POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

KIERUNEK: AUTOMATYKA I ROBOTYKA

Praca dyplomowa inżynierska

Tytuł pracy dyplomowej inżynierskiej

autor: Szymon Ciemała

kierujący pracą: dr inż. Krzysztof Jaskot

konsultant: dr inż. Imię Nazwisko

Gliwice, grudzień 2021

Spis treści

Rozdział 1

Streszczenie

Praca inżynierska „Rozpoznawanie obiektów z wykorzystaniem biblioteki OpenCV", łączy tematykę wizji komputerowej oraz algorytmów uczenia maszynowego.

Praca skupia się na stworzeniu wygodnej w użyciu oraz powszechnie dostępnej biblioteki umożliwiającej wykorzystanie gestów, pozycji dłoni, obrotu dłoni oraz odległości między palcem wskazującym a kciukiem jak elementów sterujących w dowolnych projektach napisanych w języku Python.

Do napisania pracy wykorzystano biblioteki języka Python o otwartym kodzie źródłowym, głównie OpenCV, MediaPipe oraz SciKit Learn.

Rozdział 2

Wstęp

- wprowadzenie w problem/zagadnienie
- osadzenie problemu w dziedzinie
- cel pracy
- zakres pracy
- zwięzła charakterystyka rozdziałów
- jednoznaczne określenie wkładu autora, w przypadku prac wieloosobowych – tabela z autorstwem poszczególnych elementów pracy

2.1 Wprowawdzenie w problem

Rozwój technolgii w ostatnich czasach przyczynił się do coraz częstszego wykorzystywania wizji komputerowej oraz metod uczenia maszynowego do rozpoznawania oraz klasyfikacji różnego typu obiektów, w tym części ludzkiego ciała. Pozwala to na interakcję człowieka z aplikacjami, często w spób bardziej naturalny.

2.2 Cel pracy

Projekt inżynierski ma na celu stworzenie biblioteki w języku Python, która pozwoli na przystępne wykorzystanie algorytmów rozpoznawania gestów oraz ruchu dłoni. Biblioteka powinna oferować gotowe rozwiązania, na przykład przygotowane modele matematyczne pozwalające na rozpoznawanie języka migowego oraz gestów podstawowych. Dodatkowo powinna pozowlić na wyznaczenie pozycji dłoni, jej typu oraz wartości charakterystycznych, na przykład odlgłości między końcówkami wybranych palców czy kąta obrotu dłoni.

2.3 Osadzenie problemu w dziedzienie

Aktualnie istnieją częściowo gotowe rozwiązania pozwalające na rozpoznanie i klasyfikację dłoni - bibliotka MediaPipe.

2.4 Charakterystyka rozdziałów

W pierwszym rozdziale zostanie poruszony temat genezy problemu oraz jego sformułowania. Zostaną dodatkowo opisane jego założenia wraz z wyamaganą funkcjonalnością projektu.

W drugim temacie zostaną opisne techniczne tworzonej biblioteki oraz narzędzia wymagane do jej stworzenia. Zakres opisywanych narzędzi rozpocznie się od opisu wybranego systemu operacyjnego do programów służących do stworzenia pobieralnej paczki na platformi PyPi.

W rozdziale nr 5 zostaną opisane wymagania sprzętowe wymagane do poprawnego działania funkcji biblioteki. Przedstawione zostanie działanie klasy wraz

z jej parametrami oraz sposób jego wykorzysania. W drugiej jego części zostaną przedstawione przykłady wykorzysania modułu. Przykładowym projektami będą: program rozpoznający gest, interaktywny kiosk bezdotykowy i system doboru koloru przy pomocy gestu.

Kolejny rozdział opisuje budowę klasy wraz z najważniejszymi algorytmami oraz strukturami danych. Dodakowo opisany zostanie Jupyter Notebook służący do generowania modeli uczenia maszynowego rozpoznających gest dłoni.

Rozdział 3

[Analiza tematu]

- sformułowanie problemu
- osadzenie tematu w kontekście aktualnego stanu wiedzy (state of the art) o poruszanym problemie
- studia literaturowe [?, ?, ?, ?] - opis znanych rozwiązań (także opisanych naukowo, jeżeli problem jest poruszany w publikacjach naukowych), algorytmów,

3.1 Sformułowanie problemu

Problem można podzielić na trzy części. Każda z nich odpowiada za wybraną część funkcjonalności biblioteki.

- Wyznaczenie pozyci dłoni, odległości pomiędzy wybranymi palcami oraz jej kąta obrotu.
- Rozpoznawanie gestów dłoni w dwóch trybach: prostym (parę dostępnych gestów) oraz zaawansowanym, który rozoznaje alfabet języka migowego.
- Dostępność biblioteki poprzez platfromę PyPi.

Stworzenie modułu będzie wymagało napisania klasy wykorzystującej odpowiednie biblioteki pozwalających na rozpoznanie elementów charakterystycznych dłoni oraz przetworzenia obrazu. Obraz będzie pochodził z kamerki internetowej, który zostanie odpowiednio przetworzony z wykorzystaniem funkcji dostępnych

poprzez bibliotekę OpenCV. Przetworzony obraz zostanie wykorzystany przez metody biblioteki MediaPipe, która pozowli na rozpoznanie elementów charakterystycznych dłoni. W napisanej klasie zostaną zaimplementowane metody, które pozowlą na rozpoznanie typu dłoni (prawa, lewa), odległości między paliczkiem palca wskazującego oraz kciuka, oraz obrotu dłoni.

Kolejnym elementem jest wygenerowanie modelu matematycznych klasyfikujących gesty dłoni. W tym celu zostanie wykorzystana bibliotegk SciKit Learn wraz z dostępnymi poprzez nią algorytmamy ucznia maszynowego. Odpowiednio zebrane dane pozowlą na przeprowadzeni procesu ucznie dla paru wybranych algorytmów.

Najważniejszym elementem pracy jest powyżej wymieniony ostatni punkt listy. Dzięki wykorzystaniu platformy PyPi deweloperzy będą mogli przy pomocy programu **pip** za pomocą jednej komendy zainstalować paczkę wraz ze wszystkimi wymaganymi zależnościami.

3.2 Podobne rozwiązania

Rozdział 4

Wymagania i narzędzia

4.1 Wymagania Funkcjonalne i Niefunkcjonalne

Klasa powinna zawierać w sobie wszystkie niezbędne funkcje oraz parametry pozwalające na wykorzystanie jej w dowolnym projekcie, takie jak na przykład obliczanie kąta obrotu dłoni względem nadgarstka czy rozpoznawanie gestów. Dwa podstawowe modele rozpoznające gesty powinny zostać uprzednio przygotowane, natomiast użytkownik powinien mieć dodatkowo możliwość stworzenia własnego modelu rozpoznającego wybrane ułożenie dłoni.

Pobranie modułu powinno być możliwe poprzez wykorzystanie standardowego menedżera do zarządzania paczkami w języku Python, czyli programu **pip**. Ten menedżer pozwala na instalacje paczki wraz z jej zależnościami, czyli innymi paczkami wymaganiami do poprawnego działania pobieranej biblioteki.

Na głównej stronie projektu powinna zostać zamieszczona krótka dokumentacja w postaci pliku **README.md**, w którym zostanie opisany proces instalacji paczki, jej funkcjonalność oraz przykładowe programy.

4.2 Narzędzia

Przy tworzeniu modułu ważne są wykorzystywane narzędzia, zaczynając od wybranego systemu operacyjnego, a kończąc na programach pozwalających na przygotowanie plików źródłowych wysyłanych na zdalne repozytorium PyPi, każde

z nich jest kluczowe do zbudowania pełnego projektu.

4.2.1 System operacyjny Ubuntu

Do stworzenia oprogramowania została wybrana dystrybucja Ubuntu w wersji 20.04 LTS. System został wybrany ze względu na istnienie takich elementów jak powłoka systemowa **bash**, menedżer pakietów oraz wsparcie dla języka Python.

4.2.2 GitHub

Platforma GitHub, która wykorzystuje rozproszony system kontroli Git, pozwala na stworzenie głównej strony biblioteki, na której znajdą się pliki programu wraz z dokumentacją oraz instrukcją instalacji i korzystania. W czasie pracy nad modułem system Git pomaga w tworzeniu kopii zapasowych. W przypadku dalszego rozwoju projektu GitHub może posłużyć jako medium pracy innych kontrybutorów.

4.2.3 Python

Do napisania biblioteki został wykorzystany język skryptowy Python. Dzięki swojej popularności oraz dostępności, Python stał się językiem powszechnie stosowanym w pracy związanej z zagadnieniami uczenia maszynowego, analizy danych oraz przetwarzania obrazów. Na platformie PyPi można znaleźć wiele narzędzi pozwalających na pracę właśnie w tych dziedzinach, jak i również wielu innych.

Ze względu na swoją budowę, Python nie należy do najwydajniejszych języków. Python jest językiem interpretowanym, co oznacza, że jego program nie zostaje skompilowany do kodu maszynowego, a zostaje on zinterpretowany przez interpreter. Ma ta jednak swoje zalety, skrypt można wykonywać w częściach, co pozwala na testowanie działania programu. Dzięki czemu łatwiej jest znajdować błędy i na bieżąco je likwidować. Potencjał tego w pełni wykorzystuje interaktywna powłoka iPython.

4.2.4 iPython

To interaktywna powłoka dla języka Python, która rozszerza jego działanie o introspekcję, czyli możliwość wykonywania poprzednich części programu. W praktyce oznacza to, że użytkownik ma możliwość wykonywania programu zawartego w komórkach w dowolnej kolejności, nawet jeśli oznacza to wykonywanie programu zapisanego w komórkach poprzedzających aktualną.

Dodatkowo iPython oferuje możliwość korzystania z komend wiersza poleceń. Pozwala to, na przykład na instalowanie modułów wewnątrz programu.

4.2.5 Jupyter Notebook

Do obsługi interaktywnej powłoki iPython, wykorzystany został Jupyter Notebook („notatnik"), czyli webowy edytor, który został stworzony z myślą o pracy związanej z analizą danych oraz obliczeniami naukowymi.

