniki końcowe. Druga sekcja porównuje czasy trwania algorytmów, wraz z analizą przyczyn różnic w zmierzonych czasach. Następna sekcja opisuje wyniki obciążenia pamięciowego przez zaimplementowane algorytmy wykrywania fiksacji. W przedostatnim podrozdziale zanalizowano metody wprowadzania danych wejściowych do aplikacji. Ostatnia analiza dotyczy dokładności algorytmu uczenia maszynowego. Sekcję [2.4.1](#ssec:algorithms) przeznaczono na porównanie wyników otrzymanych w trzech algorytmach.

#### Analiza wpływu parametrów algorytmów

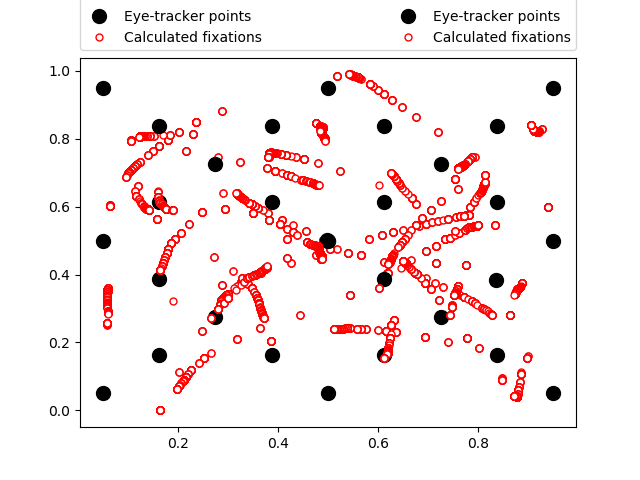
W poniższej sekcji zostanie zaprezentowany wpływ parametrów wewnętrznych algorytmów opisanych w sekcji [2.4.1](#ssec:algorithms). Zbadany zostanie ten wpływ pod kątem różnic w obciązeniu pamięciowym przez algorytmy, porównania ilości elementów wyjściowych oraz czasu trwania algorytmów. Ten podrozdział został podzielony na 3 części, każda prezentująca algorytmy opisane w sekcji [2.4.1](#ssec:algorithms). Uczenie maszynowe posiada czwartą część, precyzję wyników algorytmu. W celu prezentacji rezultatów wykorzystano pierwszych dziesięć plików opisanych w sekcji [4.2](#sec:entryparameters).

##### Algorytm I-VT

Zgodnie z definicją algorytmu I-VT zaprezentowaną w sekcji [2.4.1.1](#ssec:ivt), możemy zdefiniować jeden parametr wewnętrzny, który może wpływać na wyniki końcowe algorytmów. Tym parametrem jest próg prędkości międzypunktowej. W tym podrozdziale został zaprezentowany wpływ tego parametru na wyniki, a także przeprowadzono analizę wyników. Analizę wykonano dla wartości: *v = 0.0001, 0.005, 0.001, 0.01*.

**Liczba wykrytych fiksacji**

W tabeli [[tab:ivtfixationcomparison]](#tab:ivtfixationcomparison) zaprezentowano wynik pomiaru liczby wykrytych fiksacji przeprowadzonej na dziesięciu plikach dla algorytmu I-VT. W pierwszej kolumnie umieszczono nazwę pliku, a w następnych zmierzoną liczbę fiksacji oraz sakad, względem odpowiedniej wartości parametru prędkości. Jak można zauważyć dla tego algorytmu im mniejsza prędkość graniczna, tym więcej fiksacji jest wykrywanych. Jest to spowodowane tym iż, im mniejszy parametr tym mniejsza dokładność filtrowania punktów. Przykład takiej niskiej precyzji w oddzielaniu fiksacji od sakad zaprezentowano na rysunku [4.1](#fig:ivterrorresults). Niedokładność tego wyniku polega na wykrywaniu jako fiksacji ruchu międzypunktowego. Ze względu na tylko jedno porównanie, zaprezentowane w warunku 3 pseudokodu [[lst:ivtpseudocode]](#lst:ivtpseudocode) należy określić dokładną wartość tej prędkości. Można dzięki temu stwierdzić, iż otrzymane wyniki są zgodne z oczekiwaniami.



Błędne przedstawienie punktów dla algorytmu I-VT

Po zaobserwowaniu wyników graficznych w plikach wynikowych stwierdzono, iż parametr prędkości powinien znajdować się w okolicach wartości , gdyż dla tego wyniku zaobserwowano najdokładniejsze wyniki. Ostatnią badaną wartością było , jednak rezultatem takiego parametru było 0 lub 1 wykrytych fiksacji, dlatego zdecydowano się na pominięcie tej kolumny w tabeli [[tab:ivtfixationcomparison]](#tab:ivtfixationcomparison).

Wpływ parametru prędkości granicznej dla algorytmu I-VT, ilość fiksacji

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | 6434 | 91444 | 780 | 3744 | 18015 | 52054 | 513 | 1545 |
|  | 5896 | 92107 | 110 | 2415 | 1011 | 8817 | 37 | 405 |
|  | 5645 | 92313 | 99 | 2027 | 1592 | 7547 | 54 | 446 |
|  | 6792 | 90736 | 101 | 1688 | 1526 | 9217 | 30 | 263 |
|  | 6475 | 91476 | 204 | 2877 | 11015 | 25880 | 85 | 939 |
|  | 6815 | 90960 | 98 | 1889 | 3308 | 8590 | 42 | 288 |
|  | 6726 | 91100 | 118 | 1865 | 6954 | 14511 | 42 | 399 |
|  | 11964 | 82189 | 83 | 2213 | 278 | 6612 | 46 | 641 |
|  | 7737 | 89751 | 87 | 1208 | 12041 | 22008 | 42 | 263 |
|  | 6509 | 91431 | 126 | 1745 | 17453 | 42023 | 48 | 529 |
|  |  |  |  |  |  |  |  |  |

**Analiza czasu trwania algorytmu**

W tabeli [[tab:ivttimecomparison]](#tab:ivttimecomparison) zaprezentowano wyniki pomiaru czasu trwania algorytmu I-VT dla różnych wartości prędkości progowej. Czas trwania zaprezentowano w sekundach. Obserwując otrzymane wyniki możemy stwierdzić, iż czas trwania algorytmu wzrasta odwrotnie proporcjonalnie do ilości wykrytych fiksacji. Zastanawiającym wynikiem jest pomiar czasu dla parametru , ze względu na podobieństwo do pomiarów z pierwszej kolumny. Prawdopodobnie jest to spowodowane niską liczbą elementów koniecznych do wykonania grupowania punktów, ze względu na stosunkowo duży parametr prędkości.

Wpływ parametru prędkości granicznej dla algorytmu I-VT, pomiar czasu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  | **v = 0,01** |
|  | 1.265625 | 4.21875 | 24.234375 | 3.515625 |
|  | 1.203125 | 1.671875 | 4.375 | 1.5625 |
|  | 1.15625 | 1.5625 | 6.203125 | 1.46875 |
|  | 1.578125 | 1.546875 | 6.0625 | 1.5 |
|  | 1.203125 | 1.921875 | 31.765625 | 1.625 |
|  | 1.25 | 1.53125 | 12.3125 | 1.4375 |
|  | 1.359375 | 1.546875 | 24.875 | 1.4375 |
|  | 4.203125 | 1.484375 | 1.90625 | 1.5 |
|  | 1.421875 | 1.421875 | 33.875 | 1.421875 |
|  | 1.140625 | 1.46875 | 34.5625 | 1.453125 |
|  |  |  |  |  |

**Analiza wykorzystania pamięci operacyjnej**

Tabela [[tab:ivtmemorycomparison]](#tab:ivtmemorycomparison) przedstawia wyniki pomiaru wykorzystania pamięci przez algorytm I-VT. Jednostką pomiaru jest MiB. Zgodnie z zaobserwowanymi pomiarami, można stwierdzić, iż dla obecnych maszyn pomiar algorytmem I-VT nie stanowi zbytniego obciążenia, gdyż te pomiary wynoszą setki kilobajtów na cały pomiar, czyli około 90000 plików. Nie zauważono znaczącego wpływu parametru algorytmu na wykorzystanie pamięciowe, bo jak można zaobserwować, wartości w kolumnach nie posiadają specjalnej charakterystyki.

