POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

KIERUNEK: AUTOMATYKA I ROBOTYKA

Praca dyplomowa inżynierska

System wizyjny do śledzenia ruchomych obiektów

autor: Szymon Ciemała

kierujący pracą: dr inż. Krzysztof Jaskot

konsultant: dr inż. Imię Nazwisko

Gliwice, styczeń 2022

Spis treści

Rozdział 1

Streszczenie

Praca inżynierska pod tytułem „System wizyjny do śledzenia ruchomych obiektów” 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 wybranymi palcami jako elementów sterujących w dowolnym projekcie napisanym w języku Python.

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

Całość projektu jest dostępna na platformie PyPi, która jest standardowym repozytorium paczek języka Python. Korzysta z niego menedżer paczek o nazwie **pip**, który jest programem automatycznie pobieranym podczas instalacji interpretera języka. Na głównej stronie paczki znajduje się dokumentacja oraz instrukcja instalacji i użytkowania.

Użytkownik paczki ma możliwość stworzenia własnych modeli rozpoznających gesty za pomocą interaktywnej instrukcji napisanej w Jupyter Notebook.

Rozdział 2

Wstęp

2.1 Wprowadzenie w problem

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

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 alfabetu języka migowego oraz podstawowych gestów (otwarta, zamknięta dłoń). Dodatkowo powinna pozwolić na wyznaczenie pozycji dłoni, jej typu oraz innych własności takich jak, odległości między końcówkami wybranych palców czy kąta obrotu dłoni.

2.3 Charakterystyka rozdziałów

- **Analiza tematu** – Opis założeń technicznych oraz funkcyjnych całości systemu.
- **Wymagania i narzędzia** – Wybrane narzędzia, w tym system operacyjny, biblioteki oraz programy.
- **Specyfikacja zewnętrzna** – Działanie oraz obsługa programu z perspektywy użytkownika.
- **Specyfikacja wewnętrzna** – Opis budowy całości paczki, wykorzystanych metod, struktur danych oraz algorytmów.
- **Podsumowanie** – Podsumowanie całości projektu.

Rozdział 3

Analiza tematu

3.1 Założenia

Wymagane możliwości oprogramowania można podzielić na trzy elementy:

- Łatwość użytkowania – Poprzez łatwość użytkowania można rozumieć oprogramowanie zapisane zgodnie z powszechnie stosowanymi standardami („Zen of Python") oraz przygotowaną dokumentację wraz z przykładami wykorzystania.
- Dostępność – Paczka powinna być dostępna poprzez menedżer **pip**, dzięki czemu użytkownik ma możliwość instalacji z repozytorium paczki przy pomocy jednej komendy.
- Możliwość dokładania własnych elementów – Użytkownik powinien mieć możliwość stworzenia własnego modelu rozpoznającego gesty, na przykład przy pomocy interaktywnej instrukcji.

3.2 Sformułowanie problemu

Stworzenie modułu będzie wymagało napisania klasy pozwalającej na rozpoznanie elementów charakterystycznych dłoni oraz przetworzenia obrazu. Obraz będzie pochodził z kamery internetowej, który zostanie odpowiednio przetworzony z wykorzystaniem funkcji dostępnych poprzez bibliotekę OpenCV. Przetworzony obraz zostanie wykorzystany przez metody biblioteki MediaPipe, która pozwoli na

rozpoznanie elementów charakterystycznych dłoni. W napisanej klasie zostaną zaimplementowane metody, które pozwolą na rozpoznanie typu dłoni (prawa, lewa), odległości między paliczkiem palca wskazującego oraz kciuka, oraz obrotu dłoni.

Kolejnym elementem jest wygenerowanie modelu matematycznego klasyfikujących gesty wykonywane przez dłonie. W tym celu zostanie wykorzystana biblioteka SciKit-Learn wraz z dostępnymi poprzez nią algorytmami ucznia maszynowego. Odpowiednio zebrane dane pozwolą na przeprowadzenie procesu uczenia dla kilku wybranych algorytmów.

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. Główny powodem, dla którego ten system został wybrany, jest fakt istnienia rozbudowanej społeczność, która pomaga w rozwiązywaniu problemów.

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. Zważając na fakt, że biblioteka jest wolnoźródłowa, GitHub może posłużyć jako system, który pozwala na rozbudowywanie projektu przez innych użytkowników lub tworzenia własnej i niezależnej wersji tego oprogramowania.

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 to 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 przez 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 źródłowym 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 [6] [4] 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 [3] [8], 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 [2] [7] 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.

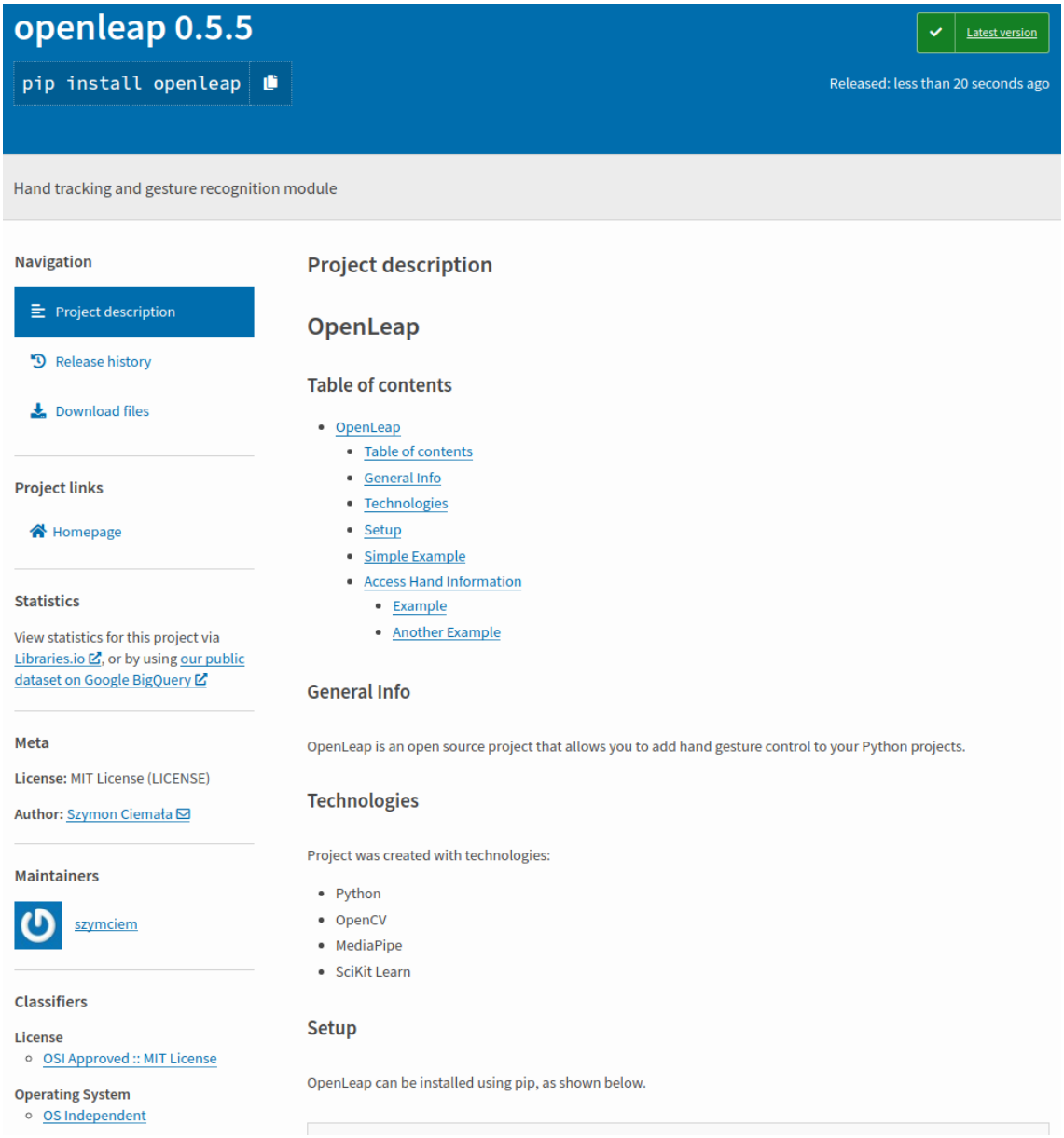
5.2 Instalacja paczki

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 [1] 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