Dodatkowym atutem jest fakt, że Jupyter Notebook pozwala na zapis tekstu w języku Markdown, czyli języku znaczników stosowanym do formatowania tekstu. Co pozwala na czytelny opis programu oraz wstawianie obrazów, jeśli są wymagane. Dodatkowo wszystkie wykresy generowane przez na przykład bibliotekę **matplotlib** będą wyświetlane na stałe pod komórką, w której został wykonany program. Pozwala to na stworzenie programu, który może służyć jednocześnie jako interaktywna instrukcja.

4.2.6 Visual Studio Code

Do stworzenia oprogramowania został wybrany edytor Visual Studio Code. Edytor jest programem o otwartym kodzie źródłowy i został stworzony przez korporację Microsoft. Aplikacja posiada wiele darmowych rozszerzeń, często tworzonych przez użytkowników edytora. To właśnie dzięki rozszerzeniom program pozwala na pracę z wieloma typami plików oraz języków programowania.

4.2.7 Python Package Index

Python Package Index, w skrócie PyPi, jest platformą, na której udostępniane są moduły dla języka Python. Aktualnie jest ona zarządzana przez fundację

„Python Software Foundation". Paczki pobierane są przy pomocy programu **pip**, standardowo instalowanego wraz z językiem Python, co oznacza, że PyPi jest oficjalnym repozytorium oprogramowania właśnie dla tego języka.

Każdy użytkownik, organizacja lub firma mają możliwość stworzenia swojej własnej paczki i udostępnienia jej na repozytorium wraz z krótką dokumentacją oraz instrukcją instalacji.

4.2.8 Programy bdist_wheel, sdist orz twine

Oba programy są niezbędne do przygotowania w pełni działającej paczki udostępnianej na repozytorium PyPi. Program **sdist** pozwala na stworzenie źródłowej dystrybucji, czyli w praktyce pliku typu **.zip**, **.tar** czy też **.gztar**, w których znajdują się wybrane pliki będące częścią biblioteki.

Kolejnym programem jest **bdist_wheel**, który odpowiada za stworzenie paczki typu **WHEEL**, która pozwala na szybszą instalację niż w przypadku instalacji wykorzystującej pliki źródłowe. WHEEL pozwala na pominięcie etapu kompilacji, przez co kompilator nie jest wymagany na sprzęcie użytkownika.

Ostatnim program jest **twine**, który pozwala na załadowanie gotowej paczki na repozytorium wraz z autoryzacją poprzez HTTPS. Program wspiera różne typy paczek, w tym typ WHEEL.

4.3 Biblioteki

Biblioteki opisane w tej sekcji są bibliotekami o otwartym kodzie źródłowym. Pozwala to na wykorzystanie ich w projekcie bez opłacania lub łamania żadnych licencji.

4.3.1 OpenCV

OpenCV jest biblioteką, która zajmuje się wizją komputerową, w tym głównie przetwarzaniem obrazów w czasie rzeczywistym. Projekt został rozpoczęty przez firmę Intel w 2001 roku, a później był wspierany przez laboratorium Willow Garge, które jest odpowiedzialne za stworzenie systemu ROS. OpenCV wspiera

wiele języków programowania: C++, C#, Python, Java oraz JavaScript, co oznacza, że jest to biblioteka wieloplatformowa i znajduje ona zastosowanie w wielu aplikacjach oraz systemach.

4.3.2 MediaPipe

Biblioteka MediaPipe, która została stworzona przy wsparciu firmy Google, udostępnia wieloplatformowe oraz konfigurowalne rozwiązania wykorzystujące uczenie maszynowe w dziedzinie rozpoznawania, segmentacji oraz klasyfikacji obiektów wizji komputerowej. Niektórymi z rozwiązań są:

- Segmentacja włosów oraz twarzy
- Śledzenie pozycji obiektów
- Rozpoznawanie dłoni

Oprogramowanie tworzone przez MediaPipe jest wysoce zoptymalizowane, co pozwala na wykorzystanie go na urządzeniach codziennego użytku, smartfonach, czy komputerach osobistych.

4.3.3 SciKit-Learn

SciKit-Learn początkowo został stworzony jako dodatek do biblioteki SciPy jako projekt w ramach **Google Summer of Code** w roku 2007. Biblioteka oferuje różnego typu metody uczenia maszynowego, w tym algorytmy klasyfikacji, regresji oraz analizy skupień. Przykładowym algorytmami w bibliotece są:

- Las losowy – polegająca na konstruowaniu wielu drzew decyzyjnych w czasie uczenia.
- Algorytm centroidów – algorytm wykorzystywany w analizie skupień.
- Maszyna wektorów nośnych – algorytm klasyfikujący, często wykorzystywany w procesie rozpoznawania obrazów.

Rozdział 5

Specyfikacja Zewnętrzna

5.1 Wymagania sprzętowe

5.1.1 Kamera

Elementem niezbędnym do korzystania z modułu OpenLeap jest kamera, która pozwoli na pozyskanie obrazu. Rozdzielczość matrycy kamery powinna być wystarczająco duża, aby pozwolić na rozpoznanie dłoni. Nie ma tutaj minimalnych wymagań, większa rozdzielczość, czy też lepsza praca w warunkach niskiego oświetlenia kamery pozwoli na poprawniejsze działania algorytmów identyfikujących dłoń. Podobnie ma się liczba klatek na sekundę, która powinna być wystarczająco wysoka, tak aby można było sprawnie korzystać z możliwości oprogramowania.

5.1.2 Komputer

Jednostka obliczeniowa powinna zostać wyposażona w system operacyjny dający możliwość obsługi języka Python, taki warunek spełnia większość systemów operacyjnych na rynku. Komputer powinien spełniać minimalne wymagania w kwestii wydajności przetwarzania obrazu. Fakt istnienia możliwości instalacji i wykorzystania biblioteki MediaPipe przez minikomputery Raspberry Pi w wersji 3 i 4 oznacza, że wymagania nie są wysokie.

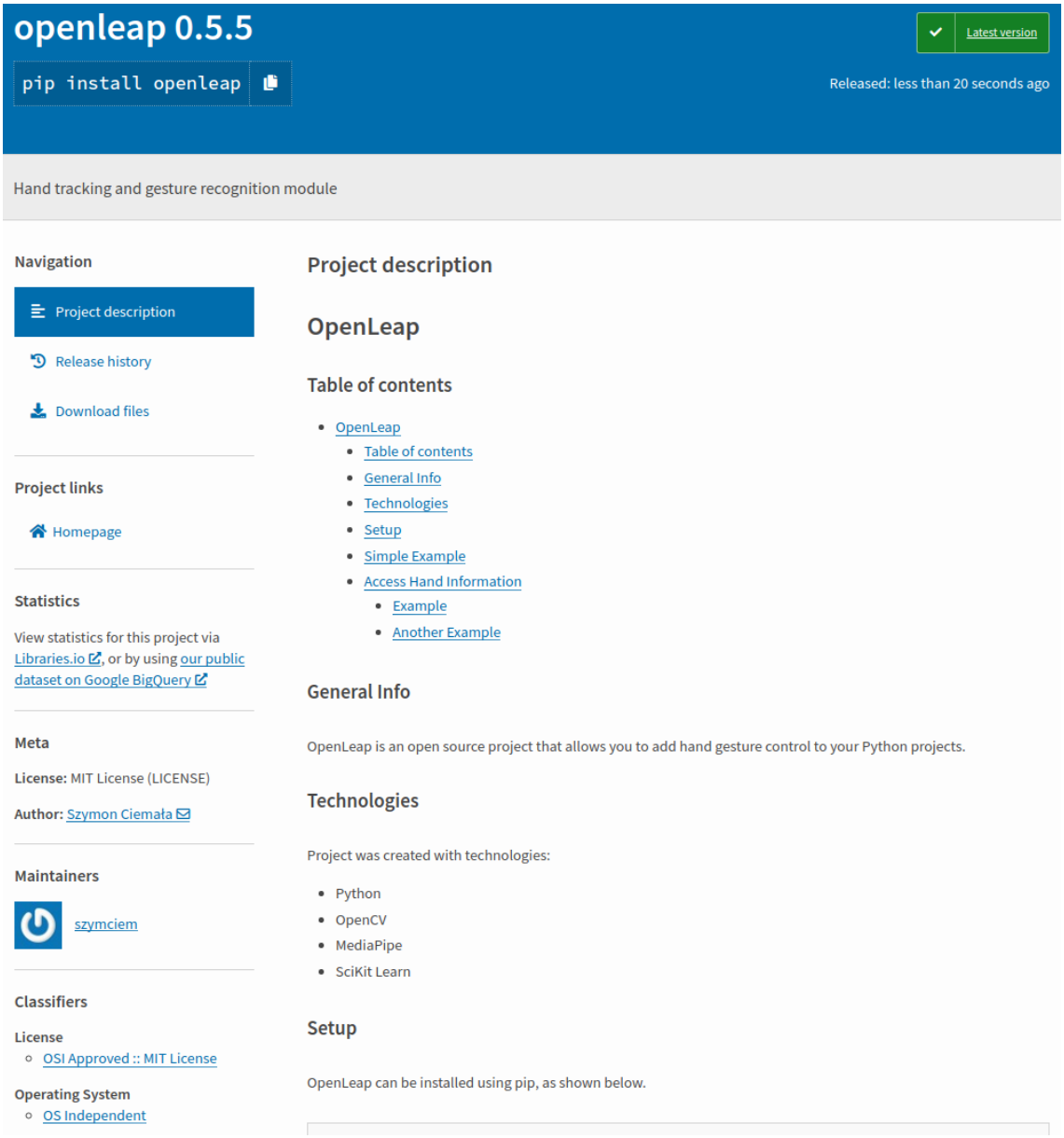
5.2 Instalacja paczki

Instalacja paczki odbywa się poprzez wykorzystanie programu **pip**, który jest instalowany automatycznie razem z językiem Python. W zależności od wybranego systemu operacyjnego komenda może przybierać różne formy, ogólnie można przyjąć poniższy zapis 5.1. Komenda powinna zostać wykonana poprzez powłokę **bash** lub inną dostępną w systemie Unix-owym lub poprzez wiersz poleceń w systemie Windows.

```
$ pip install openleap
```

Listing 5.1: Instalacja paczki

Informacje na temat biblioteki można znaleźć na platformie PyPi pod linkiem: **https://pypi.org/project/openleap/**. Na tej stronie znajduje się opis modułu, instrukcja instalacji oraz przykładowe programy i możliwości wykorzystania.