Wpływ parametru prędkości granicznej dla algorytmu I-VT, wykorzystanie pamięci operacyjnej

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| (w MiB) |  |  |  |  |
| **Kontynuacja tabeli** |  |  |  |  |
|  |  |  |  |  |
| (w MiB) |  |  |  |  |
|  |  |  |  | **v = 0,01** |
|  | 0.12500 | 0.17188 | 0.15234 | 0.21094 |
|  | 0.10156 | 0.24609 | 0.16797 | 0.18359 |
|  | 0.16016 | 0.15234 | 0.13281 | 0.12891 |
|  | 0.19531 | 0.17188 | 0.12500 | 0.15625 |
|  | 0.12109 | 0.15625 | 0.16406 | 0.24219 |
|  | 0.06641 | 0.11328 | 0.12500 | 0.25000 |
|  | 0.12500 | 0.22656 | 0.24219 | 0.17188 |
|  | 0.20313 | 0.18750 | 0.21094 | 0.16016 |
|  | 0.12109 | 0.12891 | 0.12891 | 0.21094 |
|  | 0.10156 | 0.19922 | 0.17969 | 0.16797 |
|  |  |  |  |  |

##### Algorytm I-DT

Algorytm I-DT różni się od algorytmu I-VT zaprezentowanego w sekcji [4.3.1.1](#sssec:ivtresults) tym, iż posiada on 2 parametry wewnętrzne: granicę dyspersji oraz czas trwania okna. W tej sekcji wyniki będą badane pod kątem modyfikacji tych dwóch parametrów. W tej sekcji badany będzie wpływ tych dwóch parametrów na otrzymane wyniki końcowe. W pierwszej części sprawdzany będzie wpływ parametru czasu trwania okna w odstępach co 50 ms, rozpoczynając od wartości 100 ms, kończąc na wartości 300 ms. Uznano, iż będzie ukazywać to dobrze różne wartości okna. Wartością dyspersji dla tego badania jest . W drugej części badany będzie wpływ wartości dyspersji na wyniki końcowe. Wartości dyspersji w tej części wynoszą odpowiednio *0.01, 0.05, 0.1*. Punkty znajdujące się poniżej i powyżej tych wartości nie nadają się do dalszej analizy, gdyż dla mniejszych punktów otrzymujemy 0 fiksacji, a dla wartości *1* osiągnięto po 3 fiksacje, przy czasie trwania 1000 sekund na plik. Dla tej części parametr rozmiaru okna wynosi 200 ms.

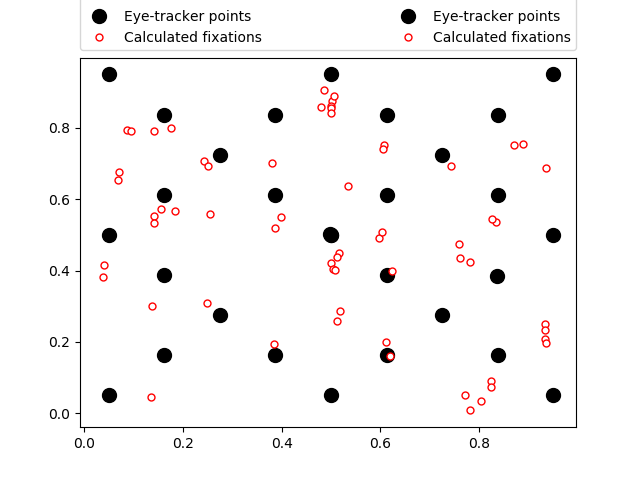
**Liczba wykrytych fiksacji - czas trwania okna**

W tabeli [[tab:idttimefixcomparison]](#tab:idttimefixcomparison) zaprezentowano liczbę wykrytych fiksacji oraz sakad przez algorytm I-DT w zależności od parametru rozmiaru okna. Każda zależność od czasu okna została podzielona na dwie kolumny, odpowiednio fiksacje oraz sakady. Jak można zaobserwować w wynikach, wraz ze wzrostem rozmiaru okna, spada ilość wykrytych fiksacji. Wpływ na to ma umieszczenie punktów w oknie zgodnie z opisem zamieszczonym w rozdziale [2.4.1.2](#ssec:idt). Im większe okno, tym więcej punktów w nim się znajduje, a skoro wyliczona fiksacja jest centroidem tego okna, to wynikiem tego algorytmu jest mniej fiksacji. Również liczba sakad jest zmniejszona od różnicy *wszystkie punkty - fiksacje* ze względu na to, iż więcej punktów znajduje się w oknie i nie jest traktowanych jako sakady.

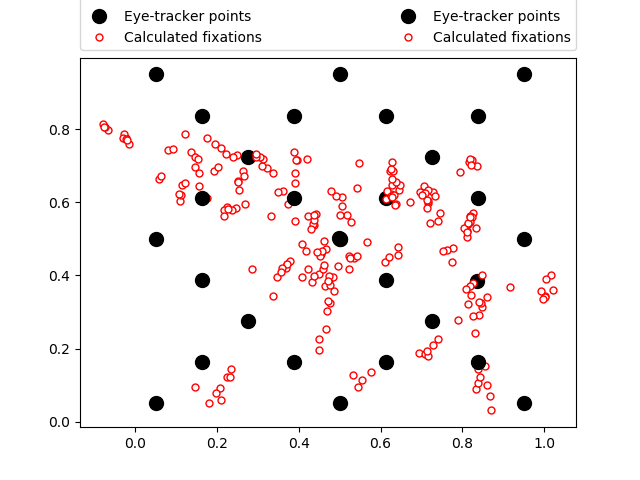
Wpływ parametru czasu okna, algorytm I-DT, wyniki

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| 1\_01\_1311201811.cal | 232 | 41639 | 151 | 51478 | 108 | 62301 | 84 | 70311 | 20 | 91783 |
| 1\_02\_1310301856.cal | 79 | 53260 | 46 | 56741 | 30 | 59877 | 28 | 63884 | 23 | 66419 |
| 1\_03\_1310301904.cal | 133 | 30814 | 98 | 38404 | 85 | 37146 | 73 | 44526 | 70 | 44934 |
| 1\_04\_1310301810.cal | 115 | 39130 | 85 | 38721 | 62 | 42761 | 56 | 48617 | 53 | 47580 |
| 1\_05\_1310301817.cal | 107 | 37975 | 81 | 44105 | 61 | 50329 | 52 | 51646 | 45 | 54772 |
| 1\_06\_1310301837.cal | 84 | 63589 | 58 | 65821 | 39 | 69476 | 33 | 70490 | 31 | 72190 |
| 1\_12\_1310301901.cal | 82 | 45263 | 61 | 46338 | 57 | 43221 | 47 | 48092 | 44 | 50457 |
| 1\_14\_1310301912.cal | 85 | 48028 | 58 | 54627 | 42 | 59476 | 31 | 65319 | 27 | 64586 |
| 1\_15\_1310301840.cal | 144 | 36881 | 116 | 42391 | 95 | 45018 | 83 | 50141 | 70 | 52791 |
| 1\_16\_1310301844.cal | 151 | 27608 | 122 | 32658 | 95 | 38068 | 80 | 40244 | 68 | 46049 |
|  |  |  |  |  |  |  |  |  |  |  |

Na rysunku [4.2](#fig:idtt200) zaprezentowano przykładowy wynik tego algorytmu dla wartości .



Wynik algorytmu I-DT, T = 200



Niepoprawny wynik algorytmu I-DT

**Analiza czasów trwania algorytmu - czas trwania okna**

Tablica [[tab:idttimetimecomparison]](#tab:idttimetimecomparison) zawiera zmierzony czas trwania algorytmu I-DT w zależności od rozmiaru okna. Podczas gdy pierwszy punkt wydaje się być anomalią, jak zaprezentowano na rysunku [4.3](#fig:idtweird), nie znaleziono przyczyny tak niskich pomiarów czasu wykonywania.

Reszta zmierzonego czasu trwania algorytmu oscyluje w okolicach 200-300 sekund, co pozwala określić ten algorytm jako wolny, ale dokładny, ze względu na małą liczbę fiksacji zaprezentowaną w tabeli [[tab:idttimefixcomparison]](#tab:idttimefixcomparison).

Wpływ parametru czasu okna, algorytm I-DT, czas trwania

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | 39.734375 | 35.734375 | 23.1875 | 13.265625 | 3.546875 |
|  | 345.546875 | 354.015625 | 345.328125 | 365.359375 | 349.40625 |
|  | 223.84375 | 216.15625 | 209.59375 | 220.234375 | 189.734375 |
|  | 264.78125 | 243.984375 | 247.3125 | 247.765625 | 228.765625 |
|  | 246.1875 | 254.265625 | 227.453125 | 240.96875 | 231.15625 |
|  | 382.828125 | 371.15625 | 377.578125 | 372.875 | 350.765625 |
|  | 313.765625 | 305.515625 | 291.9375 | 304.3125 | 299.34375 |
|  | 386.15625 | 404.671875 | 404.0 | 407.734375 | 373.71875 |
|  | 220.328125 | 221.0 | 215.703125 | 209.234375 | 195.1875 |
|  | 193.15625 | 186.0625 | 182.734375 | 171.96875 | 166.328125 |
|  |  |  |  |  |  |

**Analiza obciążenia pamięci - czas trwania okna**

W tej sekcji zaprezentowano różnicę w obciążeniu pamięci operacyjnej dla algorytmu I-DT w zależności od parametru rozmiaru czasowego okna. Wynik tego pomiaru umieszczono w tabeli [[tab:idttimetimecomparison]](#tab:idttimetimecomparison). Przeglądając wyniki możemy zaobserwować brak zmian pomiędzy różnymi wynikami, co pozwala nam wywnioskować, iż nie ma żadnej korelacji między parametrem okna, a zużyciem pamięci.