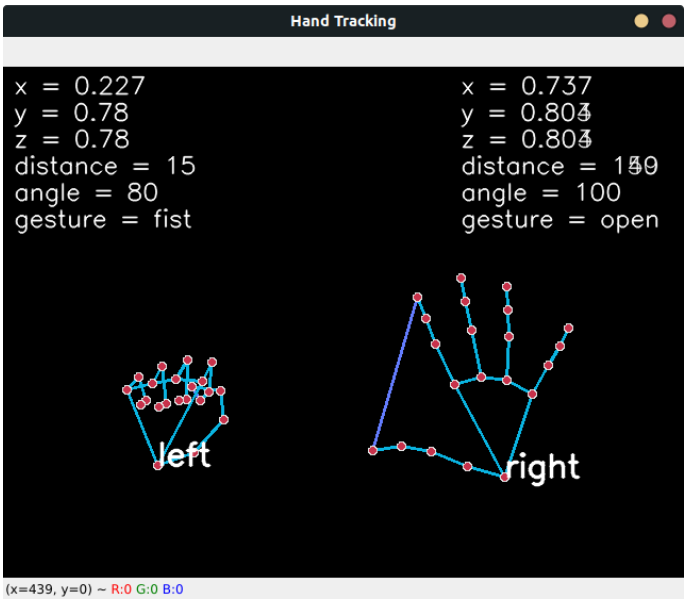
Alternatywą dla powyższego przykładu jest wykorzystanie metody **main()**, która nie wymaga wykorzystania wątku do obsługi kontrolera. Natomiast należy ją wywołać w pętli oraz zapisać warunek zamknięcia okna, jeśli takowe zostało wywołane. Do zamknięcia okna można posłużyć się wbudowanymi funkcjami: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

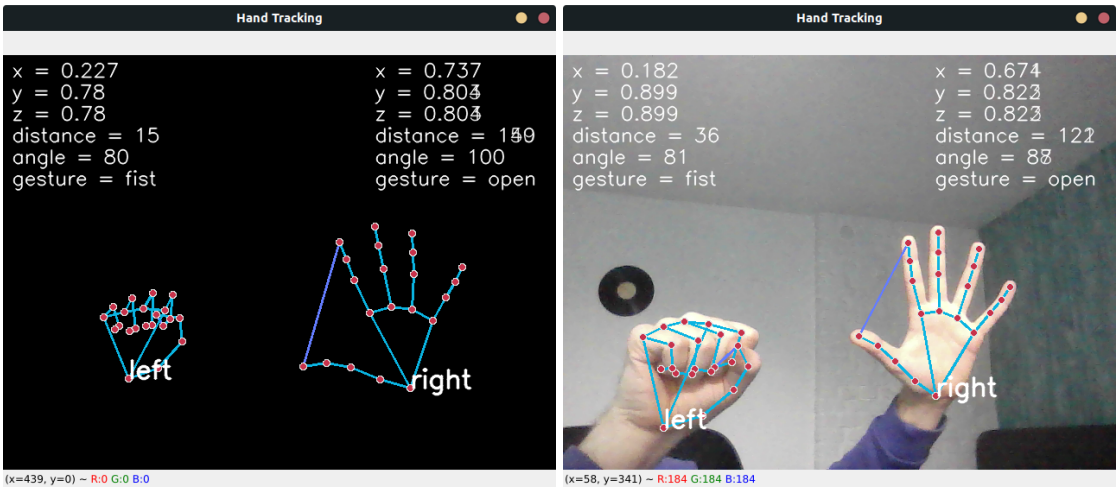
5.3.1 Parametry

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.



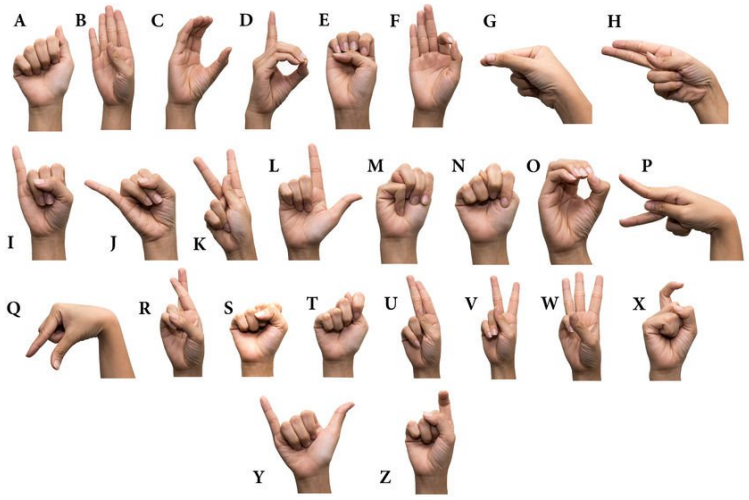
(a) Czarne tło. (b) Obraz z kamery.

Rysunek 5.3: Parametr określający typ tła.

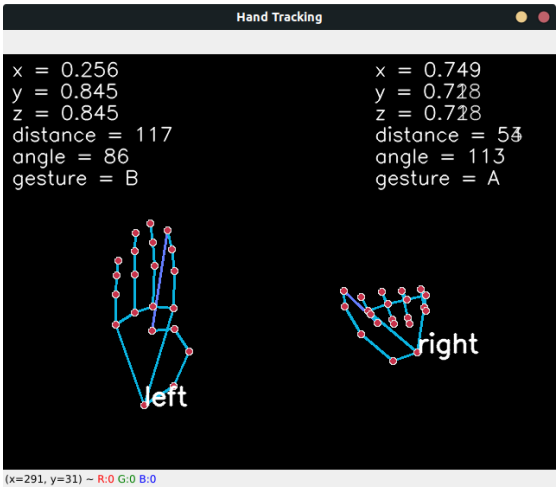
5.4 Przykłady użycia

5.4.1 Rozpoznawanie alfabetu w języku migowym

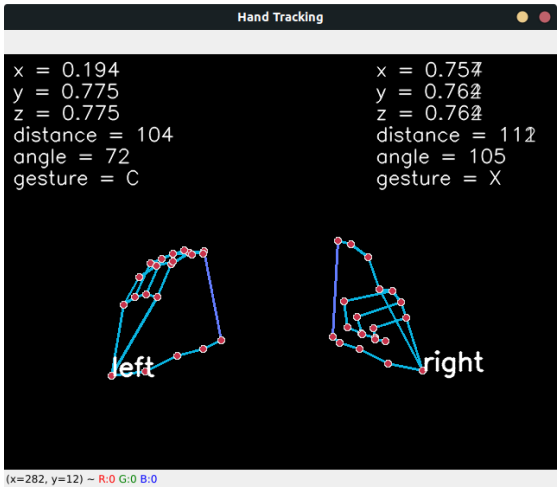
Pierwszym przykładem zastosowania jest wykorzystanie paczki do rozpoznawania alfabetu języka migowego. Taki program może umożliwić komunikację między osobą głuchoniemą posługującą się językiem migowym a osobą, która takiego języka nie zna. Przygotowany model rozpoznaje litery przedstawione na poniższej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



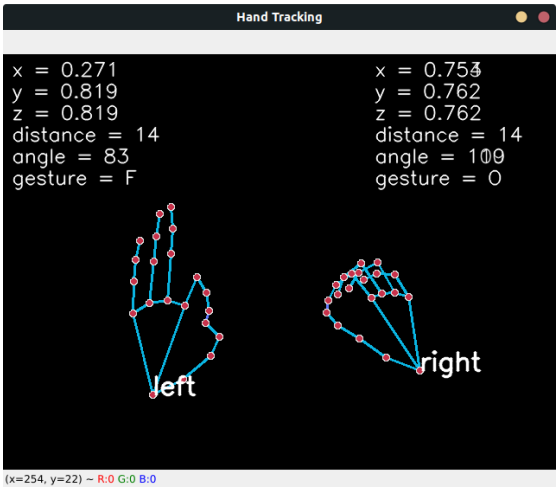
Rysunek 5.4: Gesty alfabetu języka migowego



(a) Litery A i B.



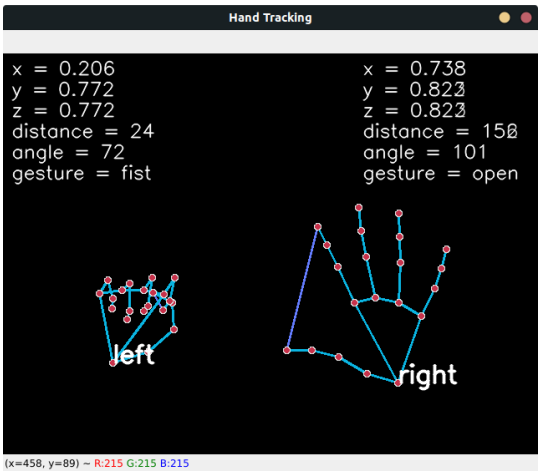
(b) Litery C i X.



(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.