Rysunek 5.1: Strona modułu OpenLeap na PyPi

5.3 Program testowy

Paczkę można przetestować korzystając z dostępnych metod klasy. W ramach testu istnieje możliwość napisania programu wyłącznie w powłoce języka

Python. W pierwszym kroku do programu zostaje zaimportowana paczka **openleap**. Zostaje stworzony obiekt kontrolera z wybranymi parametrami, dokładny opis parametrów znajduje się w późniejszej części rozdziału, w podsekcji (5.3.1). Metoda **loop()** odpowiada za wywołanie głównej logiki programu odpowiedzialnej za generowanie danych oraz wyświetlanie ich w wybranym miejscu (powłoka, okno graficzne). Funkcja **loop()** zawiera sobie pętlę nieskończoną, więc aby dane z kontrolera mogły być wykorzystane w dalszej części programu, funkcja **loop()** powinna zostać wywołana jako osobny wątek.

```
1    import openleap
2
3    controller = openleap.OpenLeap(screen_show=True,
4                                   screeen_type='BLACK',
5                                   show_data_on_image=True,
6                                   gesture_model='basic')
7
8    controller.loop()
```

Listing 5.2: Program testowy

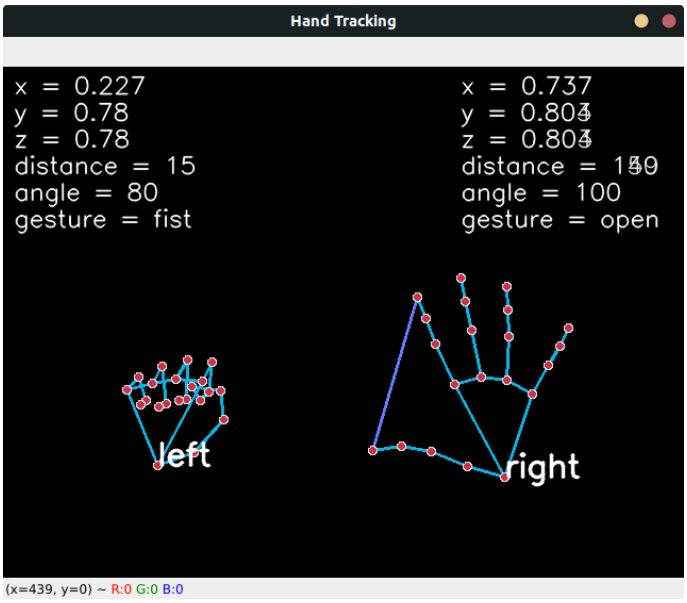
Alternatywą dla powyższego przykładu jest wykorzystanie metody **main()**, która nie wymaga wykorzystania wątku do obsługi kontrolera. Natomiast należy ją wywołać w pętli oraz zapisać warunek zamknięcia okna, jeśli takowe zostało wywołane. Do zamknięcia okna można posłużyć się wbudowanymi funkcjami: **detect_key()** oraz **close_window()**.

```
1   import openleap
2
3   controller = openleap.OpenLeap(screen_show=True,
4                                  screeen_type='BLACK',
5                                  show_data_on_image=True,
6                                  gesture_model='basic')
7
8   while True:
9       controller.main()
10
11      if controller.detect_key('q'):
12          controller.close_window()
13          break
```

Listing 5.3: Program testowy

Tak zapisany program (5.3) pozwala na równoległe wykorzystanie kontrolera z resztą pisanego programu, bez wywoływania jego logiki we wcześniej wspomnianym wątku.

Przykładowy program pozwoli na wyświetlenie okna z widocznymi dłońmi wraz z oznaczonymi punktami charakterystycznymi (końcówki palców, stawy) oraz opisem parametrów, takich jak obrót dłoni względem nadgarstka, jej pozycja względem lewego górnego rogu obrazu kamery czy rozpoznany gest. Zrzut ekranu testowego programu znajduje się poniżej na rysunku 5.2.



Rysunek 5.2: Zrzut ekranu programu testowego

Opis generowanych danych

- **x, y, z** – Pozycja dłoni opisana jako pozycja punktu nadgarstka. Może być znormalizowana lub nie. Układ współrzędnych ma swój początek w lewym górnym rogu obrazu pobranego z kamery.
- **distance** – Odległość między końcówką palca wskazującego a końcówką kciuka. Zaznaczona przez niebieską linię. Odległość może być również znormalizowana, czyli liczona na podstawie znormalizowanej pozycji elementów.
- **angle** – Obrót dłoni, który obliczany jest jako kąt ramienia łączącego wybrany punkt dłoni oraz punkt nadgarstka, względem układu współrzędnych, którego środkiem jest właśnie punkt nadgarstka.
- **gesture** – Rozpoznany gest dłoni na podstawie wybranego modlu.

Dane zostają zapisane w słowniku składającym się z obiektów typu **dataclass**, które przechowują wygenerowane informacje o prawej i lewej dłoni. Klasa **dataclass** zostanie dokładniej opisana w kolejnym rozdziale.

Pobranie informacji o danej dłoni polega na podaniu typu dłoni jako klucz słownika **data**, który jest atrybutem obiektu kontrolera. Po znaku kropki należy zapisać nazwę pobieranej danej, na przykład „gesture" lub „distance" tak jak w poniższych przykładach.

```
1     if controller.data['right'].gesture == 'open':
2         print('Right␣hand␣is␣opened!')
3     elif controller.data['right'].gesture == 'fist':
4         print('Right␣hand␣is␣closed!')
```

Listing 5.4: Odczyt gestów

```
1     if controller.data['right'].distance < 20:
2         print('Click␣has␣been␣detected!')
```

Listing 5.5: Odczyt edległości między palcami

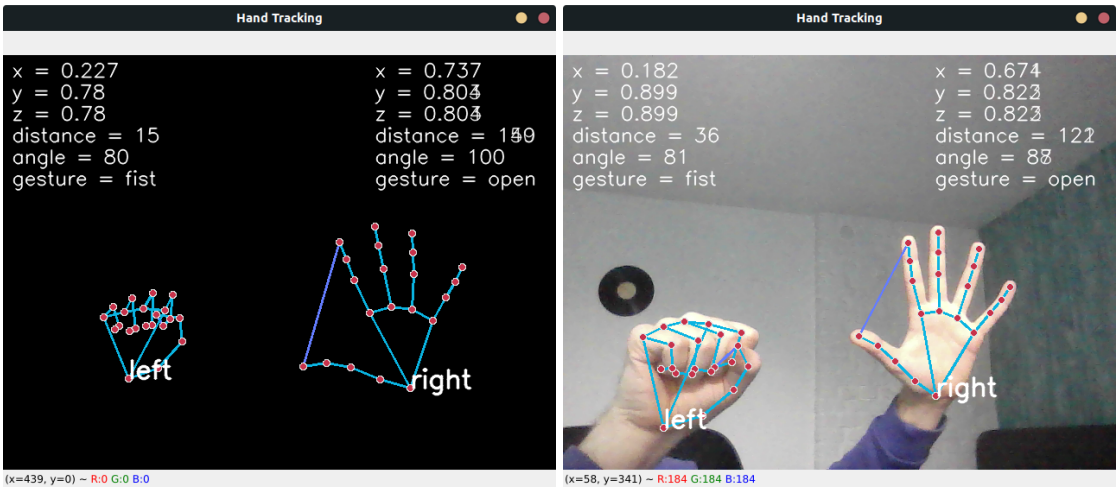
5.3.1 Parametry

Obiekt klasy **openleap** przyjmuje za argumenty inicjalizatora parametry, określające działanie programu. Parametry określają czy należy wyświetlić okno graficzne, wartości obliczanych danych oraz, który model rozpoznający gesty ma zostać wybrany. Wszystkie atrybuty klasy zostaną dokładnie opisane w kolejnym rozdziale. W tej sekcji zostały opisane jedynie parametry inicjalizatora wraz ze swoimi typami.

Parametry Inicjalizatora:

- **screen_show** (boolean) – podgląd okna
- **screen_type** (string) – typ tła wyświetlany w oknie graficznym
  - „cam" – obraz z kamery
  - „black" – czarne tło
- **show_data_in_console** (boolean) – wyświetlanie danych w konsoli
- **show_data_on_image** (boolean) – wyświetlanie danych w oknie graficznym

- **normalized_position** (boolean) – wyświetlanie znormalizowanej pozycji
- **gesture_model** (string) – wybór modelu rozpoznającego gesty
  - „basic” – gesty podstawowe
  - „sign_language” – język migowy
  - Trzecią opcją jest podanie ścieżki do innego modelu.
- **lr_mode** (string) – metoda rozpoznawania typu dłoni
  - „AI” – wykorzystanie algorytmów klasyfikacji
  - „position” – określenie typu na podstawie pozycji dłoni według siebie
- **activate_data** (boolean) – Warunek odpowiadający za ciągły zapis do struktury danych **data**, opisanej w przykładzie programu 5.4.



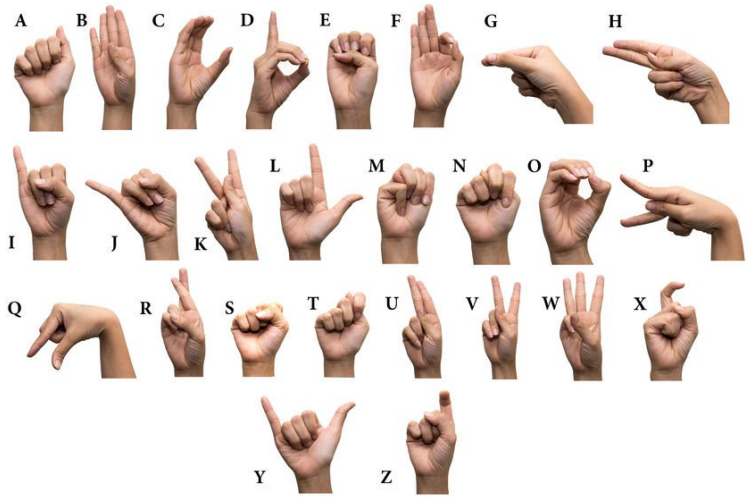
(a) Czarne tło. (b) Obraz z kamery.