Wpływ parametru czasu okna, algorytm I-DT, zużycie pamięci

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| (w MiB) |  |  |  |  |  |
| **Kontynuacja tabeli** |  |  |  |  |  |
|  |  |  |  |  |  |
| (w MiB) |  |  |  |  |  |
|  |  |  |  |  |  |
|  | 0.07031 | 0.07813 | 0.07422 | 0.07031 | 0.00781 |
|  | 0.13281 | 0.07813 | 0.07422 | 0.08984 | 0.10938 |
|  | 0.21484 | 0.13672 | 0.10938 | 0.13672 | 0.08984 |
|  | 0.11719 | 0.13281 | 0.11719 | 0.10938 | 0.14063 |
|  | 0.14063 | 0.12109 | 0.14063 | 0.24219 | 0.12891 |
|  | 0.16016 | 0.07031 | 0.16406 | 0.08594 | 0.10547 |
|  | 0.16797 | 0.16797 | 0.12891 | 0.16016 | 0.12891 |
|  | 0.11719 | 0.12109 | 0.00000 | 0.12500 | 0.10156 |
|  | 0.12109 | 0.08984 | 0.10156 | 0.09375 | 0.15625 |
|  | 0.12891 | 0.09766 | 0.20703 | 0.11719 | 0.10156 |
|  |  |  |  |  |  |

**Liczba wykrytych fiksacji - dyspersja**

W tabeli [[tab:idtdispfixcomparison]](#tab:idtdispfixcomparison) zaprezentowano wyniki algorytmu I-DT w zależności od wartości parametru dyspersji. Analogicznie do innych tabel dotyczących fiksacji i sakad przedstawionych w tym rozdziale wewnętrzne komórki zostały podzielone na wykryte fiksacje oraz sakady. Odczytując wyniki można wywnioskować, iż zmniejszanie parametru dyspersji powoduje zwiększenie ilości wykrytych fiksacji. Jest to spowodowane porównywaniem wartości dyspersji wewnątrz okna, więc logicznym następstwem tego rozwiązania jest, im mniejszy wymagany parametr dyspersji, tym więcej punktów można wykryć jako fiksacje. Wyniki dla pliku 1 i 10 zostaną przeanalizowane dla tabeli [[tab:idtdispmemorycomparison]](#tab:idtdispmemorycomparison).

Wpływ parametru dyspersji, algorytm I-DT, wyniki

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 1\_01\_1311201811.cal | 134 | 50061 | 108 | 62301 | 0 | 99129 |
| 1\_02\_1310301856.cal | 22 | 71070 | 30 | 59877 | 137 | 53162 |
| 1\_03\_1310301904.cal | 36 | 63871 | 85 | 37146 | 43 | 88033 |
| 1\_04\_1310301810.cal | 39 | 59185 | 62 | 42761 | 91 | 70730 |
| 1\_05\_1310301817.cal | 39 | 57755 | 61 | 50329 | 34 | 89663 |
| 1\_06\_1310301837.cal | 28 | 81731 | 39 | 69476 | 129 | 59483 |
| 1\_12\_1310301901.cal | 18 | 77708 | 57 | 43221 | 79 | 77443 |
| 1\_14\_1310301912.cal | 23 | 87308 | 42 | 59476 | 130 | 50360 |
| 1\_15\_1310301840.cal | 28 | 68066 | 95 | 45018 | 18 | 94418 |
| 1\_16\_1310301844.cal | 49 | 55464 | 95 | 38068 | 2 | 98779 |
|  |  |  |  |  |  |  |

**Analiza obciążenia pamięci - dyspersja**

Tabela [[tab:idtdispmemorycomparison]](#tab:idtdispmemorycomparison) prezentuje otrzymane wyniki pomiaru wykorzystania pamięci przez algorytm I-DT, mając na uwadze różnice w parametrze dyspersji. Przeglądając wyniki, możemy zauważyć lekki spadek w otrzymanych wartościach pomiędzy drugą a trzecią kolumną. Pomimo zaobserwowanego zwiększenia liczby otrzymanych fiksacji wyświetlonego w tabeli [[tab:idtdispfixcomparison]](#tab:idtdispfixcomparison), możemy zauważyć spadek w wykorzystaniu pamięciowym. Dzięki temu, można określić dla niektóych wartości, iż parametr dyspersji ma wpływ na zużycie pamięci operacyjnej. Wniosek dotyczący porównywania wartości dyspersji wewnątrz okna zaprezentowany przy okazji analizy tabeli [[tab:idtdispfixcomparison]](#tab:idtdispfixcomparison) można również zastosować do tabeli [[tab:idtdispmemorycomparison]](#tab:idtdispmemorycomparison).

Wpływ parametru dyspersji, algorytm I-DT, obciązenie pamięciowe

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| (w MiB) |  |  |  |
| **Kontynuacja tabeli** |  |  |  |
|  |  |  |  |
| (w MiB) |  |  |  |
|  |  |  |  |
| 1\_01\_1311201811.cal | 0,07031 | 0,07422 | 0,00000 |
| 1\_02\_1310301856.cal | 0,13281 | 0,07422 | 0,06641 |
| 1\_03\_1310301904.cal | 0,21484 | 0,10938 | 0,00391 |
| 1\_04\_1310301810.cal | 0,11719 | 0,11719 | 0,05078 |
| 1\_05\_1310301817.cal | 0,14063 | 0,14063 | 0,01563 |
| 1\_06\_1310301837.cal | 0,16016 | 0,16406 | 0,06641 |
| 1\_12\_1310301901.cal | 0,16797 | 0,12891 | 0,06641 |
| 1\_14\_1310301912.cal | 0,11719 | 0,00000 | 0,08984 |
| 1\_15\_1310301840.cal | 0,12109 | 0,10156 | 0,01563 |
| 1\_16\_1310301844.cal | 0,12891 | 0,20703 | 0,00000 |
|  |  |  |  |

**Analiza czasów trwania algorytmu - dyspersja**

Czasy trwania algorytmu I-DT dla różnych wartości dyspersji zaprezentowano w tabeli [[tab:idtdisptimecomparison]](#tab:idtdisptimecomparison). Analizując otrzymane wartości, można wywnioskować, iż w parze z mniejszą wartością dyspersji idzie zmniejszenie czasu trwania algorytmu. Potwierdza to wstęp do tej analizy, gdzie dla wartości , zmierzono czas około 1000 sekund. Mniejsza ilość porównań jest powodem zależności dyspersji od czasu.

Wpływ parametru dyspersji, algorytm I-DT, czas trwania

|  |  |  |  |
| --- | --- | --- | --- |
| **Table  continued from previous page** |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 1\_01\_1311201811.cal | 35.296875 | 23.1875 | 1.421875 |
| 1\_02\_1310301856.cal | 432.015625 | 345.328125 | 30.125 |
| 1\_03\_1310301904.cal | 501.46875 | 209.59375 | 4.53125 |
| 1\_04\_1310301810.cal | 397.765625 | 247.3125 | 17.125 |
| 1\_05\_1310301817.cal | 376.671875 | 227.453125 | 5.15625 |
| 1\_06\_1310301837.cal | 544.515625 | 377.578125 | 21.296875 |
| 1\_12\_1310301901.cal | 575.140625 | 291.9375 | 9.3125 |
| 1\_14\_1310301912.cal | 636.828125 | 404.0 | 43.46875 |
| 1\_15\_1310301840.cal | 491.1875 | 215.703125 | 3.4375 |
| 1\_16\_1310301844.cal | 356.453125 | 182.734375 | 1.34375 |
|  |  |  |  |

##### Algorytm ML - wpływ parametru prędkości

Ze względu na konieczność określenia klasy przynależności dla punktu, tzn. czy należy on do fiksacji, należało wyznaczyć fiksację dowolnym algorytmem. Ze względu na prostotę obliczeń oraz małą złożoność czasową, zdecydowano się na wstępną analizę algorytmem I-VT. Analizę bezpośredniego wpływu parametru wewnętrznego na algorytm I-VT zaprezentowano w sekcji [4.3.1.1](#sssec:ivtresults). Wyniki są podane dla parametru podziału danych: 50% danych testowych, 50% danych treningowych.

**Liczba wykrytych fiksacji**

Jak zaprezentowano w tabeli [[tab:mlivtfixcomparison]](#tab:mlivtfixcomparison), otrzymane wyniki pozwalają stwierdzić, iż parametr progowy prędkości międzypunktowej nie wpływa znacząco na wyniki końcowe algorytmu. Ciekawym efektem ubocznym zmiany parametru, zaobserwowanym w ostatnich kolumnach tabeli [[tab:mlivtfixcomparison]](#tab:mlivtfixcomparison) jest brak otrzymanych wyników końcowych ze względu na otrzymany wyjątek w trakcie tworzenia listy punktów końcowych. Także zaobserwowano znaczący przyrost liczby fiksacji względem sakad, prawdopodobnie jest to spowodowane zbyt małą próbką danych typu *"punkt należy do fiksacji"*, co przedstawiono w ostatnich kolumnach tabeli [[tab:ivtfixationcomparison]](#tab:ivtfixationcomparison).