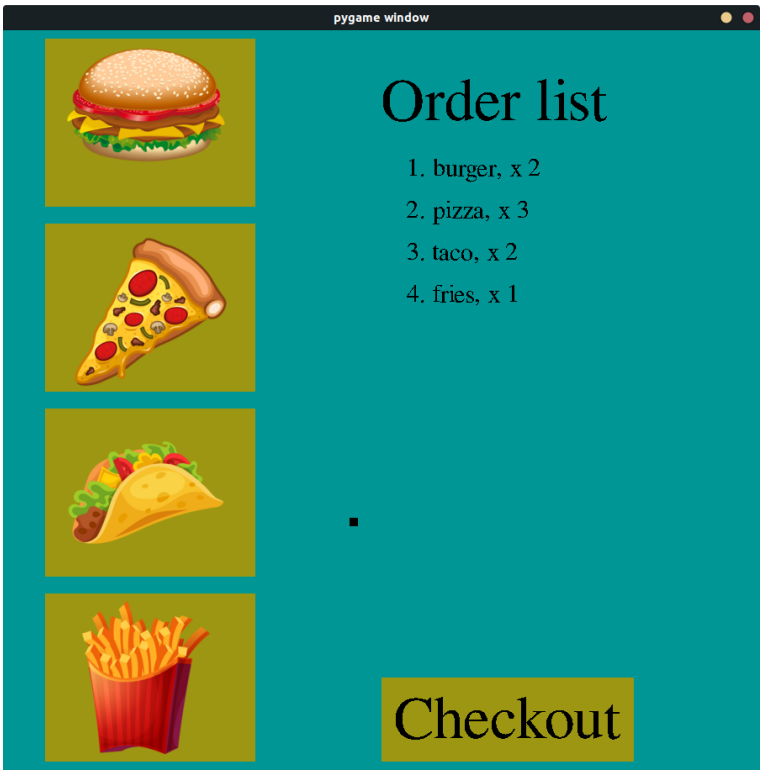
Rysunek 5.6: Gesty otwartej i zamkniętej dłoni

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.

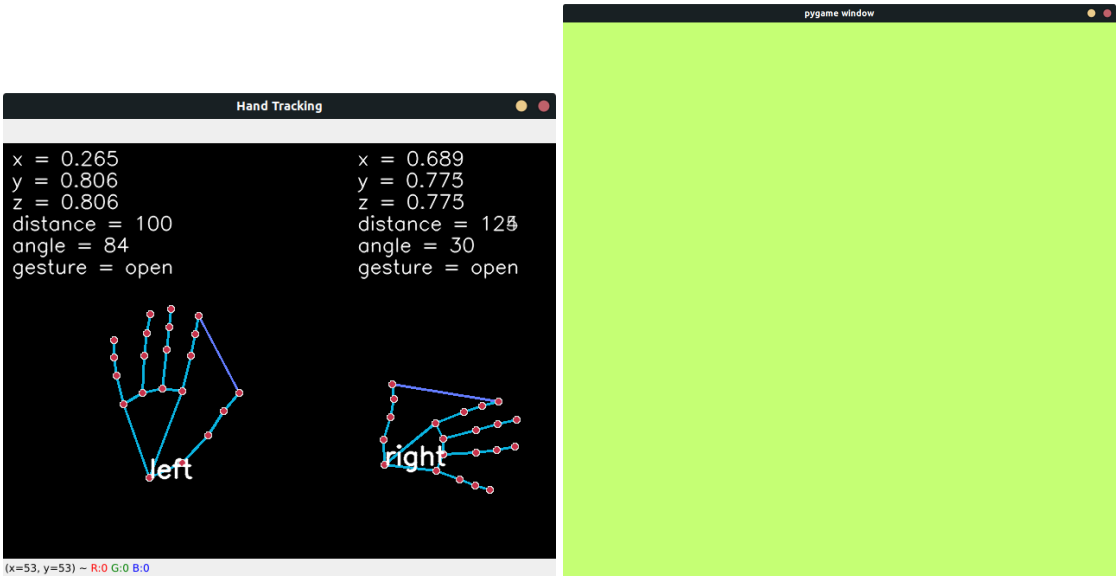


Rysunek 5.7: Zrzut ekranu kiosku interaktywnego

5.4.3 Dobór koloru

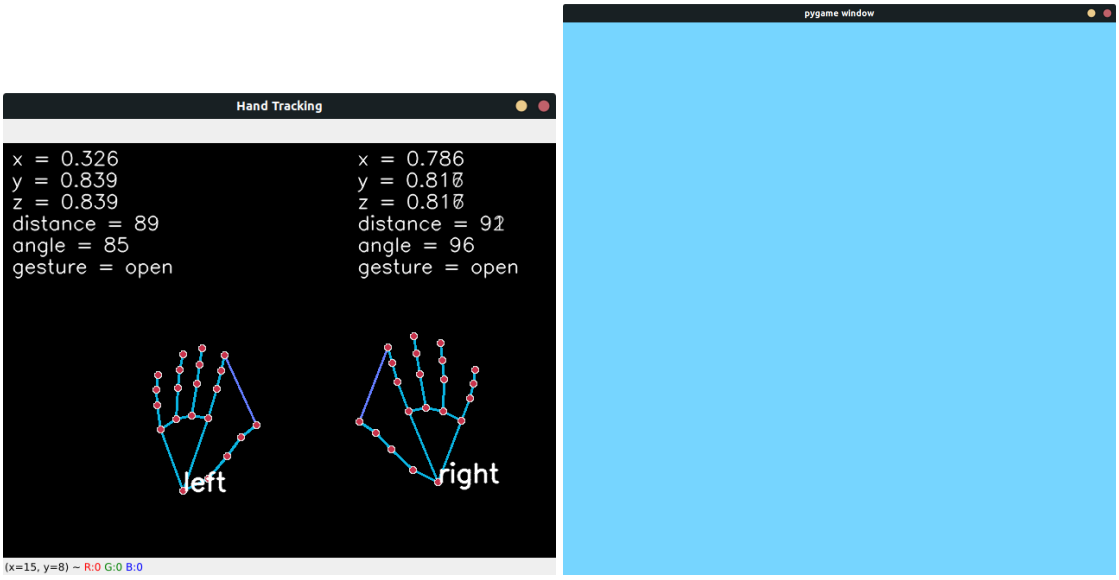
Ostatnim przykładem jest wykorzystanie dłoni jako kontrolera, za którego pomocą można wybrać dowolny kolor. Takie zastosowanie może zostać wykorzystane w pracy grafika komputerowego. Dzięki temu użytkownik będzie mógł zmieniać kolor wykorzystywanego narzędzia, na przykład przy malowaniu. Kolor można 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.



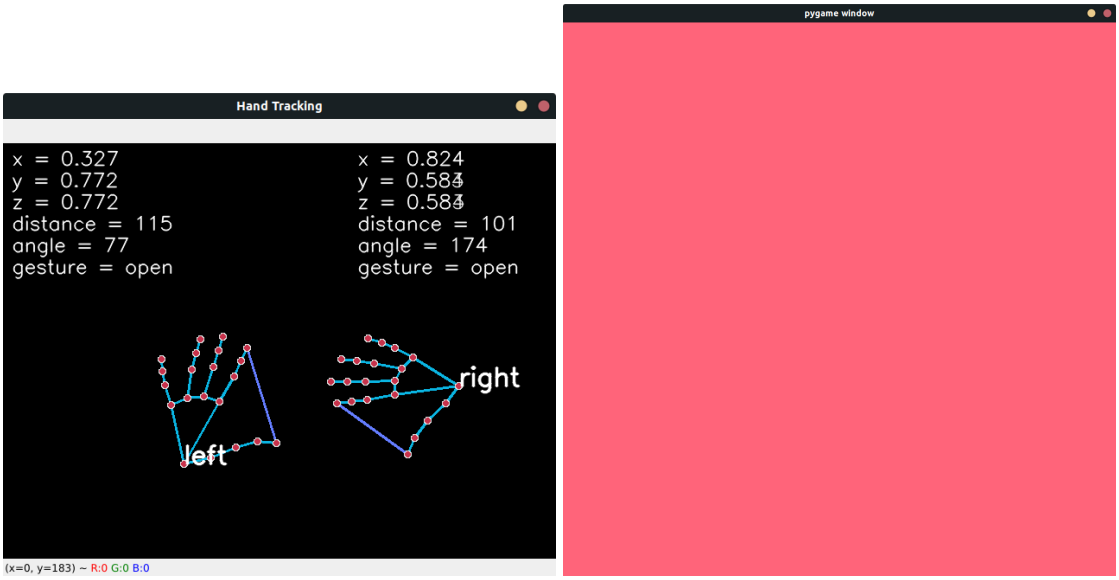
(a) Widok dłoni. (b) Dobrany kolor.