Rysunek 5.3: Parametr określający typ tła.

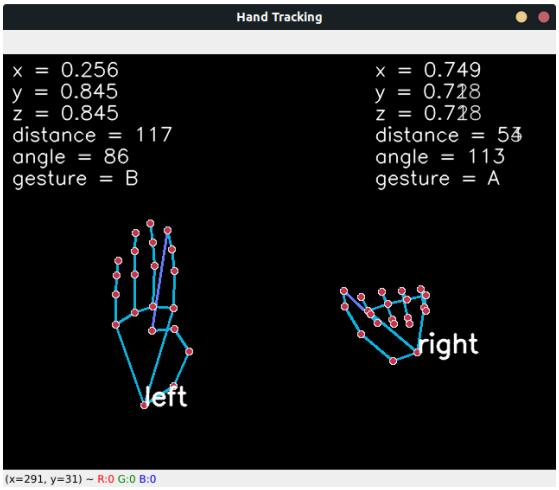
5.4 Przykłady użycia

5.4.1 Rozpoznawanie alfabetu w języku migowym

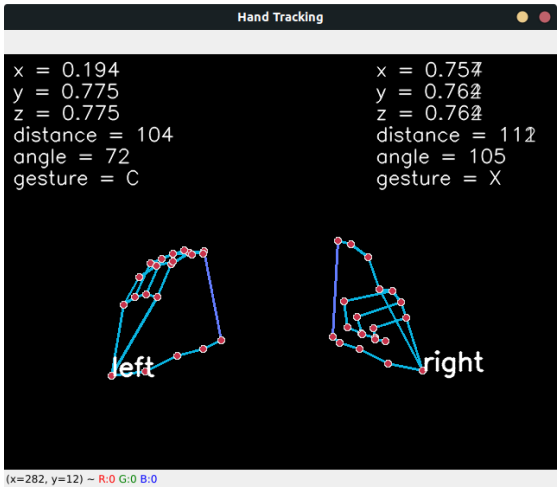
Pierwszym przykładem zastosowania jest wykorzystanie paczki do rozpoznawania alfabetu języka migowego. Taki program może umożliwić komunikację między osobą głuchoniemą posługującą się językiem migowym a osobą, która takiego języka nie zna. Przygotowany model rozpoznaje litery przedstawione na poniższej grafice 5.4. Dzięki wykorzystanym algorytmom uczenia maszynowego model potrafi rozpoznać skomplikowane ułożenie dłoni symbolizujące daną literę.



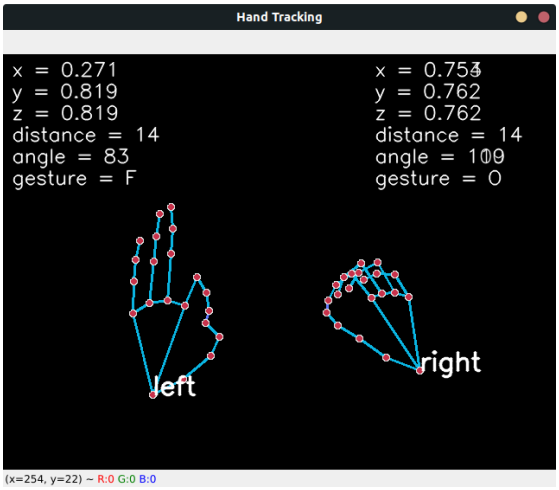
Rysunek 5.4: Gesty alfabetu języka migowego



(a) Litery A i B.



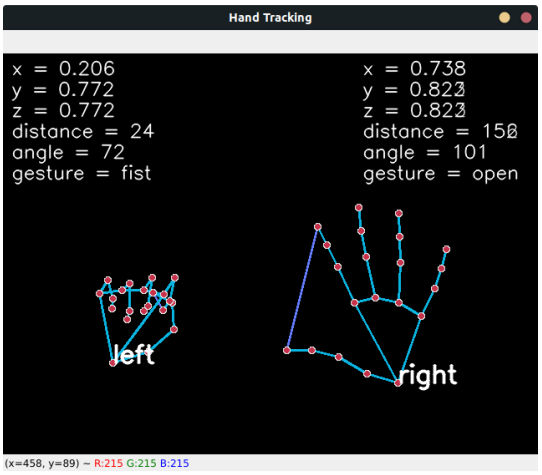
(b) Litery C i X.



(c) Litery F i O.

Rysunek 5.5: Rozpoznawanie języka migowego.

Oprócz modelu rozpoznającego gesty alfabetu języka migowego klasa została wyposażona w model rozpoznający podstawowe gesty, czyli gesty otwartej i zamkniętej dłoni. Wybór odpowiedniego modelu odbywa się w momencie tworzenia obiektu poprzez ustawienie parametru **gesture_model**. Programista może wybrać model, który lepiej sprawdzi się w tworzonej aplikacji.



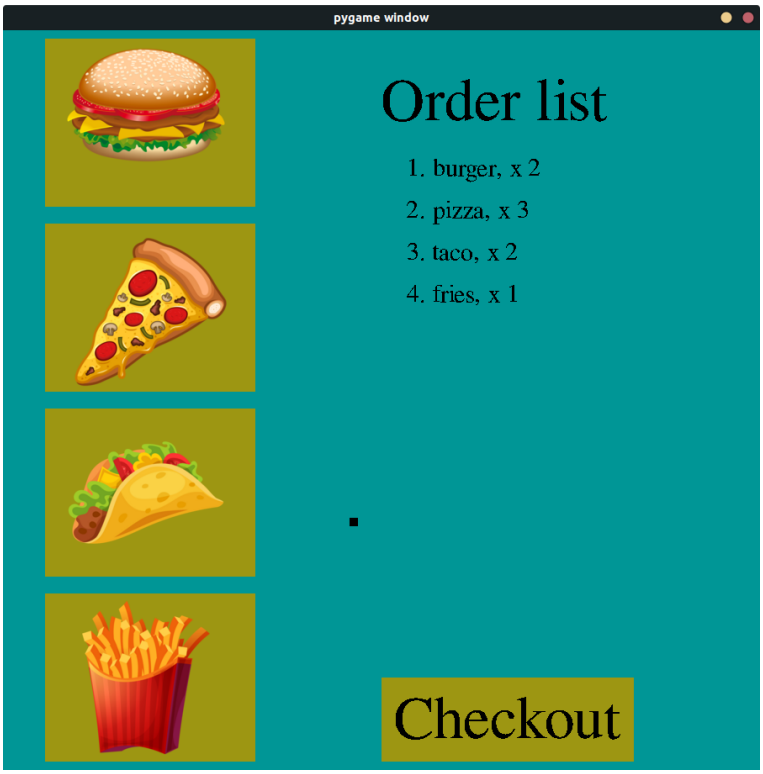
Rysunek 5.6: Gesty otwartej i zamkniętej dłoni

5.4.2 Interaktywny Kiosk

Kolejnym przykładem jest Interaktywny kiosk, który pozwala na złożenie zamówienia w sposób, który nie wymaga dotykania ekranu dotykowego. Warunkiem komfortowego korzystania z aplikacji jest wysoka liczba klatek na sekundę generowana przez kamerę, co wpływa na płynność ruchu kursora.

W dobie pandemii, ale i nie tylko, takie rozwiązanie może potencjalnie przyczynić się do spowolnienia rozprzestrzeniania się różnego rodzaju wirusów i drobnoustrojów, poprzez brak konieczności dotykania ekranu w kasach samoobsługowych.

Wskaźnik kiosku interaktywnego jest sterowany poprzez pozycję dłoni, czyli informację, która jak już wiadomo jest generowana automatycznie. Kliknięcie przycisku zostaje aktywowane poprzez wykrycie odpowiednio małej odległości między końcówką palca wskazującego a końcówką kciuka.