Wpływ parametru prędkości granicznej dla algorytmu uczenia maszynowego, liczba fiksacji

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | 3809 | 95319 | 3800 | 95328 | 3801 | 95327 |  |  |
|  | 3336 | 95835 | 3345 | 95826 | 3433 | 95738 | 97942 | 1229 |
|  | 3213 | 95993 | 3265 | 95941 | 3448 | 95758 |  |  |
|  | 3449 | 95699 | 3632 | 95516 | 4128 | 95020 | 98261 | 887 |
|  | 3834 | 95375 | 3826 | 95383 | 3805 | 95404 | 97756 | 1453 |
|  | 4039 | 95141 | 4027 | 95153 | 4030 | 95150 |  |  |
|  | 3897 | 95127 |  |  | 3909 | 95115 |  |  |
|  | 3166 | 96008 | 3241 | 95933 | 8533 | 90641 |  |  |
|  | 4196 | 94913 | 4245 | 94864 | 4605 | 94504 | 98476 | 633 |
|  | 3800 | 95389 | 3863 | 95326 | 3836 | 95353 | 98258 | 931 |
|  |  |  |  |  |  |  |  |  |

**Analiza czasu trwania algorytmu**

Tabela [[tab:mlivttimecomparison]](#tab:mlivttimecomparison) prezentuje otrzymane wyniki pomiaru czasu ze względu na prędkość graniczną. Obserwując wyniki, możemy wywnioskować, iż parametr progu prędkości ma wpływ na czas trwania algorytmu. Jest to spowodowane większą ilością przeprowadzonych porównań względem tablicy elementów posiadających fiksację. Również prawdopodobną przyczyną takiej zależności jest konieczność dokładnego podzielenia wyników na dane testowe i treningowe.

Wpływ parametru prędkości granicznej dla algorytmu uczenia maszynowego, czas trwania

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  | **v = 0,005** |
|  | 1.921875 | 1.9375 | 1.96875 |  |
|  | 1.953125 | 1.921875 | 1.9375 | 2.265625 |
|  | 1.9375 | 1.90625 | 1.953125 |  |
|  | 1.9375 | 1.921875 | 2.015625 | 2.265625 |
|  | 1.921875 | 1.921875 | 1.9375 | 2.203125 |
|  | 1.9375 | 1.9375 | 1.953125 |  |
|  | 1.921875 |  | 1.96875 |  |
|  | 1.921875 | 1.921875 | 1.953125 |  |
|  | 1.953125 | 1.921875 | 1.9375 | 2.171875 |
|  | 2.03125 | 1.9375 | 1.96875 | 2.171875 |
|  |  |  |  |  |

**Analiza wykorzystania pamięci operacyjnej**

Wyniki pomiaru zużycia pamięci operacyjnej dla uczenia maszynowego przedstawiono w tabeli [[tab:ivtmemorycomparison]](#tab:ivtmemorycomparison). Porównywalnie do wyników otrzymanych w tabeli [[tab:mlivtfixcomparison]](#tab:mlivtfixcomparison), możemy zaobserwować brak wpływu parametru progowego na rezultaty, poza czwartą kolumną. Dla plików numer 2 i 4 możemy zaobserwować znaczący wzrost zużycia pamięci operacyjnej. Jest to wynikiem zwiększonej liczby porównań.

Wpływ parametru prędkości granicznej dla algorytmu uczenia maszynowego, wykorzystanie pamięci operacyjnej

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| (w MiB) |  |  |  |  |
| **Kontynuacja tabeli** |  |  |  |  |
|  |  |  |  |  |
| (w MiB) |  |  |  |  |
|  |  |  |  | **v = 0,005** |
|  | 8,36719 | 7.12891 | 8,12891 |  |
|  | 7,64063 | 7,38672 | 7,79688 | 11,36719 |
|  | 7,80859 | 8,37891 | 6,64063 |  |
|  | 8,02344 | 7,90625 | 8,53516 | 11,14844 |
|  | 6,93750 | 6,89063 | 7,50391 | 7,62500 |
|  | 6,91797 | 7,85547 | 7,69141 |  |
|  | 7,21094 |  | 8,55859 |  |
|  | 7,09766 | 7,79297 | 7,28516 |  |
|  | 7,86328 | 7,62891 | 7,16016 | 6,72266 |
|  | 7,59766 | 6,07813 | 7,18359 | 9,64453 |
|  |  |  |  |  |

**Analiza precyzji algorytmu**

Jedną ze statystyk wynikowych dla uczenia maszynowego jest precyzja danych wynikowych względem rzeczywistych, poprzez obliczenie następującej wartości:

Wyniki takich obliczeń zaprezentowano w tabeli [[tab:mlivtprecisioncomparison]](#tab:mlivtprecisioncomparison). Takie wyniki można zinterpretować jako weryfikację odpowiedniego przygotowania modelu. W wypadku otrzymanych wyników, możemy stwierdzić, że utworzony model został przygotowany dobrze. Wyniki poza jednym rekordem nie osiągają wartości niższych niż 90% precyzji, co jest bardzo dobrym wynikiem.

Wpływ parametru prędkości granicznej dla algorytmu uczenia maszynowego, precyzja algorytmu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  | **v = 0,005** |
|  | 92,36% | 92,35% | 92,35% |  |
|  | 93,29% | 93,31% | 92,86% | 97,55% |
|  | 93,64% | 93,74% | 93,22% |  |
|  | 92,94% | 92,85% | 91,53% | 98,32% |
|  | 92,33% | 92,31% | 92,26% | 97,07% |
|  | 91,72% | 91,70% | 91,71% |  |
|  | 92,04% |  | 92,06% |  |
|  | 93,50% | 93,65% | 83,11% |  |
|  | 91,66% | 91,75% | 90,58% | 98,78% |
|  | 92,19% | 92,31% | 92,26% | 98,30% |
|  |  |  |  |  |

##### Algorytm ML - wpływ podziału danych testowych i danych treningowych

Sama techonologia uczenia maszynowego pozwala na modyfikację proporcji danych testowych względem danych treningowych. Celem tej sekcji jest prezentacja otrzymanych wyników przy zastosowaniu różnych wartości tej proporcji. By obliczyć istniejące fiksacje, w celu stworzenia poprawnego modelu danych użyto parametru prędkości granicznej *v = 0,0001*, pobrany on został z danych przedstawionych w sekcji [4.3.1.3](#sssec:mlivt). Różnica w tych danych będzie badana przy stopniowym modyfikowaniu różnicy w proporcjach o 15%, tzn. 20%-35%-50% dla danych testowych, resztę danych wydzielono na trening modelu.

**Liczba wykrytych fiksacji/sakad**

W tabeli [[tab:mlpropfixcomparison]](#tab:mlpropfixcomparison) przedstawiono wyniki pomiaru liczności wykrytych fiksacji przez algorytm uczenia maszynowego oraz różnice względem zmian parametru proporcji danych na ten wynik. Jak można zaobserwować po wynikach, im mniej danych jest przeznaczonych na trening modelu, tym zmniejsza się liczba wykrytych fiksacji. Jest to zgodne z założeniami uczenia maszynowego, gdzie średnimi wartościami jest podział 30-70%.

Wpływ parametru podziału zbiorów dla algorytmu uczenia maszynowego, liczba fiksacji

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | 6118 | 93010 | 4948 | 94180 | 3749 | 95379 |
|  | 5534 | 93637 | 4582 | 94589 | 3511 | 95660 |
|  | 5419 | 93787 | 4475 | 94731 | 3382 | 95824 |
|  | 6662 | 92486 | 5466 | 93682 | 4168 | 94980 |
|  | 6116 | 93093 | 4928 | 94281 | 3780 | 95429 |
|  | 6501 | 92679 | 5253 | 93927 | 4084 | 95096 |
|  | 6289 | 92735 | 5079 | 93945 | 3904 | 95120 |
|  | 13505 | 85669 | 11012 | 88162 | 8498 | 90676 |
|  | 7415 | 91694 | 6030 | 93079 | 4635 | 94474 |
|  | 6158 | 93031 | 5045 | 94144 | 3957 | 95232 |
|  |  |  |  |  |  |  |

**Analiza czasu trwania algorytmu**

Celem tego paragrafu jest zaprezentowanie wyników pomiaru czasu trwania algorytmu uczenia maszynowego oraz analiza różnic w rezultatach względem modyfikacji podziału danych. Wyniki zaprezentowano w tabeli [[tab:mlproptimecomparison]](#tab:mlproptimecomparison). Jak zaobserwowano w tabeli, wyniki są porównywalne do tych otrzymanych w tabeli [[tab:mlivttimecomparison]](#tab:mlivttimecomparison), co jest spowodowane doborem parametru prędkości granicznej. Nie zauważono innych różnic w pomiarze czasu, przez co wnioskiem tej analizy może być, iż parametr proporcji nie ma wpływu na czas trwania algorytmu.

Wpływ parametru podziału zbiorów dla algorytmu uczenia maszynowego, czas trwania

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| (w s) |  |  |  |
| **Kontynuacja tabeli** |  |  |  |
|  |  |  |  |
| (w s) |  |  |  |
|  |  |  |  |
|  | 2.015625 | 1.953125 | 1.953125 |
|  | 1.9375 | 1.9375 | 1.9375 |
|  | 1.953125 | 1.96875 | 1.9375 |
|  | 1.9375 | 1.96875 | 2.015625 |
|  | 1.96875 | 1.953125 | 1.953125 |
|  | 1.9375 | 1.953125 | 1.953125 |
|  | 1.96875 | 1.9375 | 1.953125 |
|  | 1.9375 | 1.984375 | 1.984375 |
|  | 1.90625 | 1.96875 | 1.96875 |
|  | 1.921875 | 1.96875 | 1.96875 |
|  |  |  |  |

**Analiza wykorzystania pamięci operacyjnej**

Wyniki pomiaru obciążenia pamięci wykorzystując uczenie maszynowe przy zmianie parametru proporcji danych testowych zaprezentowano w tabeli [[tab:mlpropmemorycomparison]](#tab:mlpropmemorycomparison). Zgodnie z tymi wynikami, możemy zauważyć, iż są one podobne do wyników zaprezentowanych w tabeli [[tab:mlivtmemorycomparison]](#tab:mlivtmemorycomparison). Oznacza to, iż zmiana tego parametru nie ma wpływu na obciążenie pamięci dla wyników.