Rysunek 5.8: Obrót dłoni o 30 stopni.



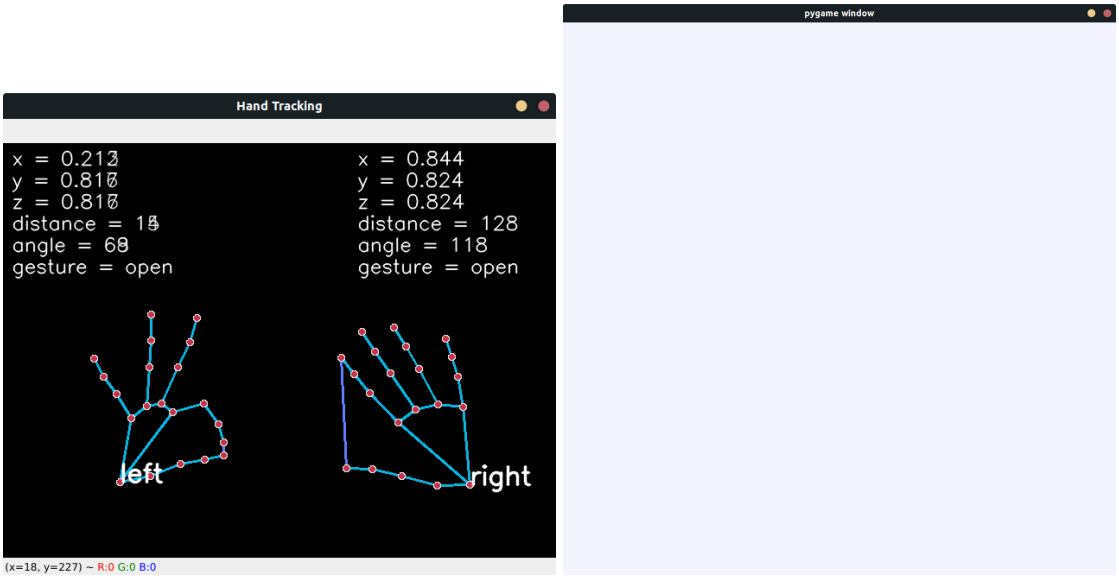
(a) Widok dłoni. (b) Dobrany kolor.

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

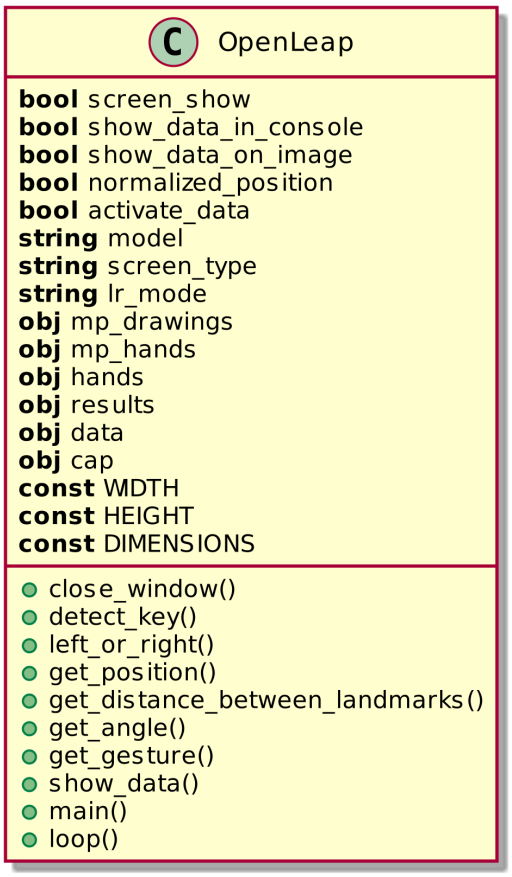
Klasa OpenLeap korzysta z różnych bibliotek. Natomiast nie dziedziczy ona żadnej klasy. Zostały jedynie wykorzystane wybrane metody tworzonych obiektów wewnątrz klasy.

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 rozpoznawanie 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.

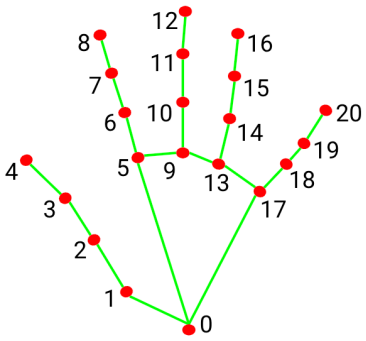
6.2 Struktury danych oraz pliki

Klasa zawiera w sobie parę struktur danych, które są kluczowe dla poprawnej pracy klasy oraz przechowywania informacji. Dobrze dobrane i zaprojektowane struktury danych pozwalają na pisanie programu, który jest przejrzysty i czytelny, na przykład poprzez wykorzystanie klasy **dataclass** zamiast tablicy czy słownika.

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 odczytuje informacje o danym elemencie. Ta struktura jest częścią biblioteki MediaPipe, więc jest już z góry zdefiniowana.

Pierwszym elementem określanym przez enumerator jest nadgarstek. Dalej znajduje się reszta elementów, oznaczająca resztę części palców i dłoni. Wszystkie elementy wraz z ich indeksami można zobaczyć na poniższej grafice 6.2. To właśnie wyznaczenie pozycji tych elementów jest najważniejszą częścią, za która jest odpowiedzialna biblioteka MediaPipe i na której bazuje biblioteka OpenLeap.



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

Przykład wykorzystania polega na odczytaniu współrzędnej X nadgarstka z listy o nazwie **landmark** będącej częścią obiektu **hand**. Jak można zobaczyć na przykładzie 6.1, wykorzystanie enumeratora pomaga w stworzeniu zapisu, który jest po prostu czytelny.

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

Listing 6.1: Przykładowe wykorzystanie **mp_hands**

6.2.2 Struktura results

Struktura **results** jest obiektem, który przechowuje informacje zwrócone przez metodę **hands.process()**, czyli dokładanie dane, które są wygenerowane przy rozpoznawaniu dłoni. Obiekt składa się z dwóch list, które przechowują dane ogólne o dłoniach widocznych w obrazie oraz o szczególne dotyczące każdej dłoni z osobna.

- **multi_hand_landmarks** – To pierwsza lista, która przechowuje dane o pozycjach wszystkich elementów danej dłoni. Odczyt danych wybranej dłoni wymaga podania jej indeksu. Indeksy punktów charakterystycznych można wybrać ze wcześniej wspomnianej struktury **mp_hands**.

```
wrist_x = self.results.multi_hand_landmarks[index].
    landmark[self.mp_hands.HandLandmark.WRIST].x
wrist_y = self.results.multi_hand_landmarks[index].
    landmark[self.mp_hands.HandLandmark.WRIST].y
```

Listing 6.2: Przykład korzystania z **multi_hand_landmarks**

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

Listing 6.3: Przykład elementu listy **multi_hand_landmarks**

- **multi_handedness** – Drugim typem jest lista przechowująca dane o wszystkich dłoniach widocznych na obrazie. Danymi są indeks, typ dłoni oraz punktacja określającą pewność poprawnego rozpoznania dłoni. Może posłużyć do sprawdzenia jaka jest liczba widocznych dłoni.

```
n_hands = len(self.results.multi_handedness)
```

Listing 6.4: Przykład korzystania z **multi_handedness**

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

Listing 6.5: Przykład elementu listy **multi_handedness**

6.2.3 Dataclass

Klasa typu **Dataclass** pozwala na stworzenie struktury podobnej do **struct** istniejącej w języku programowania **C**. Taka klasa pozwala na stworzenie obiektu składającego się jedynie z atrybutów wymaganych do opisu danych generowanych podczas rozpoznawania dłoni. Dodatkowym atutem takiej klasy jest możliwość prostego odczytu zapisanych danych, korzystając z operatora kropki, a nie z operatorów wykorzystywanych w odczycie danych ze słownika lub listy.

```
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
```

Listing 6.6: Struktura Data

W języku Python stworzenie klasy typu **dataclass** zaczyna się od zapisania dekoratora, który jest definiowany poprzez znak **@**. W tym wypadku nie jest wymagana funkcja inicjalizująca obiekt, czyli **___init___**. W definicji klasy zostały przypisane wartości początkowe tak jak w przykładzie powyżej. Dzięki czemu inicjalizacja nie wymaga podawania wartości początkowych, które i tak w chwili rozpoczęcia programu nie są potrzebne.

6.2.4 Słownik

Instancja klasy **Data** jest przypisana dla każdej dłoni (lewej oraz prawej) poprzez wykorzystanie słownika. Klucze słownika to „right" oraz „left".