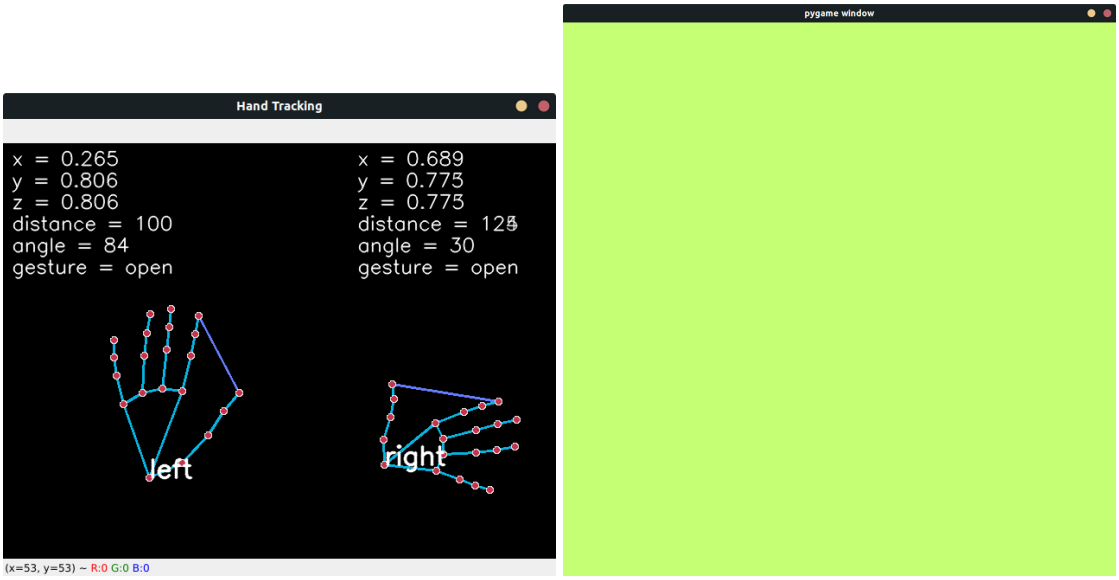


Rysunek 5.7: Zrzut ekranu kiosku interaktywnego

5.4.3 Dobór koloru

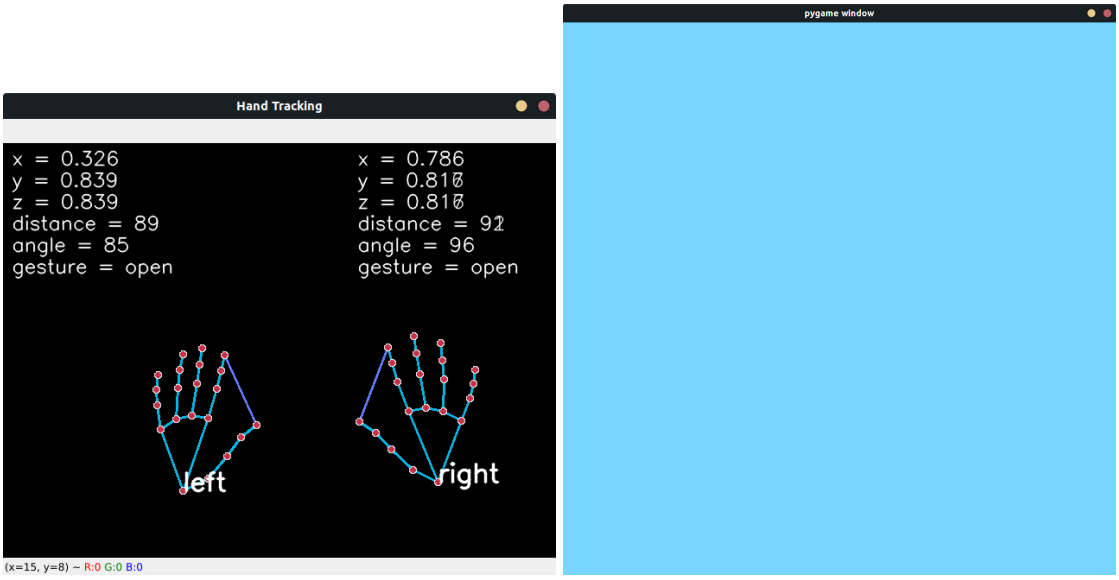
Ostatnim przykładem jest wykorzystanie dłoni jako kontrolera, za którego pomocą można wybrać dowolny kolor. Takie zastosowanie może zostać wykorzystane w pracy grafika komputerowego. Dzięki temu użytkownik będzie mógł zmieniać kolor wykorzystywanego narzędzia, na przykład przy malowaniu. Kolor można ustawiać tylko, wtedy kiedy gest lewej ręki jest gestem otwartej dłoni. Dzięki czemu obrót ręki zmienia kolor tylko wtedy kiedy jest to wymagane.

Kolor wybierany jest poprzez wykorzystanie przestrzeni kolorów HSV. Wartość kąta obrotu jest mapowana do wartości H, która w praktyce określa odcień koloru. Reszta parametrów to saturacja oraz jasność. Saturacją można sterować poprzez dostosowanie wartości parametru **distance**, która też jest odpowiednio mapowana.



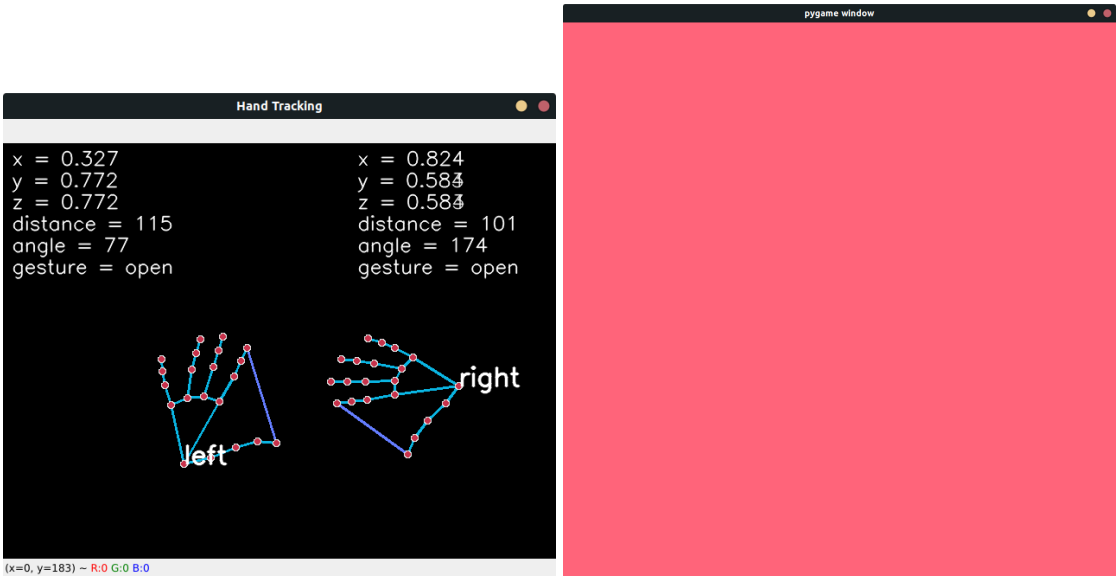
(a) Widok dłoni. (b) Dobrany kolor.

Rysunek 5.8: Obrót dłoni o 30 stopni.



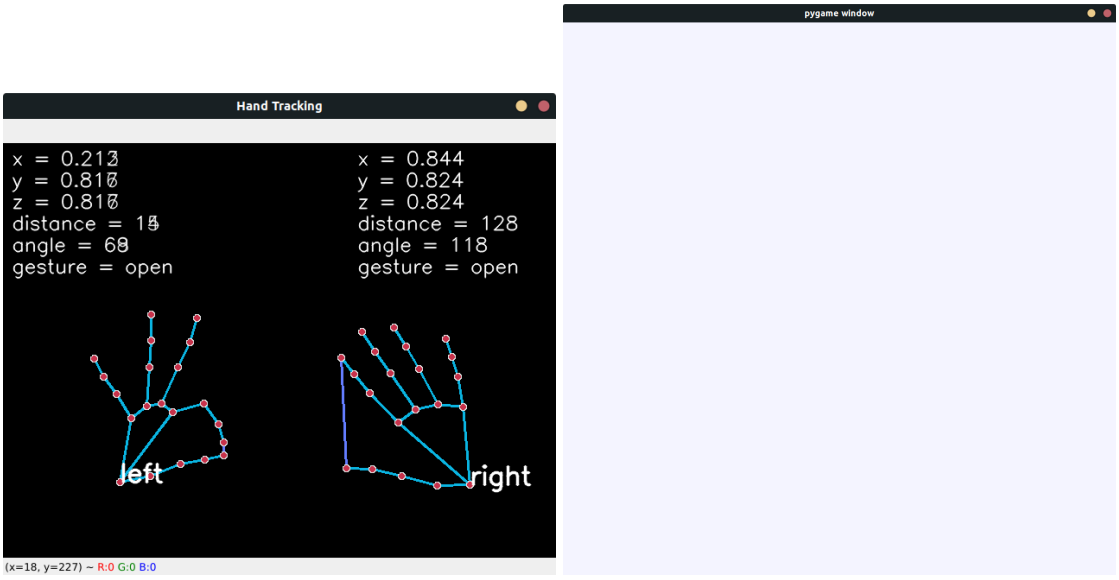
(a) Widok dłoni. (b) Dobrany kolor.

Rysunek 5.9: Obrót dłoni o 96 stopni.



(a) Widok dłoni. (b) Dobrany kolor.

Rysunek 5.10: Obrót dłoni o 174 stopni.



(a) Widok dłoni. (b) Dobrany kolor.

Rysunek 5.11: Ustawienie saturacji

5.5 Stworzenie nowego modelu

Użytkownik biblioteki może stworzyć własny model rozpoznający gesty. Wykorzystać do tego można program napisany przy pomocy Jupyter Notebook. Plik jest dostępny do pobrania na platformie **GitHub**. Plik posiada rozszerzenie **.ipynb**, ale można go również otworzyć przy pomocy edytor Visual Studio Code. Link do notatnika: https://bit.ly/3J72U0z

Plik został przygotowany w formie instrukcji, tak aby osoba korzystająca mogła krok po kroku zebrać oraz przygotować dane, a później wytrenować model i go zapisać. Zapisany model może zostać ponownie wykorzystany w programie podając jego ścieżkę jako argument inicjalizatora o nazwie **gesture_model**.

W praktyce przy pomocy instrukcji zostanie wygenerowanych parę modeli korzystających z różnych algorytmów uczenia maszynowego. Z tego powodu w notatniku została przygotowana funkcja, która oblicza poprawność działania każdego z modeli. Natomiast użytkownik może wybrać dowolny model, niekoniecznie ten, który daje najlepszy wynik, ale na przykład ten, który wykorzystuje wymaganą technologię. Tworzenie modelu oraz wybrane algorytmy klasyfikacji zostaną dokładnie opisane w kolejnym rozdziale.

To narzędzie daje spore możliwości w przypadku dopasowania modelu rozpoznającego gesty do wymagań pisanego oprogramowania. Można stworzyć dowolny model rozpoznający tylko wybrane gesty, co może przenieść się na poprawność działania algorytmu klasyfikującego. Na przykład, mniej gestów oznacza, że model niekoniecznie musi rozpoznawać subtelne różnice w ułożeniu dłoni względem podobnych gestów. W takim przypadku aplikacja będzie działać sprawniej i istnieje większe prawdopodobieństwo, że gest zostanie poprawnie rozpoznany.