Wpływ parametru podziału zbiorów dla algorytmu uczenia maszynowego, zużycie pamięci

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| (w MiB) |  |  |  |
| **Kontynuacja tabeli** |  |  |  |
|  |  |  |  |
| (w MiB) |  |  |  |
|  |  |  |  |
|  | 7,42578 | 7,24609 | 7,65234 |
|  | 7,50391 | 8,21094 | 7,78516 |
|  | 8,50391 | 7,22266 | 7,39453 |
|  | 7,49609 | 7,57422 | 8,27734 |
|  | 8,07422 | 7,32813 | 7,89453 |
|  | 6,59375 | 7,24219 | 7,97656 |
|  | 6,48438 | 7,49219 | 7,57422 |
|  | 7,01953 | 8,26563 | 8,05469 |
|  | 6,71094 | 7,26172 | 7,05859 |
|  | 6,85547 | 7,16797 | 7,07813 |
|  |  |  |  |

**Precyzja algorytmu**

Zgodnie z wynikami porównania liczby wykrytych fiksacji dla zmian proporcji danych testowych względem danych treningowych zaprezentowanymi w tabeli [[tab:mlpropprecisioncomparison]](#tab:mlpropprecisioncomparison), możemy zauważyć iż nie ma to wpływu na wyniki końcowe precyzji. Jest to prawdopodobnie spowodowane brakiem zmian w liczbie elementów należących do klasy przy tworzeniu modelu.

Wpływ parametru podziału zbiorów dla algorytmu uczenia maszynowego, precyzja algorytmu

|  |  |  |  |
| --- | --- | --- | --- |
| **Kontynuacja tabeli** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | 92.55% | 92.37% | 92.24% |
|  | 92.73% | 93.11% | 93.02% |
|  | 92.99% | 93.28% | 93.09% |
|  | 91.60% | 91.75% | 91.61% |
|  | 92.30% | 92.18% | 92.21% |
|  | 91.72% | 91.67% | 91.82% |
|  | 92.17% | 92.04% | 92.05% |
|  | 82.85% | 83.02% | 83.04% |
|  | 90.62% | 90.65% | 90.64% |
|  | 92.36% | 92.42% | 92.50% |
|  |  |  |  |

Jak przedstawiono w tabelach powyżej, wpływ parametru proporcji podziału danych testowych i treningowych jest znikomy na czas trwania, precyzję i obciążenie pamięciowe. Wartość tego parametru ma wpływ na wynik końcowy algorytmu, jednak różnica pomiędzy danymi nie jest tak istotna jak wyniki zaprezentowane w tabeli [[tab:mlivtfixcomparison]](#tab:mlivtfixcomparison). Jak wywnioskowano w paragrafach powyżej, wpływ parametru proporcji jest znikomy na wynik końcowy algorytmu uczenia maszynowego, co może oznaczać dobrze wybrany typ algorytmu uczenia maszynowego. Większy wpływ na wyniki końcowe dla algorytmu ML ma zmiana parametru prędkości granicznej, zgodnie z zaprezentowanymi wynikami w sekcji [4.3.1.3](#sssec:mlivt).

#### Analiza metod wprowadzania danych wejściowych

Celem poniższej sekcji jest prezentacja otrzymanych pomiarów czasów pobrania danych wejściowych oraz ich konwersji na format czytelny dla aplikacji. Pierwsza kolumna w tabelach [[tab:fileimport]](#tab:fileimport) i [[tab:importdb]](#tab:importdb) prezentuje nazwę pliku, a w drugiej kolumnie znajdziemy wartości wykonanego pomiaru podane w sekundach.

W tabeli [[tab:fileimport]](#tab:fileimport) zaprezentowano wynik pomiaru importu danych oraz konwersji na format czytelny dla aplikacji dla plików pobieranych z dysku twardego. Dla otrzymanych wyników obliczono następujące wartości statystyczne: łączny czas trwania importu danych wynosi 65,673 sekund, średni czas wynosi 1,3681875 sekundy, odchylenie standardowe równa się 0,03755 w przybliżeniu do piątego miejsca po przecinku, a mediana wynosi 1,359 sekundy. Elementy maksymalne i minimalne wynoszą odpowiednio: 1,438 i 1,297 sekundy. Zgodnie z wartościami pokazanymi w sekcji [4.2](#sec:entryparameters), oznacza to iż średnio 65734 elementów jest przetwarzanych na sekundę.

Czas wczytywania plików oraz konwersji z pliku .cal

|  |  |
| --- | --- |
| **Kontynuacja tabeli  z poprzedniej strony** |  |
|  |  |
| 1\_01\_1311201811.cal | 1.344 |
| 1\_02\_1310301856.cal | 1.328 |
| 1\_03\_1310301904.cal | 1.375 |
| 1\_04\_1310301810.cal | 1.312 |
| 1\_05\_1310301817.cal | 1.344 |
| 1\_06\_1310301837.cal | 1.328 |
| 1\_12\_1310301901.cal | 1.297 |
| 1\_14\_1310301912.cal | 1.297 |
| 1\_15\_1310301840.cal | 1.312 |
| 1\_16\_1310301844.cal | 1.359 |
| 1\_17\_1310301825.cal | 1.391 |
| 1\_20\_1310301822.cal | 1.438 |
| 1\_21\_1310301833.cal | 1.359 |
| 1\_22\_1310301908.cal | 1.359 |
| 1\_23\_1310301722.cal | 1.344 |
| 1\_24\_1311201804.cal | 1.328 |
| 1\_25\_1311201806.cal | 1.344 |
| 1\_26\_1311201835.cal | 1.344 |
| 1\_27\_1311201823.cal | 1.359 |
| 1\_28\_1310301813.cal | 1.391 |
| 1\_29\_1310301914.cal | 1.375 |
| 1\_31\_1310301807.cal | 1.438 |
| 1\_32\_1310301831.cal | 1.406 |
| 1\_34\_1310301804.cal | 1.391 |
| 3\_01\_1401151823.cal | 1.406 |
| 3\_02\_1401151905.cal | 1.438 |
| 3\_03\_1401151858.cal | 1.344 |
| 3\_04\_1401151803.cal | 1.344 |
| 3\_05\_1401151752.cal | 1.359 |
| 3\_06\_1311201818.cal | 1.406 |
| 3\_12\_1401151843.cal | 1.391 |
| 3\_14\_1401151837.cal | 1.375 |
| 3\_15\_1401151641.cal | 1.312 |
| 3\_16\_1401151847.cal | 1.391 |
| 3\_17\_1401151814.cal | 1.375 |
| 3\_20\_1401151759.cal | 1.359 |
| 3\_21\_1401151911.cal | 1.344 |
| 3\_22\_1401151828.cal | 1.422 |
| 3\_23\_1401151854.cal | 1.344 |
| 3\_24\_1401151840.cal | 1.344 |
| 3\_25\_1401151908.cal | 1.406 |
| 3\_26\_1401151826.cal | 1.422 |
| 3\_27\_1401151811.cal | 1.406 |
| 3\_28\_1401151747.cal | 1.359 |
| 3\_29\_1401151833.cal | 1.344 |
| 3\_31\_1401151755.cal | 1.375 |
| 3\_32\_1401151808.cal | 1.422 |
| 3\_34\_1401151805.cal | 1.422 |
|  |  |

Tabela [[tab:importdb]](#tab:importdb) przedstawia odpowiadające wyniki dla pobierania elementów z bazy danych. Łączny czas tej operacji dla wszystkich 48 plików wynosi 118,395 sekund. Średni czas importu równa się 2,4666 sekundy, odchylenie standardowe osiąga wartość 0,06259 w przybliżeniu do piątego miejsca po przecinku. Mediana wynosi 2,453 sekundy, wartość maksymalna 2,594 sekundy, a minimalna 2,344. Prędkość wykonywania operacji wynosi około 36488 elementów na sekundę.