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

Listing 6.7: Słownik przechowujący dane każdej z dłoni

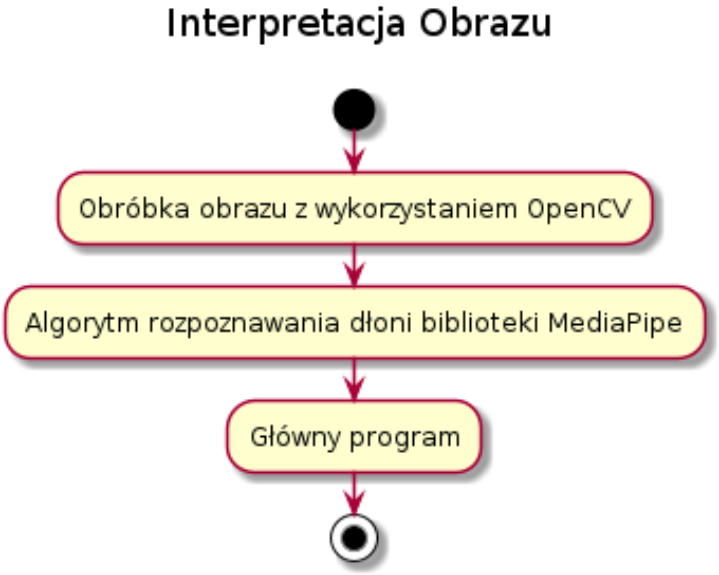
Taka konstrukcja pozwoli użytkownikowi w prosty i przejrzysty sposób na odnajdowanie potrzebnych wartości oraz informacji. Przykład odczytu informacji z tej struktury został już opisany w przykładzie 5.4

6.2.5 Pickle

Modele rozpoznające gesty muszą zostać zapisane do pliku, tak aby można było z nich skorzystać po ich wygenerowaniu. W języku Python standardowym narzędziem wykorzystywanym do zapisu zmiennych lub obiektów służy plik typu **pickle**, który pozwala na zapis tych elementów w postaci binarnej. Pozwala to na ponownym odczyt obiektu po restarcie programu. W przypadku modelu rozpoznających gesty będzie można je zapisać właśnie z rozszerzeniem **.pickle**, a później odczytać je w chwili inicjalizacji obiektu klasy OpenLeap.

6.3 Rozpoznawanie dłoni

Pierwszym elementem projektu jest rozpoznanie dłoni poprzez wyznaczenie 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

Obraz powinien 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 identyfikacja gestu czy obliczenia kąta obrotu dłoni.

6.3.1 OpenCV – przygotowanie obrazu z kamery

Na początku należy zainicjalizować obiekt obsługujący kamerę. Argumentem 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)
```

Listing 6.8: Wybór kamery

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

```
1     def main(self):
2         """
3         Main function that runs the core of the program.
4         """
5
6         hand_type = None
7         if self.cap.isOpened():
8             ret, frame = self.cap.read()
```

Listing 6.9: Przygotowanie funkcji głównej

Model rozpoznający dłoń korzysta z przestrzeni barw RGB (red, green, blue), a nie BGR (blue, green, red), która jest standardem w OpenCV. Dlatego należy przekształcić obraz na przestrzeń RGB. Dodatkowo obraz powinien zostać obrócony horyzontalnie, tak aby stworzył lustrzane odbicie względem użytkownika ustawionego naprzeciwko kamery. Tak ustawiony obraz będzie lepiej sprawdzał się przy sterowaniu.

```
1     #BGR to RGB
2     image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
3
4     #Flip horizontal
5     image = cv2.flip(image, 1)
```

Listing 6.10: Pierwsze przekształcenie

Flaga **writeable** ustawiona na **False** pozwala na uzyskanie lepszej wydajności przy przetwarzaniu obrazu w ramach modelu rozpoznającego dłonie. Na co wskazuje dokumentacja biblioteki MediaPipe. Przy pomocy metody **process** będącej częścią obiektu **hands** zostają wygenerowane dane zapisywane do struktury

results. Po tym etapie flaga **writeable** może zostać ustawiona z powrotem na wartość **True**.

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

Listing 6.11: Drugie przekształcenie

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

```
1     #RGB to BGR
2     image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

Listing 6.12: Powrót do przestrzeni BGR

6.4 Elementy charakterystyczne

Pozycje nadgarstka, paliczków oraz stawów dłoni zostaną wykorzystane do obliczenia kąta obrotu dłoni względem punktu 0, oraz do wytrenowania modeli uczenia maszynowego, których zadaniem będzie rozpoznawanie 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. Stworzenie oznaczeń polega na nałożeniu ich na obraz, który będzie wyświetlany w oknie. W tym wypadku jest to obraz o nazwie **background**, który przybiera formę czarnego tła lub obrazu z kamery w zależności od wybranej opcji.

```
1     self.mp_drawing.draw_landmarks(background,
2             hand,
3             self.mp_hands.HAND_CONNECTIONS,
4             self.mp_drawing.DrawingSpec(color=(75,50,200),
5                 thickness=2,
6                 circle_radius=2),
7             self.mp_drawing.DrawingSpec(color=(225,180,10),
8                 thickness=2,
9                 circle_radius=2))
```

Listing 6.13: Obrys dłoni

Tworzone oznaczenia istnieją w dwóch formach. W postaci kropek nałożonych na nadgarstek i stawy oraz linii, które je łączą.

6.5 Pomiary oraz inne ważne elementy

Dane zapisywane w obiektach typu **Data** są generowane poprzez odpowiednie metody, które wykorzystują między innymi punkty charakterystyczne.

6.5.1 Rozpoznawanie typu dłoni

Rozpoznawanie typu dłoni wykonuje się na dwa różne sposoby. Pierwszym bazującym na metodach wykorzystujących uczenie maszynowe oraz drugim, który wykorzystuje względną pozycję dłoni względem siebie.

- W pierwszym sposobie należy wyszukać w liście **multi_handedness** słownika, który posiada w sobie indeks sprawdzanej dłoni i w tym samym słowniku sprawdzić, jaki jest jej typ.

```
1  def left_or_right(self, index, mode='AI', hand=None):
2      label = 'right'
3          if mode == 'AI':
4
5              for idx, classification in enumerate(self.results
                   .multi_handedness):
6                  if classification.classification[0].index ==
                       index:
7
8                      label = classification.classification[0].
                           label.lower()
9
10                     return label
```

Listing 6.14: Określenie typu dłoni – Sposób nr 1

Jak się okazuje, nie jest to sposób, który daje zawsze poprawne rezultaty. Z tego powodu została stworzona alternatywny sposób wyznaczania typu dłoni.

- Poniższy sposób bazuje na założeniu, że lewa dłoń znajduje się na lewo od dłoni prawej, a prawa na prawo od lewej. Poprawność działania tego algorytmu wymaga jeszcze jednego założenia, że na ekranie powinny znajdować się dokładnie dwie dłonie, co nie zawsze może zostać spełnione. Można temu zapobiec poprzez wywołanie ponownie tej samej metody z wykorzystaniem pierwszego sposobu, kiedy na ekranie nie ma widocznych dwóch dłoni.

```
1      elif mode == 'position':
```

```
2               coords = np.array((hand.landmark[self.mp_hands.
                    HandLandmark.WRIST].x, hand.landmark[self.
                    mp_hands.HandLandmark.WRIST].y))
3
4               #Get x values from both hands and compare
5               if len(self.results.multi_handedness) >= 2:
6                   for i in [0, 1]:
7                       if index == i:
8                           another_hand_x = self.results.
                                multi_hand_landmarks[1-index].
                                landmark[self.mp_hands.
                                HandLandmark.WRIST].x
9                           if coords[index] > another_hand_x:
10                              label='right'
11                          else:
12                              label='left'
13
14                          return label
15              else:
16                  return self.left_or_right(index, mode='AI',
                        hand=hand)
```

Listing 6.15: Określenie typu dłoni – Sposób nr 2

6.5.2 Pozycja elementów

Metoda zwraca zwykłą lub znormalizowaną pozycję wybranego elementu dłoni. W dokumentacji można znaleźć informację, że normalizacja współrzędnej Z jest wykonywana względem szerokości obrazu tak jak współrzędnej X. Stąd też w przeliczeniu wartości znormalizowanych na zwykłe należy przemnożyć współrzędną Z przez szerokość obrazu tak jak współrzędną X.

```
1 def get_position(self, index=0, landmark_idx=1, normalized=False):
2     x = self.results.multi_hand_landmarks[index].landmark[
          landmark_idx].x
3     y = self.results.multi_hand_landmarks[index].landmark[
          landmark_idx].y
```

```
4       z = self.results.multi_hand_landmarks[index].landmark[
            landmark_idx].y
5
6       if normalized:
7           #Choose proper index instead of fixed one (idx=0)
8           return x, y, z
9       else:
10          x = int(x*self.WIDTH)
11          y = int(y*self.HEIGHT)
12          z = int(z*self.WIDTH)
13      return x, y, z
```