Rozdział 6

Specyfikacja Wewnętrzna

6.1 Klasa OpenLeap

6.1.1 Wykorzystanie innych klas

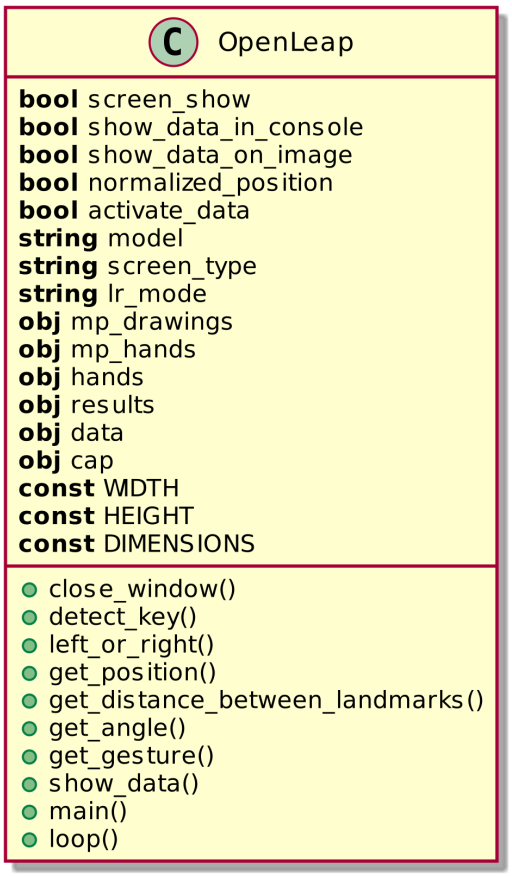
Klasa OpenLeap korzysta z różnych bibliotek. Natomiast nie dziedziczy ona żadnej klasy. Zostały jedynie wykorzystane wybrane metody tworzonych obiektów wewnątrz klasy.

OpenLeap wykorzystuje metody klasy **mediapipe.solutions.hands.Hands** oraz **cv2.VideoCapture**. Pierwsza klasa pozwala na stworzenie obiektu, którego zadaniem jest rozpoznawanie dłoni na podstawie podawanego obrazu. Druga klasa tworzy obiekt, który pozwala na pobranie obrazu z kamery, określenie jego wielkości oraz sprawdzenia, czy kamera jest w ogóle dostępna.

6.1.2 Budowa klasy

Klasa posiada atrybuty, których część jest parametrami, czyli dokładnie argumentami inicjalizatora klasy. Oprócz parametrów istnieją jeszcze zmienne oraz obiekty klas odpowiadające za rozpoznanie dłoni, czy wykonanie operacji związanych z obliczeniem wartości zapisywanych w słowniku data, który też jest atrybutem klasy.

Niektóre metody klasy mogą zostać wykonane automatycznie poprzez funkcje **main()** lub **loop()**. Natomiast można też wykonać potrzebne operacje bezpośrednio z wykorzystaniem wbudowanych metod, co może pomóc w optymalizacji programu.



Rysunek 6.1: Schemat klasy OpenLeap

6.1.3 Atrybuty klasy

Część atrybutów klasy została opisana w podrozdziale **Parametry**, są to te elementy, które mogą zostać zdefiniowane przez użytkownika. Druga część atrybutów jest generowana automatycznie.

1. **mp_drawings** — Obiekt pobierany z biblioteki **MediaPipe** pozwalająca na rysowanie obrysu dłoni w oknie generowanym przez OpenCV.

2. **mp_hands** – Obiekt przechowujący informacje, na przykład o indeksach opisujących poszczególne elementy charakterystyczne dłoni.
3. **hands** – Model biblioteki MediaPipe rozpoznający dłonie. Inicjalizacja tego obiektu wymaga podania dwóch parametrów.
    - **min_detection_confidence** – Minimalna wartość szacunkowa, dla której model określa czy została wykryta dłoń.
    - **min_tracking_confidence** – Minimalna wartość szacunkowa, pozwalająca określić dokładność śledzenia dłoni.
4. **results** – Obiekt przechowujący informacje o rozpoznanych dłoniach i ich elementach.
5. **data** – Słownik, w którym są przechowywane informacje o obydwóch dłoniach.

6.1.4 Metody klasy

W klasie zostały stworzone metody, które pozwalają na zbudowanie logiki programu.

1. **close_window()** – Zamknięcie wszystkich okien biblioteki OpenCV.
2. **detect_key()** – Wykrycie kliknięcia wybranego przycisku podanego w argumencie.
3. **left_or_right()** – Metoda rozpoznająca lewą oraz prawą dłoń. Metoda za argumenty przyjmuje:
    - **index** – indeks badanej dłoni
    - **mode** – metoda określania typu dłoni
        - **„AI”** – Wykorzystuje gotowy model MediaPipe. Niestety ten może nie zawsze działać poprawnie.
        - **„position”** – Drugą metodą jest bazowanie na pozycji dłoni, dłoń po prawej stronie względem drugiej dłoni jest dłonią prawą i odwrotnie. Jeśli na ekranie widoczna jest jedna dłoń, wtedy funkcja jest wywoływana ponownie z argumentem **mode** ustawionym na **„AI”**.

- **hand** – Obiekt przechowujący współrzędne elementów charakterystycznych danej dłoni.

4. **get_position()** – Funkcja zwraca pozycję elementu danej dłoni.
   - **index** – indeks badanej dłoni
   - **landmark_idx** – indeks elementu charakterystycznego, którego pozycję ma zwrócić funkcja
   - **normalized** – Parametr określający czy zwrócone współrzędne mają zostać znormalizowane.
5. **get_distance_between_landmarks()** – Metoda obliczająca odległość między dwoma wybranymi elementami charakterystycznymi.
   - **index** – Indeks dłoni, na której znajdują się mierzone elementy.
   - **landmark_1** – indeks pierwszego elementu dłoni
   - **landmark_2** – indeks drugiego elementu dłoni
   - **normalized** – Parametr określający czy odległość ma zostać znormalizowane.
6. **get_angle()** – Metod zwracająca kąt obrotu wybranego elementu charakterystycznego dłoni względem nadgarstka.
   - **index** – indeks wybranej dłoni
   - **landmark_idx** – Indeks elementu względem, którego obliczany jest kąt.
   - **mode**
     - **half** – kąt półpełny
     - **full** – kąt pełny
   - **unit**
     - **radians** – jednostka kąta w radianach
     - **degrees** – jednostka kąta w stopniach
7. **get_gesture()** – Metoda określa gest wybranej dłoni.
   - **index** – indeks wybranej dłoni

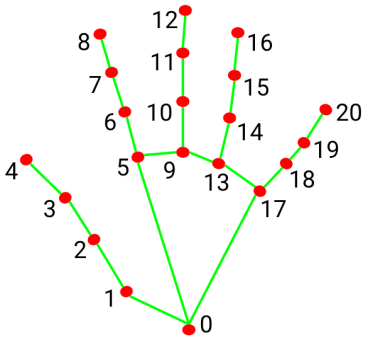
8. **show_data()** – Metoda wyświetlająca dane w oknie lub w konsoli w zależności od wybranej opcji.
   - **console** – Flaga określająca czy dane mają zostać wypisane w konsoli.
   - **on_image** – Flaga określająca czy dane mają zostać wypisane na ekranie.
   - **image** – Wybrany obraz, na którym mają zostać wypisane dane.
9. **main()** – Główna metoda, w której wykonywane są niezbędne obliczenia oraz funkcje.
10. **loop()** – Metoda, w której wywoływana jest metoda **main()** w pętli.

6.2 Struktury danych

Klasa zawiera w sobie parę struktur danych, które są kluczowe dla wygodnego korzystania z klasy.

6.2.1 Struktura mp_hands

Struktura mp_hands jest enumeratorem. Przechowuje ona informacje o indeksach wszystkich elementów dłoni. Została ona stworzona po to, aby łatwiej było zapisywać program, który odczutje informacje o danym elemencie.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Rysunek 6.2: Elementy charakterystyczne dłoni

Pierwszym elementem określanym przez enumerator jest nadgarstek. Dalej znajduje się reszta indeksów, oznaczająca resztę części palców i dłoni.

```
1     x = hand.landmark[self.mp_hands.HandLandmark.WRIST].x
```

Listing 6.1: Przykładowe wykorzystanie **mp_hands**

Tak jak zostało to ukazane powyżej, enumerator może zostać wykorzystany, na przykłd do odczytu wpółrzędnej danego puntku. Jest to ułatwienie, dzięki któremu program staje się czytelnieszy dla programisty lub osoby przeglądającej program.

6.2.2 Struktura results

Struktura **results** jest obiektem, który przechowuje informacje zwrócone przez metodę **hands.process()**. Klasa OpenLeap korzysta z dwóch typów informacji zwracanych przez tą metodę.

- **multi_hand_landmarks** - obiekt przechowujący dane o pozycjach wszystkich elementów dłoni. Przykład wykorzystania został przedstawiony powyżej.

```
    [landmark {
        x: 0.7409825921058655
        y: 0.6843034029006958
        z: 0.0
        }
        .
        .
        .
        landmark {
        x: 0.7874422073364258
        y: 0.26234549283981323
        z: -0.16528795659542084
        }
    ]
```

Listing 6.2: Przykład elementu obiektu

- **multi_handedness** - lista przechowująca dane o wszysktich dłoniach widocznych na obraze. Danymi są indeks, typ dłoni oraz punktacja określającą pewność poprawnego rozpoznania dłoni.

```
        [classification {
                index: 1
                score: 0.9482673406600952
                label: "Right"
            }
        ]
```

Listing 6.3: Przykład elementu obiektu

6.2.3 Dataclass

Strukura typu **Dataclass** pozwala na stworznie struktury podobnej do **struct** istniejącej w języku programowania **C**. Taka klasa pozwala na storzenie obiektu składającego się jedynie z atrybutów wymaganych do opisue elementów klasy kontrolera. Dodatkowym atutem takiej klasy jest możliwość prostego odczytu zapisanych danych, korzystając z operatora kropki, a nie z operatrów opisujących słownik lub listę.

```
1    @dataclass
2    class Data:
3        x : float = 0
4        y : float = 0
5        z : float = 0
6        distance: float = 0.0
7        angle: float = 0.0
8        gesture: str = None
```

W języku Python stworzenie klasy typu **dataclass** zaczyna się od zapisania dekoratora. W tym wypadku nie jest wymagana funkcja inicjalizująca obiekt, czyli **___init___**. Parametetry początkowe można podać w chwili tworzenia obiektu, lub później. W programie zostały przypisane wartości początkowej, tak jak w przykładzie powyżej.