Czas importu danych oraz konwersji z bazy danych

|  |  |
| --- | --- |
| **Kontynuacja tabeli  z poprzedniej strony** |  |
|  |  |
| 1\_01\_1311201811.cal | 2.438 |
| 1\_02\_1310301856.cal | 2.422 |
| 1\_03\_1310301904.cal | 2.344 |
| 1\_04\_1310301810.cal | 2.391 |
| 1\_05\_1310301817.cal | 2.406 |
| 1\_06\_1310301837.cal | 2.375 |
| 1\_12\_1310301901.cal | 2.359 |
| 1\_14\_1310301912.cal | 2.344 |
| 1\_15\_1310301840.cal | 2.359 |
| 1\_16\_1310301844.cal | 2.547 |
| 1\_17\_1310301825.cal | 2.594 |
| 1\_20\_1310301822.cal | 2.547 |
| 1\_21\_1310301833.cal | 2.438 |
| 1\_22\_1310301908.cal | 2.516 |
| 1\_23\_1310301722.cal | 2.422 |
| 1\_24\_1311201804.cal | 2.391 |
| 1\_25\_1311201806.cal | 2.422 |
| 1\_26\_1311201835.cal | 2.469 |
| 1\_27\_1311201823.cal | 2.469 |
| 1\_28\_1310301813.cal | 2.438 |
| 1\_29\_1310301914.cal | 2.562 |
| 1\_31\_1310301807.cal | 2.562 |
| 1\_32\_1310301831.cal | 2.547 |
| 1\_34\_1310301804.cal | 2.531 |
| 3\_01\_1401151823.cal | 2.500 |
| 3\_02\_1401151905.cal | 2.422 |
| 3\_03\_1401151858.cal | 2.453 |
| 3\_04\_1401151803.cal | 2.422 |
| 3\_05\_1401151752.cal | 2.531 |
| 3\_06\_1311201818.cal | 2.516 |
| 3\_12\_1401151843.cal | 2.453 |
| 3\_14\_1401151837.cal | 2.453 |
| 3\_15\_1401151641.cal | 2.438 |
| 3\_16\_1401151847.cal | 2.516 |
| 3\_17\_1401151814.cal | 2.500 |
| 3\_20\_1401151759.cal | 2.453 |
| 3\_21\_1401151911.cal | 2.500 |
| 3\_22\_1401151828.cal | 2.438 |
| 3\_23\_1401151854.cal | 2.469 |
| 3\_24\_1401151840.cal | 2.438 |
| 3\_25\_1401151908.cal | 2.500 |
| 3\_26\_1401151826.cal | 2.438 |
| 3\_27\_1401151811.cal | 2.469 |
| 3\_28\_1401151747.cal | 2.547 |
| 3\_29\_1401151833.cal | 2.453 |
| 3\_31\_1401151755.cal | 2.500 |
| 3\_32\_1401151808.cal | 2.562 |
| 3\_34\_1401151805.cal | 2.531 |
|  |  |

Jak można zauważyć na powyższych wynikach, czas importu wartości z bazy danych wynosi około sekundę dłużej niż przy pobraniu z dysku twardego. Obliczone wartości odchylenia standardowego przy tych metodach wskazują na to, iż są one dość stabilne, gdyż ich odchylenie nie przekracza wartości 0,1 s, co dla operacji na takiej ilości danych powoduje zbyt dużego opóźnienia. Pomimo umieszczenia bazy danych na dysku SSD, czasy odczytu danych nie poprawiły się. Po umieszczeniu timera pomiędzy metodami inicjalizującymi połączenie z bazą danych stwierdzono, iż zajmuje ono około 0,65 sekundy, więc jest ono głównym powodem większych wartości wynikowych dla importu z bazy danych.

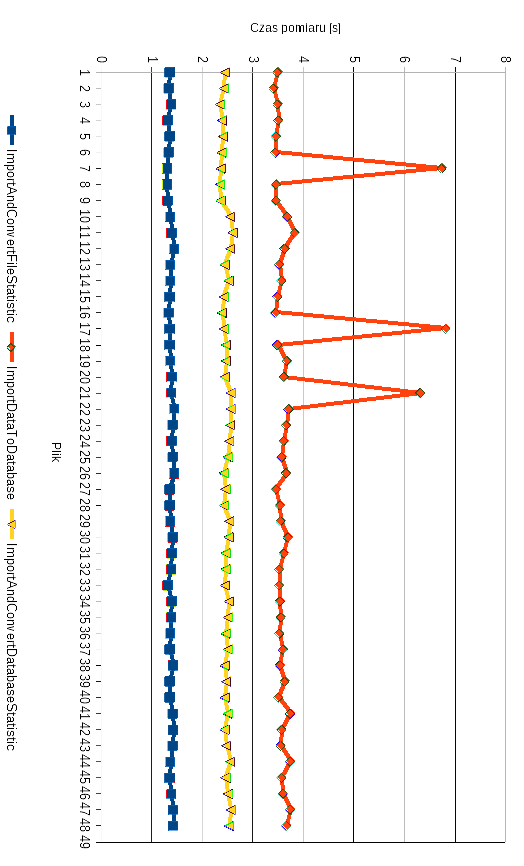
W tabeli [[tab:insertdb]](#tab:insertdb) zaprezentowano wyniki pomiaru operacji eksportu danych do bazy danych. Zostało to wykonane celem weryfikacji wydajności bazy danych. Łączny czas operacji dla wszystkich plików wyniósł 180,432 sekundy, średnia otrzymana wartość to 3,759 sekund, odchylenie standardowe 0,74628 sekund, mediana 3,562 sekund, maksimum prawie 7 sekund a minimum 3,4 sekundy.

Pomiar czasu umieszczania elementów w bazie danych

|  |  |
| --- | --- |
| **Kontynuacja tabeli** |  |
|  |  |
| 1\_01\_1311201811.cal | 3,484 |
| 1\_02\_1310301856.cal | 3,406 |
| 1\_03\_1310301904.cal | 3,484 |
| 1\_04\_1310301810.cal | 3,5 |
| 1\_05\_1310301817.cal | 3,453 |
| 1\_06\_1310301837.cal | 3,438 |
| 1\_12\_1310301901.cal | 6,734 |
| 1\_14\_1310301912.cal | 3,453 |
| 1\_15\_1310301840.cal | 3,453 |
| 1\_16\_1310301844.cal | 3,672 |
| 1\_17\_1310301825.cal | 3,828 |
| 1\_20\_1310301822.cal | 3,625 |
| 1\_21\_1310301833.cal | 3,516 |
| 1\_22\_1310301908.cal | 3,562 |
| 1\_23\_1310301722.cal | 3,484 |
| 1\_24\_1311201804.cal | 3,438 |
| 1\_25\_1311201806.cal | 6,812 |
| 1\_26\_1311201835.cal | 3,484 |
| 1\_27\_1311201823.cal | 3,672 |
| 1\_28\_1310301813.cal | 3,609 |
| 1\_29\_1310301914.cal | 6,312 |
| 1\_31\_1310301807.cal | 3,703 |
| 1\_32\_1310301831.cal | 3,656 |
| 1\_34\_1310301804.cal | 3,609 |
| 3\_01\_1401151823.cal | 3,562 |
| 3\_02\_1401151905.cal | 3,656 |
| 3\_03\_1401151858.cal | 3,453 |
| 3\_04\_1401151803.cal | 3,531 |
| 3\_05\_1401151752.cal | 3,547 |
| 3\_06\_1311201818.cal | 3,688 |
| 3\_12\_1401151843.cal | 3,609 |
| 3\_14\_1401151837.cal | 3,516 |
| 3\_15\_1401151641.cal | 3,516 |
| 3\_16\_1401151847.cal | 3,531 |
| 3\_17\_1401151814.cal | 3,547 |
| 3\_20\_1401151759.cal | 3,516 |
| 3\_21\_1401151911.cal | 3,594 |
| 3\_22\_1401151828.cal | 3,531 |
| 3\_23\_1401151854.cal | 3,625 |
| 3\_24\_1401151840.cal | 3,5 |
| 3\_25\_1401151908.cal | 3,734 |
| 3\_26\_1401151826.cal | 3,562 |
| 3\_27\_1401151811.cal | 3,547 |
| 3\_28\_1401151747.cal | 3,734 |
| 3\_29\_1401151833.cal | 3,562 |
| 3\_31\_1401151755.cal | 3,594 |
| 3\_32\_1401151808.cal | 3,734 |
| 3\_34\_1401151805.cal | 3,656 |
|  |  |

Zastanawiającym wynikiem jest wynik 7 sekund przy zapisie do bazy danych, po weryfikacji prawdopodobną przyczyną takich pomiarów jest chwilowa niedyspozycyjność połączenia z serwerem, jednak, jako iż tylko 2 operacje osiągnęły taki wynik, może to być też wynikiem przypadku, bo nie znaleziono żadnych anomalii w danych przeszukując ręcznie te dwa pliki.  
Łączny średni czas połączonej operacji odczytu i zapisu na bazie danych wynosi około 6,22 sekund, co dla ilości danych zaprezentowanych w sekcji [4.2](#sec:entryparameters) jest bardzo dobrym wynikiem. Powodem dużej prędkości takich operacji również jest mała złożoność danych, gdyż dostarczone dane są umieszczane tylko w jednej tabeli posiadającej pięć elementów.

Na rysunku [4.4](#fig:importdata) zaprezentowano wykres przedstawiający pomiar czasu powyższej analizy.