Listing 6.16: Pobranie pozycji

6.5.3 Odległość między punktami

Odległość jest obliczana ze standardowego wzoru na odległość między dwoma punktami w wybranym układzie współrzędnych.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{6.1}$$

Użytkownik ma możliwość wyboru czy wartość ma być liczona za pomocą zmiennych znormalizowanych, czy nie.

```
1   def get_distance_bettween_landmarks(self, index, landmark_1,
        landmark_2, normalized=True):
2       if normalized:
3               x1, y1, z1 = self.get_position(index, landmark_1,
                    normalized=True)
4               x2, y2, z2 = self.get_position(index, landmark_2,
                    normalized=True)
5           else:
6               x1, y1, z1 = self.get_position(index, landmark_1)
7               x2, y2, z2 = self.get_position(index, landmark_2)
8
9           distance = math.sqrt(((x1-x2)**2 + (y1-y2)**2))
10
```

```
11          return distance
```

Listing 6.17: Obliczenie pozycji punktu charakterystycznego

6.5.4 Obrót dłoni

Obrót dłoni jest obliczany jako kąt obrotu ramienia, którego początek znajduje się w punkcie nadgarstka, a drugi koniec jest wybranym dowolnym punktem dłoni. Wykorzystana jest do tego funkcja **atan2()**, która za argumenty bierze współrzędne końca ramienia w układzie współrzędnych, którego środkiem jest nadgarstek. Dodatkowe argumenty funkcji **get_angle()** określają zakres kąta oraz jego jednostkę.

```
1  def get_angle(self, index, landmark_idx, mode='half', unit='
       radians'):
2      angle=0
3
4      wrist_x = self.results.multi_hand_landmarks[index].landmark[
           self.mp_hands.HandLandmark.WRIST].x
5      wrist_y = self.results.multi_hand_landmarks[index].landmark[
           self.mp_hands.HandLandmark.WRIST].y
6
7      landmark_x = self.results.multi_hand_landmarks[index].landmark
           [landmark_idx].x
8      landmark_y = self.results.multi_hand_landmarks[index].landmark
           [landmark_idx].y
9
10     realitive_x = landmark_x - wrist_x
11     realitive_y = landmark_y - wrist_y
12
13     angle = math.atan2(realitive_y, realitive_x)
14
15     if mode=='half' and angle>0: angle=0
16
17     if unit=='degree':
18         angle = 180*abs(angle)/math.pi
19
```

```
20      return angle
```

Listing 6.18: Obliczenie odległości między wybrany punktami

6.6 Rozpoznawanie gestów

Modele pozwalają na rozpoznanie gestów na podstawie pozycji elementów dłoni. Aby było to możliwe, należy wytrenować modele na podstawie odpowiednio przygotowanych danych, czyli takich, które zależą jedynie od ułożenia dłoni. Należy również skorzystać z wcześniej opisanego pliku **pickle** tak, żeby model mógł zostać wczytany ponownie do programu.

6.6.1 Mechanizm załadowania modelu

Argumentem inicjalizatora klasy jest między innymi typ modelu rozpoznającego gesty lub ścieżka do tego pliku. Po ustaleniu odpowiedniej ścieżki należy wczytać plik jako plik binarny.

```
1 with open(data_path, 'rb') as f:
2     self.gesture_model = pickle.load(f)
```

Listing 6.19: Wczytanie modelu

6.6.2 Wykorzystanie modelu

Do rozpoznania gestu została przygotowana funkcja, która zwraca nazwę rozpoznanego gestu. W poniższym przykładzie znajduje się jedynie linia o numerze 9, która wykorzystuje sam model. Metoda przeprowadza jeszcze odpowiednie przekształcenia danych, które zostaną opisane w podsekcji opisującej proces uczenia algorytmów.

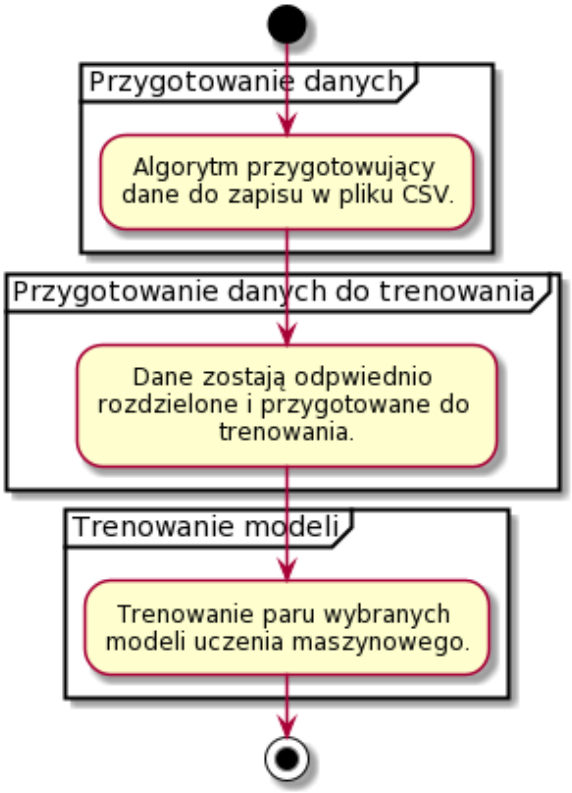
```
1 def get_gesture(self, index):
2
3     .
```

```
4       .
5       .
6
7       with warnings.catch_warnings():
8           warnings.filterwarnings("ignore")
9           gesture_class = self.gesture_model.predict(x)[0]
10
11      return gesture_class
```

Listing 6.20: Funkcja zwracająca rozpoznany gest

6.7 Przygotowanie modeli uczenia maszynowego

Zadaniem instrukcji uczącej będzie stworzenie modeli matematycznych przy pomocy algorytmów uczenia maszynowego, których celem jest rozpoznawanie gestów dłoni. Proces tworzenie takiego modelu można podzielić na trzy kroki przedstawione na schemacie 6.4.



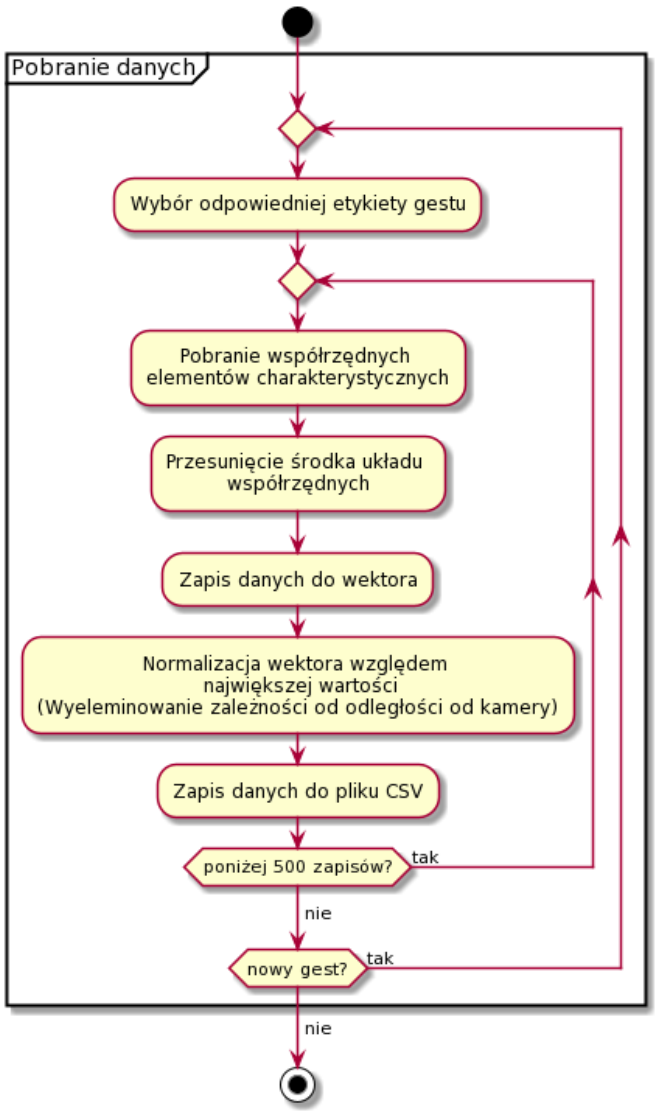
Rysunek 6.4: Ogólny algorytm przygotowania modeli uczenia maszynowego

Całość programu została napisana osobno 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, co definiuje osoba trenująca w czasie działania programu.