6.2.4 Słownik

Instancja stworzonego typu **Data** zostnie zainicjowana dla każdej dłoni (lewej oraz prawej). Te instacje zostaną zapisane w słowniku.

```
1     data = {'right':Data(), 'left':Data()}
```

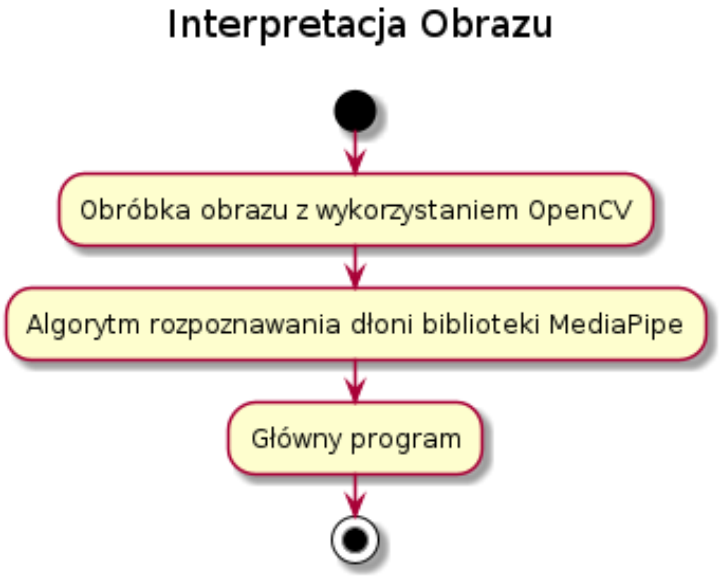
Taka konstrukcja pozwoli użytkownikowi w prosty i przejrzysty sposób na odnajodowania potrzebnych wartości oraz informacji.

6.2.5 Pickle

Modele matematyczne, które zostaną wytrenowane muszą zostać zapisane, tak aby można było je wykorzystać ponownie. Z tego powodu został wykorzystany plik typi **pickle**, który umożliwia zapis zmiennych oraz obiektów w postaci binarnej. Dzięki czemu można je wykorzystać ponownie pomimo restartu programu.

6.3 Rozpoznawanie dłoni

Pierwszym elementem projektu jest rozpoznanie dłoni poprzez wyznacznie pozycji elementów charakterystycznych. Pozycja każdego z tych elementów, jak już zostało to opisane, jest względna według lewego górnego rogu obrazu kamery.



Rysunek 6.3: Przygotowanie obrazu

W pierwszym kroku obraz powinnien zostać pozyskany z kamery oraz odpowiednio przetworzony przez funkcje biblioteki OpenCV.

Kolejnym krokiem jest rozpoznanie elementów charakterystycznych dłoni przez model biblioteki MediaPipe.

Na końcu powinna zostać wykonana główna część programu. Na przykład ropoznanie gestu czy obliczenia kąt obrotu.

6.3.1 OpenCV - przygotowanie obrazu z kamery

W pierszym kroku należy zainicjalizować obiekt obsługujący kamerę. Argumntem jest identyfikator określający, która kamera podłączona do systemu ma zostać wykorzystana. W przypadku kiedy dostępna jest tylko jedna kamera wystarczy wpisać wartość równą 0, tak jak poniżej.

```
1         self.cap = cv2.VideoCapture(0)
```

W metodzie **main()** w pierwszym kroku warunek określa czy zostało otwarte połączenia z kamerą. Jeśli tak to należy pobrać z niej aktualną klatkę obrazu (**frame**).

```
1         Main function that runs the core of the program.
2         """
3
4         hand_type = None
5         if self.cap.isOpened():
6             ret, frame = self.cap.read()
7
8             #BGR to RGB
9             image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

Model rozpoznający dłoń korzysta z przestrzeni barw RGB (red, green, blue), a nie BGR (blue, green, red). Dlatego należy obraz przekształcić właśnie na przesrzeń RGB. Dodatkowo, obraz powinnien zostać obrócony horyzntalnie, tak aby stworzył lustrzane odbicie względem użytkownika ustawionego na przeciwko kamery.

```
1
2             #Flip horizontal
3             image = cv2.flip(image, 1)
4
5             #Set flag
6             image.flags.writeable = False
```

Flage **writeable** ustawiona na **False** pozwala na uzyskanie lepszej wydajności przy przetwarzaniu obrazu w ramach modelu rozpoznającego dłonie.

```
1
2             #Detections
3             self.results = self.hands.process(image)
4             # print(self.results.multi_hand_landmarks)
5
6             #Set flag back to True
7             image.flags.writeable = True
8
9             #RGB to BGR
```

Przestrzeń barw zostaje przywrócona do RGB, aby była mogła zostać poprawnie wyświetlona przez funkcję biblioteki OpenCV.