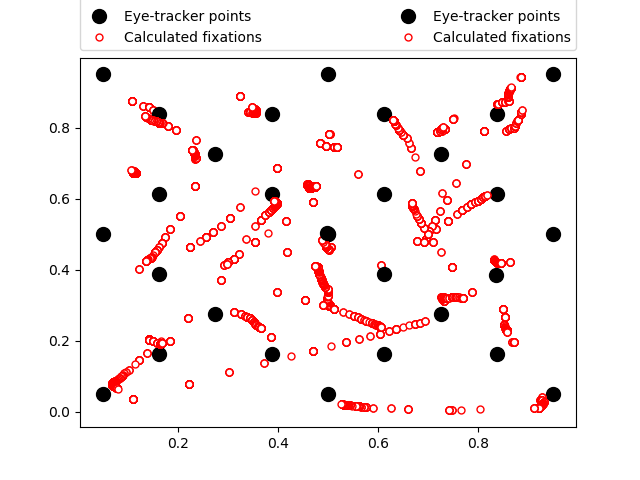
Przedstawienie wyników pomiaru czasu obsługi danych

#### Porównanie algorytmów

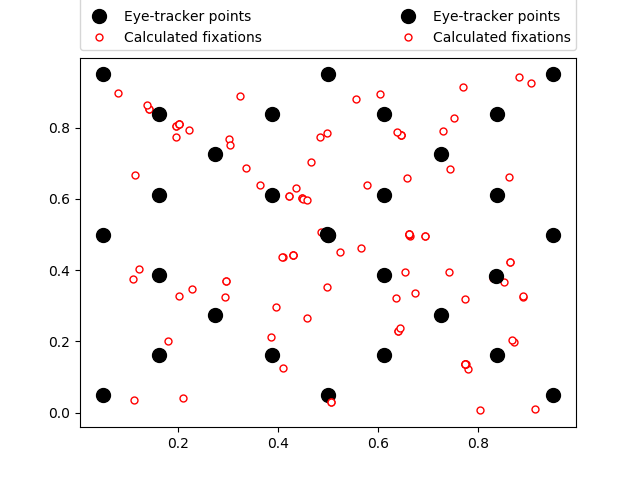
Obserwując wyniki algorytmów zaprezentowane w sekcji [4.3.1](#ssec:queryparameters) można stwierdzić, że wyniki końcowe zależą dużo od wartości określonych w parametrach wewnętrznych. Niepoprawna wartość prędkości granicznej dla algorytmu I-VT, czy też wartości dyspersji i rozmiaru okna czasowego dla algorytmu I-DT może powodować zbyt duże nagromadzenie, lub wręcz przeciwnie zbyt małą ilość danych. Dlatego niezbędne jest znalezienie w trakcie weryfikacji danych, jeżeli istnieje taka możliwość odpowiedniej wartości tych parametrów.

Jak już zostało napisane w sekcji [4.3.1.1](#sssec:ivtresults), w trakcie testów odkryto, iż prawidłowy parametr prędkości mieści się w bliskim sąsiedztwie wartości . Możemy zaobserwować duże różnice wewnątrz wynikach końcowych, dlatego należy ten parametr dostosowywać w zależności od wykonanego pomiaru, najlepiej indywidualnie dla każdego elementu. Przykładowe wyniki tego algorytmu odpowiednio dla wartości parametru *v* wynoszących *0.0001, 0.005* oraz *0.001* i *0.01* dla pliku *1\_03\_1310301904.cal* zaprezentowano w odpowiedniej kolejności na rysunkach [4.5](#fig:ivt1), [4.6](#fig:ivt2), [4.7](#fig:ivt3), [4.8](#fig:ivt4). Na rysunkach [4.5](#fig:ivt1) oraz [4.7](#fig:ivt3) możemy zaobserwować ruch sakadyczny występujący pomiędzy punktami, nie tylko prawidłowe fiksacje. Wynik otrzymany na rysunku [4.8](#fig:ivt4) posiada za małą liczbę otrzymanych elementów, dla takiej próbki danych. Wykrywa on tylko punkty bardzo długiego skupienia oka. Rysunek [4.6](#fig:ivt2) przedstawia precyzyjnie fiksacje, jednak ze względu na rzeczywiste drgania oka, wykrywa też więcej punktów niż te pokazane na rysunku [4.8](#fig:ivt4).

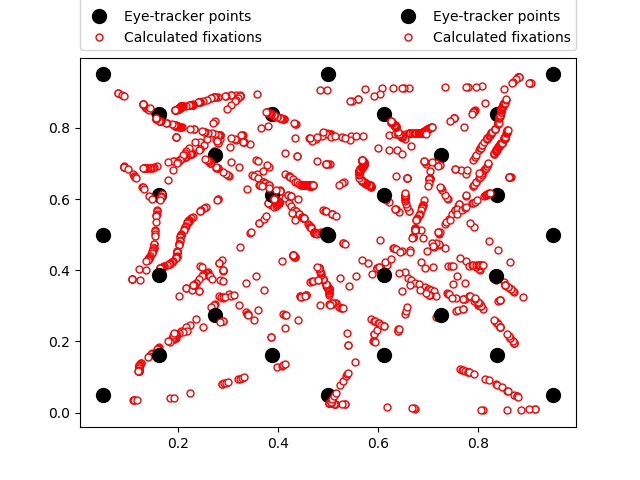
Nie jest zaskakującym wynikiem czas trwania algorytmu, wraz z wykorzystaniem pamięciowym, gdyż jedno porównanie do następnego elementu w tablicy oraz dalsza praca nad grupami fiksacji, w wypadku poprawnie skonstruowanego kodu nie powinno stanowić dużej złożoności czasowej oraz pamięciowej. W porównaniu do algorytmów I-DT oraz ML wykonanie oraz złożoność pamięciowa tego algorytmu są najmniejsze, jednak wyniki przy niedokładnie dobranym parametrze są najmniej dokładne, gdyż potrafią fałszywie uwzględnić ruch sakadyczny jako fiksację.



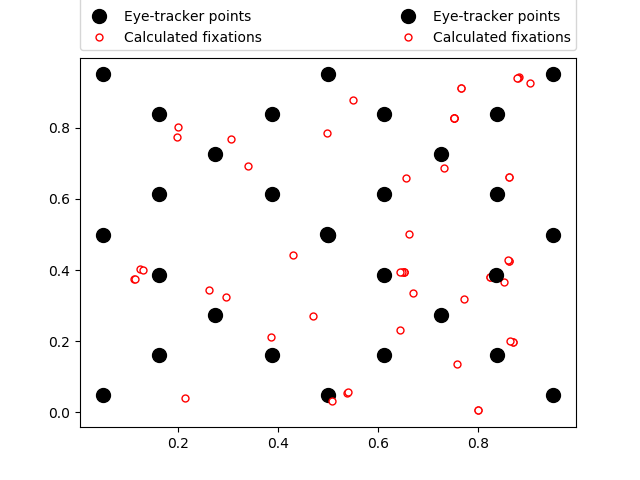
Wynik algorytmu I-VT dla v = 0.0001



Wynik algorytmu I-VT dla v = 0.0005

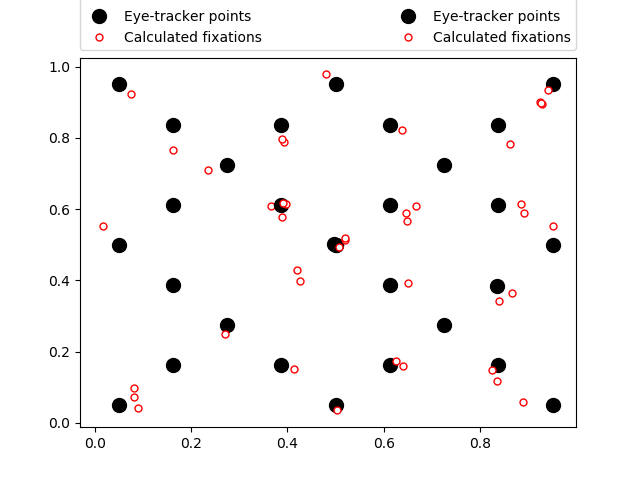


Wynik algorytmu I-VT dla v = 0.001



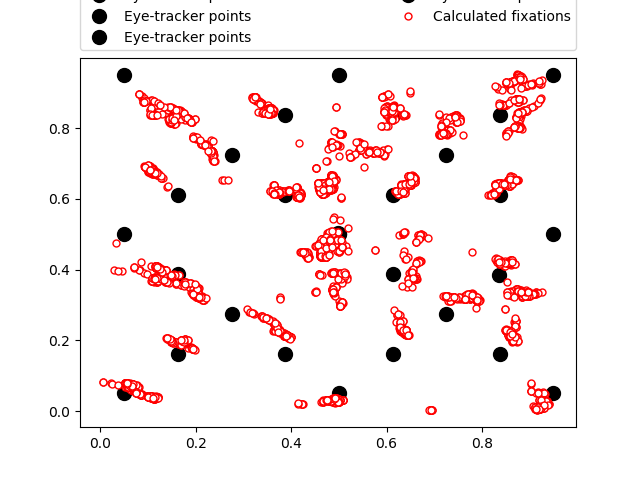
Wynik algorytmu I-VT dla v = 0.01

Analizując algorytm I-DT możemy wywnioskować, iż jest on najwolniejszy ze wszystkich algorytmów. Porównując wyniki tego algorytmu z algorytmami ML i I-VT, możemy również stwierdzić, iż jest on dość dokładny. Zaskakującym rezultatem jest podobieństwo w wykorzystaniu pamięci przez ten algorytm w porównaniu do algorytmu I-VT, ponieważ spodziewano się większego zużycia przez algorytm I-DT. Jak przedstawiono w wynikowych tabelach dotyczących algorytmu I-DT, większy wpływ na działanie algorytmu I-DT ma wartość parametru dyspersji. Czas okna służy tylko do stwierdzenia na jak długim wycinku danych ma być przeprowadzona analiza dyspersji. Przykładowy wynik algorytmu I-DT zaprezentowano na rysunku [4.9](#fig:idtresult)