Opisywany program może również pełnić funkcję instrukcji, o czym był mowa w podsekcji 5.5.

6.7.1 Zebranie danych

Algorytm rozpoznający gesty powinien zostać wytrenowany na podstawie zbiorów współrzędnych punktów charakterystycznych i przypisanych do nich nazw gestów. Za nim to jednak nastąpi, należy zauważyć parę trudności z tym związanych. Podstawowym problemem jest fakt, że pozycje elementów charakterystycznych są opisane względem układu współrzędnych (można go uznać za układ bezwzględny), którego środek znajduje się w lewym górnym rogu obrazu pobranego z kamery. Wynika, więc z tego, że skorzystanie z tych współrzędnych jako danych uczących, skutkowałoby tym, że model klasyfikowałby gest również na podstawie jego pozycji na obrazie. W takim wypadku należy przeprowadzić transformację, tutaj akurat przesunięcie układu współrzędnych do pozycji nadgarstka (układ względny).



Rysunek 6.5: Ogólny algorytm przygotowania modeli uczenia maszynowego

Współrzędne w układzie bezwzględnym opisujące pozycje punktów można zapisać w poniższej macierzy. Liczby przy symbolach są równoważne z indeksami tablicy opisującej punkty, rys. 6.2.

$$M_G = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ & \vdots & \\ x_{21} & y_{21} & z_{21} \end{bmatrix} \tag{6.2}$$

Rysunek 6.6: Pełna postać macierzy

Przesunięcie układu współrzędnych można opisać poprzez macierz transformacji, gdzie wartości x_0, y_0, z_0 to współrzędne nadgarstka w bezwzględnym układzie współrzędnych.

$$M_p = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

Rysunek 6.7: Macierz transformacji ogólna

Tak naprawdę, przesunięcie układu współrzędnych wzdłuż osi Z nie będzie miało miejsca. Wartości współrzędnej Z wszystkich elementów są obliczane względem pozycji nadgarstka, co oznacza, że współrzędna Z nadgarstka wynosi zawsze 0. W takim wypadku można ją usunąć z macierzy przesunięcia. Ostatecznie macierz ma postać:

$$M_p = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

Rysunek 6.8: Macierz transformacji szczególna

Po przekształceniu macierzy współrzędne punktów są następującej postaci. Jak można zauważyć, współrzędne nadgarstka zostały kompletnie pominięte. Po

operacji przesunięcia ich wartości zostały wyzerowane, więc nie zostały wpisane ponownie do macierzy.

$$M_G = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ & \vdots & \\ x'_{21} & y'_{21} & z'_{21} \end{bmatrix} \tag{6.5}$$

Rysunek 6.9: Pełna postać macierzy

Aby dane mogły zostać poprawnie zinterpretowane przez algorytmy uczenia maszynowego, muszą one zostać przedstawione w postaci jednowymiarowej. Macierz zostaje przekształcona do postaci następującego wektora.

$$A_f = \begin{bmatrix} x'_1 & y'_1 & z'_1 & x'_2 & y'_2 & z'_2 & \cdots & x'_{21} & y'_{21} & z'_{21} \end{bmatrix} \tag{6.6}$$

Rysunek 6.10: Wektor danych

Kolejnym krokiem jest uniezależnienie pozycji elementów od odległości dłoni od kamery. Rozwiązaniem jest normalizacja wektora danych względem największej bezwzględnej wartości.

$$A_n = \frac{A_f}{max(abs(A_f))} \tag{6.7}$$

Rysunek 6.11: normalizacja wektora danych

Każdy nowy wektor zostaje zapisany do pliku CSV z odpowiednią etykietą. Tak zebrane i przetworzone dane posłużą do wytrenowania algorytmów uczenia maszynowego.

6.7.2 Praktyczne wykorzystanie

Do programu zostają pobrane pozycje wszystkich elementów. Później zostaje stworzona pusta tablica o wymiarach 20x3, która zostaje zapisana współrzędnymi po operacji przesunięcia.

```
1     hand_landmarks = results.multi_hand_landmarks[0].landmark
2     wrist = hand_landmarks[0]
3
4     hand_landmarks_row = np.zeros((20,3))
5     for i in range(1, len(hand_landmarks)):
6         hand_landmarks_row[i-1]=[hand_landmarks[i].x-wrist.x,
7                                  hand_landmarks[i].y-wrist.y,
8                                  hand_landmarks[i].z-wrist.z]
```

Listing 6.21: Funkcja zwracająca rozpoznany gest

Kolejnym krokiem jest przekształcenie macierzy na wektor (funkcja **flatten()**) oraz znormalizowanie tego wektora. Ostatecznie wektor zostaje zapisany do pliku CSV wraz z odpowiednią etykietą.

```
1     hand_landmarks_row = hand_landmarks_row.flatten()
2     hand_landmarks_row = list(hand_landmarks_row/np.max(np.
         absolute(hand_landmarks_row)))
3
4     hand_landmarks_row.insert(0, class_name)
5
6     with open(FILE_NAME, mode='a', newline='') as f:
7         csv_writer = csv.writer(f,
8                                 delimiter=',',
9                                 quotechar='"',
10                                quoting=csv.QUOTE_MINIMAL)
11
12        csv_writer.writerow(hand_landmarks_row)
```

Listing 6.22: Funkcja zwracająca rozpoznany gest

6.7.3 Budowa pliku CSV

Dane zapisane w pliku CSV opisują przykładowe współrzędne wszystkich elementów dłoni wraz z przypisaną etykietą oznaczającą gest. Poniżej zostały przedstawione przykładowe dane trenujące dla modelu podstawowego, czyli rozpoznającego otwartą i zamkniętą dłoń.

Początek

class	x1	y1	z1	...	x20	y20	z20
open	-0.145	-0.031	-0.049	...	0.132	-0.836	-0.220
open	-0.151	-0.068	-0.026	...	0.146	-0.832	-0.245
open	-0.154	-0.072	-0.026	...	0.1335	-0.843	-0.229
open	-0.150	-0.071	-0.024	...	0.132	-0.836	-0.220

Tablica 6.1: Początek pliku

Koniec

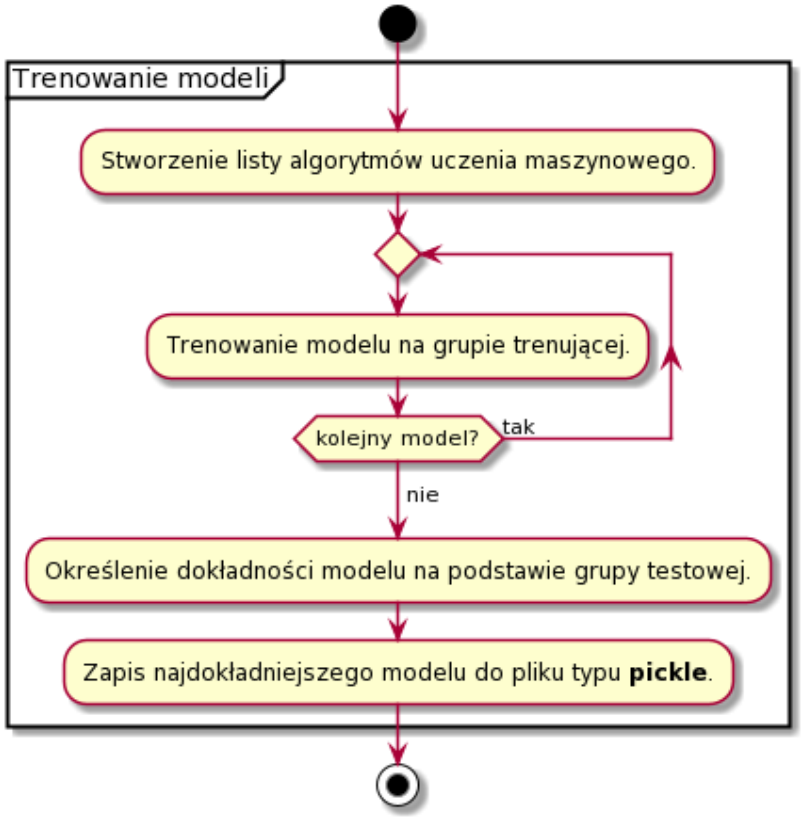
class	x1	y1	z1	...	x20	y20	z20
fist	0.417	0.088	-0.096	...	-0.029	-0.301	-0.617
fist	0.410	0.088	-0.095	...	-0.080	-0.289	-0.633
fist	0.354	0.042	-0.087	...	-0.014	-0.351	-0.495
fist	0.299	0.037	-0.101	...	-0.017	-0.445	-0.339

Tablica 6.2: Koniec pliku

6.7.4 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 drugim kroku należy te dwie grupy podzielić na grupę trenującą i grupę testową. Zadaniem grupy testowej będzie trenowanie wybranych modeli, a grupy testowej przetestowanie ich dokładności.