```
1
2             if self.screen_show:
3                 if self.screen_type=='BLACK':
```

6.4 MediaPipe - Elementy charakterystyczne

Pozycje nadgarstka, paliczków oraz stawów dłoni zostaną wykorzystane do obliczenia obrotu dłoni względem punkut 0 oraz do wytrenowania modeli uczenia maszynowego, których zadaniem będzie rozpoznawnie wybranych gestów.

6.4.1 Generowanie grafiki dłoni

Generowanie grafiki nałożonej na daną dłoń wykonuje się przy pomocy przygotowanej funkcji biblioteki MediaPipe, która współpracuje z OpenCV.

6.5 Pomiary oraz inne ważne elementy

6.5.1 Pozycja dłoni

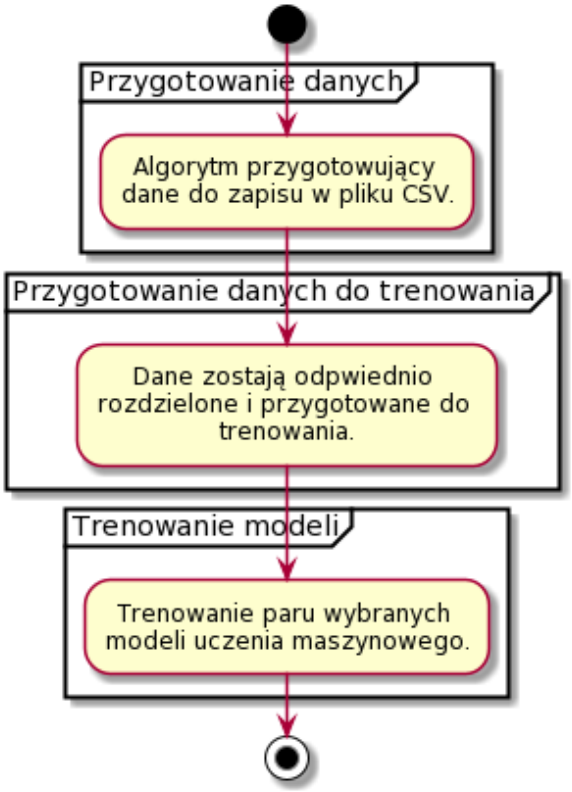
6.5.2 Obrót dłoni

6.5.3 Odległość między palcami

6.6 Rozpoznawanie gestów

6.6.1 Przygotowanie modeli uczenia maszynowego

Celem programu będzie stworzenie modeli matematycznych przy pomocy metod uczenia maszynowego, których celem będzie rozpoznawanie gestów dłoni. Proces tworzenie takiego modelu można podzielić na trzy kroki przedstawione na schemacie 6.3.



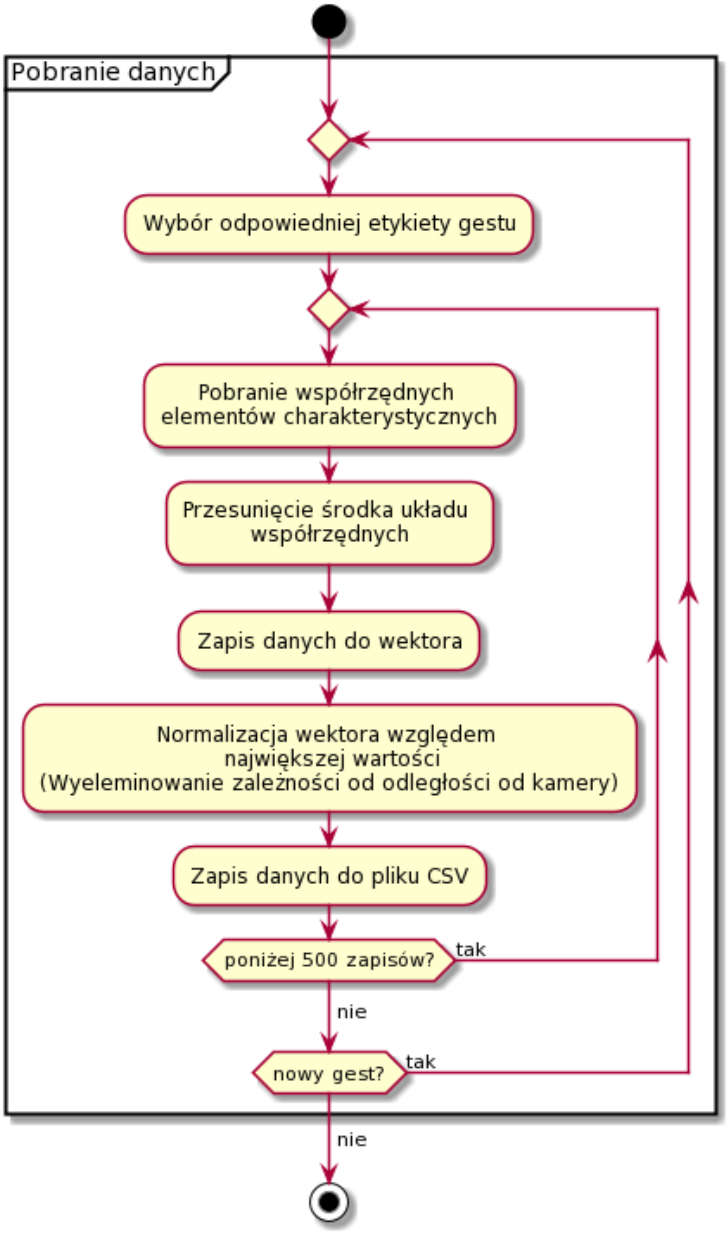
Rysunek 6.4: Ogólny algorytm przygotowania modeli uczenia maszynowego

Całość programu została napisana w notatniku Jupyter. Pozwala to na wykonanie pewnych części programu osobno, niezależnie od reszty programu. W praktyce każdy blok opisany w powyższym schemacie UML ma swoje odwzorowanie w notatniku. Na przykład, część zapisująca współrzędne do pliku będzie wykonywana tyle razy ile jest gestów do wytrenowania.

6.6.2 Zebranie danych

Algorytm zebrania danych polega na pobraniu współrzędnych oszacowanych przez model MediaPipe. Należy pamiętać o tym, że środek układu współrzędnych znajduje się lewym górnym rogu obrazu kamery. Oznacza to, że współrzędne w tej postaci nie nadają się do wyuczenia modelu matematycznego. Gest nie powinnien być rozpoznawany na podstawie pozycji dłoni w obrazie lub jej odległości od

kamery. Należy się tych zależności pozbyć.



Rysunek 6.5: Ogólny algorytm przygotowania modeli uczenia maszynowego

Gest dłoni, można scharakteryzować na podstawie pozycji elementów dłoni. Podstawowym problemem jest fakt, że pozycje elementów charakterystycznych są

opisane względem układu współrzędnych, którego środek znajduje się w lewym górnym rogu obrazu pobranego z kamery. Pozycja dłoni na obrazie nie ma żadnego znaczenia w kwestii określania gestu. Poprawnie przygotowane pozycje elementów powinnny zostać pozbawione zależności od pozycji dłoni względem obrazu. Przygotowanie danych do przetworzenia będzie wymagało paru opercaji matematycznych. W takim wypadku należy przprowadzić transformację, tutaj akurat przesunięcie układu współrzędnych do pozycji nadgarstka. Taka operacja pozwoli na opis pozycji elementów względem nagarstka. Ostatecznie pozbywamy się pozycji nadarstka z wektora danych, ponieważ jest ona środkiem nowego układu współrzędnych.

Macierz przesunięcia

$$M_p = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tak naprawdę, przesunięcie układu współrzędnych w osi Z nie będzie miało miejsca. Wartości współrzędnej Z reszty elementów, oprócz elementu z indeksem równym 0, czyli nadgarstka, są określane właśnie nadgarstka. Dlatego też, macierz przesunięcia w osi Z jest równa 0.

$$M_p = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Aby dane mogły zostać zinterpretowane przez algorytmy ucznenia maszynowego muszą one zostać przedstawione w postaci jednowymiarowej. Aktualna postać macierzy przedstawiającej wpółrzędne elementów charakterystycznych ma następującą postać. Indeksy współrzędnych są równoznaczne z indeksami elementów dłoni.

$$M_p = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ & \vdots & \\ x'_{21} & y'_{21} & z'_{21} \end{bmatrix}$$

Dane w postaci jednowymiarowej mają postać następującego wektora.

$$A_f = \begin{bmatrix} x_1 & y_1 & z_1 & x_2 & y_2 & \cdots & y_{21} & z_{21} \end{bmatrix}$$

Drugim krokiem jest uniezależnienie pozycji elementów od odległości dłoni od kamery. Najprostszym rozwiązaniem jest normalizacja wektora danych względem największej bezwzględnej wartości.

Normalizacja

$$A_n = \frac{A_f}{max(abs(A_f))}$$

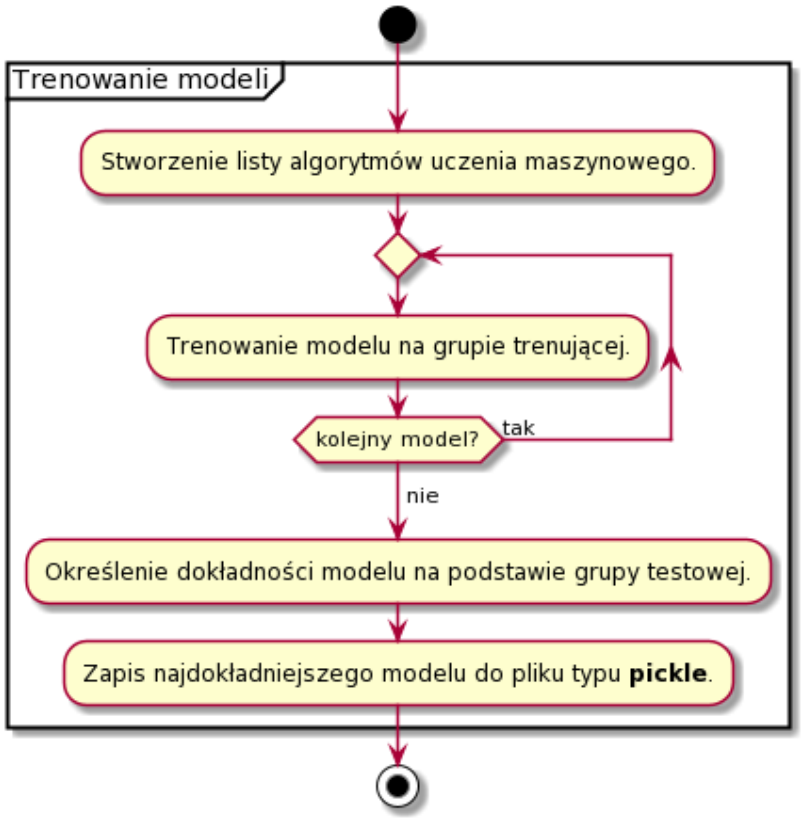
Każdy nowy wektor zostaje zapisany do pliku CSV z odpwowiednią etykietą. Zebrane dane posłużą do wytrenowania algorytmów uczenia maszynowego.

6.6.3 Budowa pliku CSV

Dane zapisane w pliku CSV opisują przykładowe współrzędne wszysktich elementów dłoni wraz z przypisaną etykietą oznaczającą gest.

6.6.4 Metody klasyfikacji - uczenie maszynowe

Przygotowane dane zostają odczytane z pliku CSV. W pierwszym kroku należy rozdzielić je na dwie części: współrzędne (dane wejściowe) oraz etykiety (dane wyjściowe). W kolejnym kroku należy te dwie grupy podzielić na grupę trenującą i grupę testową. Zadaniem grupy testowej będzie trenowanie wybranych modeli matematycznych, a grupy testowej przetestowanie ich dokładności.



Rysunek 6.6: Ogólny algorytm przygotowania modeli uczenia maszynowego

W celu wybrania najlepszej metody klasyfikacji, zostnie wybranych kilka algorytmów. Każdy z nich stworzy swój model, a ostatecznie zostanie sprawdzona ich poprawności z wykorzystaniem grupy testowej. Model z najepszym wynikiem zostanie zapisany do pliku typu **pickle**. W języku Python pliki typu **pickle** pozwalają na zapis zmiennych, obiektów lub innych struktur danych, które mają zostać wykorzystane w po zakończeniu programu.

6.6.5 Wybrane algorytmy klasyfikujące

Do wytrenowania modeli uczenia maszynowego z biblioteki SciKit Learn zostały wybrane następujące algorytmy.

- Logistic Regression

- Nearest Centroid
- Decision Tree Classifier
- Random Forest Classifier
- SGD Classifier
- Gradient Boosting Classifier
- MPL Classifier

6.6.6 Badanie dokładności każdego z algorytmów

6.6.7 Ponowne wykorzystanie modelu

Gotowy model pobieramy i testujemy w przykładowym programie.

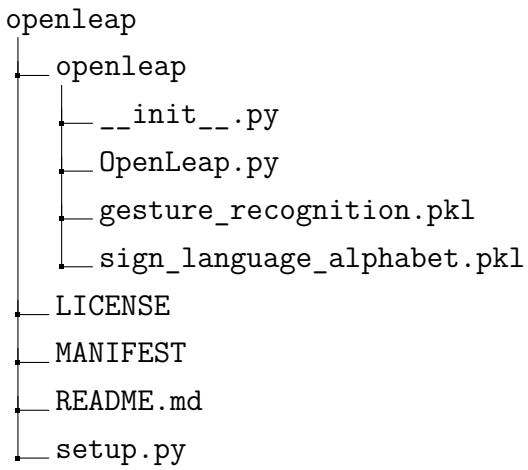
6.7 Paczka PyPi

6.7.1 Budowa paczki

Ostatecznym krokiem jest przygotowanie programu w formie paczki, która zostanie udostępniona na platformie PyPi. Wygama to przygotowania odpowiednich plików konfiguracyjnych oraz zostosowania stosownych narzędzi do stworznie pliku **wheel** oraz **tar**.

6.7.2 Struktura Paczki

Pierwszm krokiem jest przygotowanie odpowiedniej struktury paczki. Do tego celu został stworzony folder o poniższej strukturze. W tym folderze znajdują się wszystkie potrzebne elementy paczki. W podfolderze o tej samej nazwie znajduje się główna części modułu, czyli plik .py, w którym zapisana jest klasa OpenLeap. Dodatkowo w tym folderze znajdują się pliki typu **pickle**, w których zapisane są modele rozpoznające gesty.



Rysunek 6.7: Struktura paczki PyPi

6.7.3 Pliki Konfiguracyjne

Pliki setup.py oraz MANIFEST są plikami, które odpowiadają za konfigurację oraz opis paczki. W pliku setup.py zapisany jest numer aktualnej wersji, autor, kontakt do autora, nazwa paczki itp.

6.7.4 Załadowanie paczki do repozytorium

Przed załadowaniem paczki do repozytorium, należy stworzyć zapakowaną paczkę źródłową, na przykład typu .tar oraz plik typu WHEEL. Oba pliki spełniają tą samą funkcję, czyli przechowywnie niezbędnych elementów paczki oraz umożliwiają ich instalację na systemie użytkownika. Plik WHEEL pozwala na dużo szybszy proces instalacji niż instalacja ze źródła, czyli paczki typu .tar.

Rozdział 7

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Tablica 7.1: Opis tabeli nad nią.

	metoda						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Rozdział 8

Podsumowanie i wnioski

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Dodatki

Spis skrótów i symboli

DNA	kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)
MVC	model – widok – kontroler (ang. *model–view–controller*)
N	liczebność zbioru danych
μ	stopnień przyleżności do zbioru
$\mathbb{E}$	zbiór krawędzi grafu
$\mathcal{L}$	transformata Laplace'a

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść do załącznika.

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła LaTeXowe i końcowa wersja w `pdf`),
- źródła programu,
- dane testowe.

Spis rysunków

Spis tablic