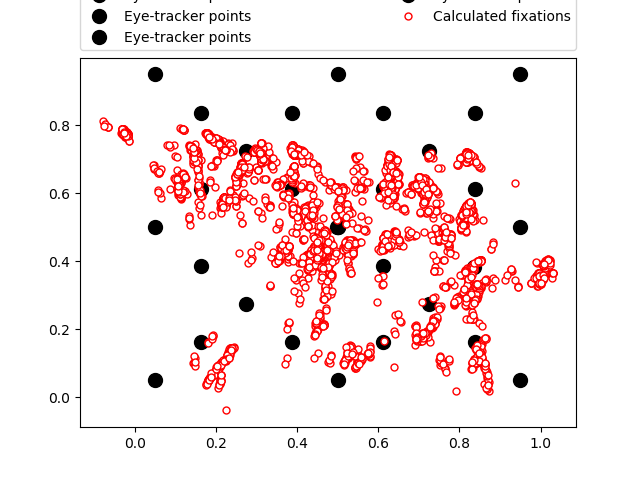
Wynik algorytmu I-DT dla D = 0.05 i T = 200 ms)

Algorytm uczenia maszynowego wymaga prawidłowego stworzenia modelu, by mógł on poprawnie funkcjonować. Porównując wyniki otrzymane za jego pomocą do tych otrzymanych za pomocą klasycznych algorytmów można stwierdzić, iż działa on dłużej od I-VT, ale zdecydowanie krócej od algorytmu I-DT, ale wykorzystuje on najwięcej pamięci operacyjnej do przygotowania obliczeń. Niebezpieczeństwem przy źle utworzonym modelu jest możliwość nieotrzymania danych wyjściowych, co zaprezentowano w tabelach umieszczonych w sekcji [4.3.1.3](#sssec:mlivt). Patrząc na otrzymaną precyzję końcową da się określić, iż wybrane parametry modelu (prędkość międzypunktowa oraz odległość) są dobrze dobrane. Na rysunku XXX zaprezentowany został wynik końcowy dla pliku *1\_03\_1310301904.cal* przy określeniu następujących parametrów: oraz proporcjach podziału modelu 35%-65%.



Wynik algorytmu ML dla v = 0.0005 oraz P=(35%-65%)

Pomimo podobnej liczby wykrytych punktów do algorytmu I-VT możemy zaobserwować większe nagromadzenie punktów w obliczonych fiksacjach przy właściwych punktach w stosunku do tych przedstawionych na rysunkach [4.5](#fig:ivt1), [4.6](#fig:ivt2), [4.7](#fig:ivt3), [4.8](#fig:ivt4). Przykład niepoprawnych wyników spowodowanych błędnym modelem zaprezentowano na rysunku [4.11](#fig:incorrectml).



Wynik algorytmu ML dla v = 0.0005 oraz P=(50%-50%)

Na tym rysunku można zaobserwować punkty skupienia będące umieszczone zarówno w okolicach punktów, jak i w trakcie ruchu sakadowego.

Powyższe wyniki pozwalają nam określić, iż algorytm ML posiada najlepszy stosunek prędkości wykonywanych operacji do dokładności, i gdyby przeznaczyć ten projekt do dalszych celów badawczych, zalecane jest korzystanie z tego rozwiązania. Algorytm I-DT mimo, iż jest dokładny, wymaga dość dużego nakładu czasowego, gdyż wykonuje się prawie 100 razy więcej niż algorytmy I-VT i ML. Zaleca się z niego korzystać w przypadku mniejszej ilości danych. Natomiast algorytm I-VT jest dość niedokładny, co może powodować błędy w dalszej analizie punktów fiksacji. W celu poprawnego działania, proponuje się zastosowanie dla każdego z plików osobnej wartości prędkości, celem osiągnięcia lepszych rezultatów.

## Podsumowanie i wnioski

Podsumowując, w tej pracy zrealizowano cel określony w rozdziale [[chapter:ch1]](#chapter:ch1), tzn. zaimplementowano wybrane algorytmy wykrywania fiksacji oraz przeprowadzono ich analizę. Zanim rozpoczęto analizę należało się zaznajomić z terminologią stosowaną przy tego typu rozwiązaniach, jak również rozważyć sposób przygotowania aplikacji. Zanim przystąpiono do implementacji wybrano platformę do implementacji, oraz zastanowiono się nad tym, które algorytmy wybrać do dalszej analizy. Ze względu na podzielenie aplikacji na wiele plików, m.in. podzielenie każdego algorytmu i potrzebnych mu metod na osobne pliki, czytelność kodu i możliwość rozwoju aplikacji w trakcie jej tworzenia stała się prostsza.

Celem dalszego rozwoju tego projektu, można zaimplementować obsługę wielu wątków, na pewno doprowadziłoby to do redukcji czasu wykonywania obliczeń. Wykorzystanoby to w podzieleniu konkretnych modułów aplikacji na odpowiednie wątki, czy też uruchamianie obliczeń plików na osobnych wątkach. Kolejnym krokiem rozwoju mogłaby być implementacja algorytmów powierzchnowych, czy też innych algorytmów dyspersyjnych lub prędkościowych opisanych w pracy . Możnaby się zastanowić także nad implementacją tych algorytmów przy wykorzystaniu innej platformy, np. *R*, w celu porównania wydajności różnych narzędzi. Ostatnim elementem, dla którego na ten moment zauważa się możliwość rozwoju jest wykorzystanie różnych klasyfikatorów przy uczeniu maszynowym oraz ich porównanie, gdyż rozwiązanie zaprezentowane w rozdziale [3.4.6](#ssec:machinelearningalg) pozwala na implementację wielu klasyfikatorów, a nie tylko regresji logistycznej, jednak przez ograniczenia czasowe zaniechano tego w tej wersji projektu.

Ze względu na małe doświadczenie w korzystaniu z niektórych narzędzi i rozwiązań przed rozpoczęciem tworzenia projektu, np. uczenia maszynowego, obsługi biblioteki *NumPy* czy też tylko podstawowa znajomość języka *Python*, niektóre elementy potrzebne do poprawnego zaimplementowania funkcjonalności początkowo sprawiały proste problemy typu obsługa specjalnych tablic, tworzenie specjalnych typów obiektów. Przejrzystość w dostępnej dokumentacji do tych modułów, czy też pospolitość występowania tego rodzaju problemów na forach dyskusyjnych dotyczących konkretnych rozwiązań ułatwiły naukę, a przez to rozwiązywanie postawionych problemów.

Realizacja tych problemów pozwoliła na poszerzenie wiedzy w zakresie uczenia maszynowego, czy też dokładnej analizy danych, oraz uporządkowała podstawy biologii związanej z budową i działaniem oka. Również zaobserwowany został rozwój w jakości tworzonego kodu, zgodnie z zasadami jego tworzenia w języku Python.

99 Arkadiusz Raj - optyczne.pl (2015), <https://bit.ly/2WmM9qn>, Monitor dla fotografa - część II : patrzeć i widzieć, strona 1 Dario D. Salvucci, Joseph H. Goldberg (2000), *Identifying Fixations and Saccades in Eye-Tracking Protocols*, Eye Tracking Research and Applications Symposium 2000 eSzkola.pl (2019), <https://bit.ly/33n7qD8>, Rysunek 1 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 (2010),  
*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. Matthew H Schneps, Jennifer M Thomson (2013), *Shorter Lines Facilitate Reading in Those Who Struggle*, Figure 4 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 Timothy C. Hain, MD (2012) <https://dizziness-and-balance.com/practice/eyemove.html> *Eye Movement Recording Devices*, Figure "Scleral Eye Coil" Tobii Pro AB,  
<https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/> 2019

# Dodatki

### Zawartość płyty

Do pracy dołączona została płyta z następującą zawartością: oryginał pracy magisterskiej, pliki źródłowe aplikacji, prezentacja, praca magisterska w wersji bez obrazków i tabel oraz wyniki badań.

1. ang. area of interest [↑](#footnote-ref-43)
2. z ang. Identification-Velocity Threshold [↑](#footnote-ref-46)
3. z ang. Identification Dispersion-Threshold [↑](#footnote-ref-50)
4. Machine Learning [↑](#footnote-ref-52)
5. z ang. Graphical User Interface - graficzny interfejs użytkownika [↑](#footnote-ref-60)
6. Machine Learning .NET [↑](#footnote-ref-62)
7. Należy nadmienić, iż chodzi o standard .NET Framework, a nie .NET Core, dla którego F# posiada otwarty kod źródłowy [↑](#footnote-ref-63)
8. binary JSON [↑](#footnote-ref-64)
9. k najbliższych sąsiadów [↑](#footnote-ref-101)
10. Mebibajt (1MiB = 1024 KB) [↑](#footnote-ref-116)