Poniższy schemat przedstawia działanie części wykorzystującej wiele algorytmów uczenia maszynowego do wyznaczenia modelu o największej poprawności działania.



Rysunek 6.12: Ogólny algorytm przygotowania modeli uczenia maszynowego

W celu wybrania najlepszej metody klasyfikacji zostanie wybranych kilka algorytmów. Każdy z nich stworzy swój model, a ostatecznie zostanie sprawdzona

ich poprawności z wykorzystaniem grupy testowej. Model z najlepszym wynikiem zostanie zapisany do pliku typu **pickle**.

6.7.5 Wybrane algorytmy klasyfikujące

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

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

Algorytmy można zapisać w słowniku, który później pozwoli na wykorzystanie ich w pętli.

```
pipelines = {
'lr':make_pipeline(StandardScaler(),LogisticRegression()),
'nc':make_pipeline(StandardScaler(),NearestCentroid()),
'dt':make_pipeline(StandardScaler(),DecisionTreeClassifier()),
'rd':make_pipeline(StandardScaler(),RidgeClassifier()),
'rf':make_pipeline(StandardScaler(),RandomForestClassifier()),
'gd':make_pipeline(StandardScaler(),SGDClassifier()),
'gb':make_pipeline(StandardScaler(),GradientBoostingClassifier()),
'nn':make_pipeline(StandardScaler(),MLPClassifier()),
}
```

Listing 6.23: Funkcja zwracająca rozpoznany gest

Pipline pozwala na zautomatyzowanie pewnych procesów związanych z uczeniem. Tutaj, na przykład zostaje wykorzystane funkcja **StandardScaler()**, która odpowiednio standaryzuje dane wejściowe.

```
models = {}
for algorithm, pipeline in pipelines.items():
    models[algorithm] = pipeline.fit(x_train, y_train)
```

Listing 6.24: Trenowanie modeli

6.7.6 Badanie dokładności każdego z algorytmów

Badanie poprawności działania każdego z algorytmów wykonuje się poprzez funkcję **accuracy_score()**, która za argumenty przyjmuje dane wyjściowe odczytane z pliku CSV oraz odpowiedzi wygenerowane przez modele. Na podstawie porównania tych danych w prosty sposób można obliczyć poprawność działania danego algorytmu.

```
for algorithm, model in models.items():
    y_predicted = model.predict(x_test)
    print(f'{algorithm}, {round(accuracy_score(y_test, y_predicted
        )*100,2)}%')
```

Listing 6.25: Sprawdzenie poprawności

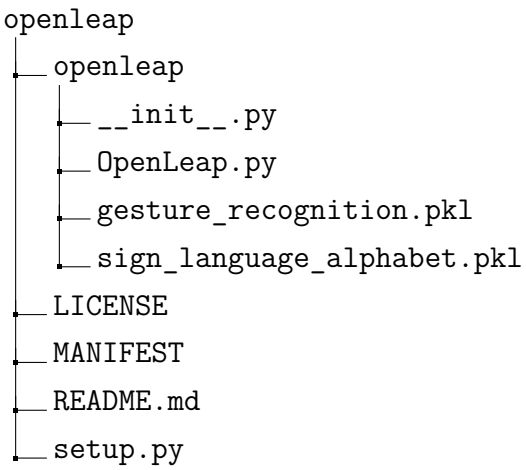
6.8 Paczka PyPi

6.8.1 Budowa paczki

Ostatecznym krokiem jest przygotowanie programu w formie paczki, która zostanie udostępniona na platformie PyPi. Wymagania to przygotowania odpowiednich plików konfiguracyjnych oraz zastosowania narzędzi do stworzenia pliku **wheel** i **tar**.

6.8.2 Struktura paczki

Na początku należy przygotować odpowiednią strukturę paczki, tak jak jest to pokazane na rysunku 6.13. W tym folderze znajdują się wszystkie potrzebne



Rysunek 6.13: Struktura paczki PyPi

elementy paczki. W podfolderze o tej samej nazwie znajduje się główna części modułu, czyli pliki z rozszerzeniem **.py**, **___init___.py** oraz **OpenLeap.py**. Dodatkowo w tym folderze znajdują się pliki typu **pickle**, w których zapisane są modele rozpoznające gesty.

6.8.3 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. Przykładowy plik **setup.py** znajduje się poniżej.

```
1    from setuptools import find_packages, setup
2
3    with open("../README.md", "r") as fh:
4        long_description = fh.read()
5
6    setup(
7        name='openleap',
8        version='0.5.06',
9        author='Szymon Ciemala',
10       author_email='szymciem@protonmail.com',
```

```
11          long_description=long_description,
12          long_description_content_type="text/markdown",
13          url="https://github.com/szymciem8/OpenLeap",
14          description='Hand tracking and gesture recognition module'
                ,
15          py_modules=['OpenLeap'],
16          classifiers=[
17               "Programming Language :: Python :: 3",
18               "License :: OSI Approved :: MIT License",
19               "Operating System :: OS Independent",
20           ],
21          package_data={'openleap':['*.pkl']},
22          packages=['openleap'],
23
24          license='LICENSE',
25
26          install_requires= [
27              "mediapipe ~= 0.8.8",
28              "opencv-python ~= 4.5.3.56",
29              "pandas ~= 1.3.4"
30          ],
31      )
```

Listing 6.26: Sprawdzenie poprawności

6.8.4 Załadowanie paczki do repozytorium

Przed załadowaniem paczki do repozytorium, należy stworzyć archiwum z plikami źródłowymi, na przykład typu **.tar** oraz plik typu **WHEEL**. Oba pliki spełniają tę samą funkcję, czyli przechowywanie niezbędnych elementów paczki oraz umożliwienie instalacji na systemie użytkownika.

Do stworzenia paczki z plikami źródłowymi służą komendy **bdist_wheel**, który tworzy plik **WHEEL** oraz **sdist**, który tworzy archiwum.

```
python3 setup.py bdist_wheel sdist
```

Listing 6.27: Przygotowanie plików źródłowych

Do załadowania całości na platformę PyPi wykorzystuje się program **twine**. Dodatkowo można wykorzystać flagę **skip-existing**, która pozwala na pominięcie wysłania istniejących już gotowych paczek.

```
twine upload --skip-existing dist\*
```

Listing 6.28: Wysłanie paczki na repozytorium

Program wymaga autoryzacji, czyli podania nazwy użytkownika i hasła do konta stworzonego na platformie PyPi.

Rozdział 7

Podsumowanie i wnioski

Podsumowując, celem pracy było stworzenie biblioteki, która pozwoli na rozszerzenie aplikacji pisanych w języku Python o możliwość sterowania dłońmi. Biblioteka jest o otwartym kodzie źródłowym, co oznacza, że można z niej bezpłatnie korzystać oraz każdy może dołączyć do jej rozbudowywania.

Bibliotekę można ulepszyć poprzez napisanie jej w języku C++, który jest kompilowany. Pozwoliłoby to na zwiększenie wydajności oprogramowania. Język Python zezwala na korzystanie z modułów napisanych właśnie w języku C i C++, więc jest to język jak najbardziej kompatybilny.

Bibliografia

[1] Dokumentacja OpenLeap na PyPi.
`https://pypi.org/project/openleap/`.

[2] Dokumentacja SciKit-Learn.
`https://scikit-learn.org/stable/index.html`.

[3] Dokumentcja MediaPipe.
`https://google.github.io/mediapipe/solutions/hands.html`.

[4] Dokumentcja OpenCV.
`https://docs.opencv.org/3.4/`.

[5] SciKit-Learn Machine Learning Algorithms.
`https://scikit-learn.org/stable/supervised_learning.html`.

[6] Gary Bradski, Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. Ó'Reilly Media, Inc.", 2008.

[7] Raúl Garreta, Guillermo Moncecchi, Trent Hauck, Gavin Hackeling. *Scikit-learn: machine learning simplified: implement scikit-learn into every step of the data science pipeline*. Packt Publishing Ltd, 2017.

[8] Indriani, Moh. Harris, Ali Suryaperdana Agoes. Applying hand gesture recognition for user guide application using mediapipe. *Proceedings of the 2nd International Seminar of Science and Applied Technology (ISSAT 2021)*, strony 101–108. Atlantis Press, 2021.

Dodatki

Spis rysunków

Spis tablic