



Politechnika
Śląska

POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI
KIERUNEK: AUTOMATYKA I ROBOTYKA

Praca dyplomowa inżynierska

Tytuł pracy dyplomowej inżynierskiej

autor: Szymon Ciemala

kierujący pracą: dr inż. Krzysztof Jaskot

konsultant: dr inż. Imię Nazwisko

Gliwice, listopad 2021

Spis treści

1	Streszczenie	1
	Streszczenie	1
2	Wstęp	3
2.1	Wprowadzenie w problem	4
2.2	Cel pracy	4
2.3	Osadzenie problemu w dziedzinie	4
2.4	Charakterystyka rozdziałów	4
3	[Analiza tematu]	5
3.1	Sformułowanie problemu	5
4	Wymagania i narzędzia	7
4.1	Rozpoznawanie dłoni	7
4.1.1	OpenCV - przygotowanie obrazu z kamery	7
4.1.2	MediaPipe - Elementy charakterystyczne	9
4.1.3	Generowanie grafiki dłoni	10
4.2	SciKit Learn - uczenie maszynowe	10
4.2.1	Budowa programu	10
4.2.2	Zebrańie danych	10
4.2.3	Metody klasyfikacji - uczenie maszynowe	11
4.2.4	Ponowne wykorzystanie modelu	12
4.3	Paczka PyPi	12
4.3.1	Budowa paczki	12
4.3.2	Załadowanie paczki do repozytorium	12
5	[Właściwy dla kierunku - np. Specyfikacja zewnętrzna]	13
6	[Właściwy dla kierunku - np.Specyfikacja wewnętrzna]	15
7	Weryfikacja i walidacja	17

8 Podsumowanie i wnioski	19
Bibliografia	19
Spis skrótów i symboli	23
Źródła	25
Zawartość dołączonej płyty	29

Rozdział 1

Streszczenie

Praca inżynierska Żozpoznawanie 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 oraz pozycji dłoni 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 Wprowadzenie w problem

Rozwój technologii 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 sposób 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 pozwolić na wyznaczenie pozycji dłoni, jej typu oraz wartości charakterystycznych, na przykład odległości 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 - biblioteka MediaPipe.

2.4 Charakterystyka rozdziałów

Rozdział 3

[Analiza tematu]

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

3.1 Sformułowanie problemu

Cele projektowe można podzielić na trzy części.

- Wyznaczenie pozycji 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 platformę PyPi.

Stworzenie modułu będzie wymagało napisania klasy wykorzystującej odpowiednie biblioteki pozwalających 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 matematycznych klasyfikujących gesty dłoni. W tym celu zostanie wykorzystana biblioteka SciKit Learn wraz z dostępnymi poprzez nią algorytmami uczenia maszynowego. Odpowiednio zebrane dane pozwolą na przeprowadzenie procesu uczenia 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.

Rozdział 4

Wymagania i narzędzia

- wymagania funkcjonalne i нефункционалне
- przypadki użycia (diagramy UML) - dla prac, w których mają zastosowanie
- opis narzędzi, metod eksperymentalnych, metod modelowania itp.
- metodyka pracy nad projektowaniem i implementacją - dla prac, w których ma to zastosowanie

4.1 Rozpoznawanie dłoni

Pierwszym elementem projektu jest rozpoznanie dłoni poprzez wyznaczenie pozycji elementów charakterystycznych. Pozycja każdego z tych elementów jest względna według pozycji nadgarstka. ????

4.1.1 OpenCV - przygotowanie obrazu z kamery

Poprawne działanie modelu MediaPipe wymaga odpowiedniego przygotowania obrazu kamery. Działanie kontrolera odbywa się poprzez główną metodę **main()**.

Metoda **main()** jest główną funkcją, w której dokonywane są obliczenia oraz przekształcenia pozwalające na obliczenie obrotu dłonie, odległości między wybranymi palcami oraz na wykrycie gestu.

```
1      #Initiate camera
2      self.cap = cv2.VideoCapture(0)
```

W pierwszym kroku tworzymy instancję klasy **VideoCapture** biblioteki **OpenCV**, która pozwoli na odczytywanie obrazu kamery.

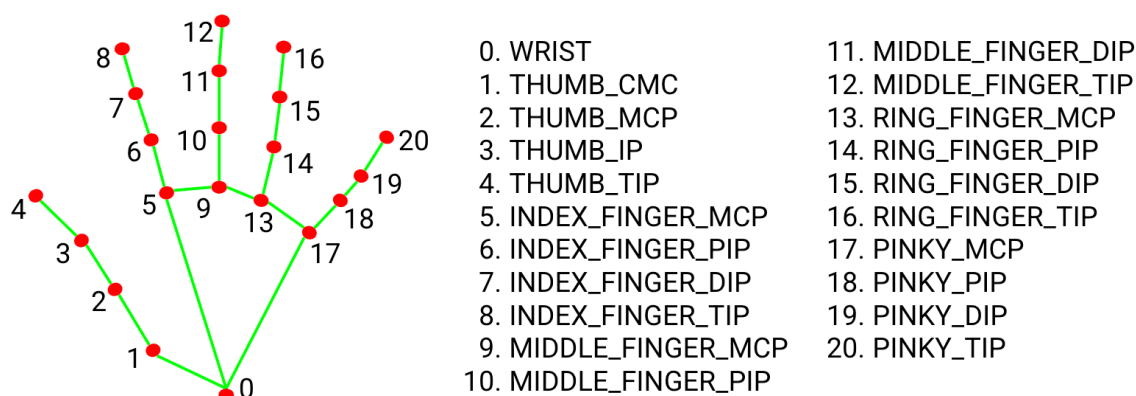
```
1      def main(self):
2          """
3          Main function that runs the core of the program.
4          """
5
6          hand_type = None
7          if self.cap.isOpened():
8              ret, frame = self.cap.read()
9
10             #BGR to RGB
11             image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
12
13             #Flip horizontal
14             image = cv2.flip(image, 1)
15
16             #Set flag
17             image.flags.writeable = False
18
19             #Detections
20             self.results = self.hands.process(image)
21
22             #Set flag back to True
23             image.flags.writeable = True
24
25             #RGB to BGR
26             image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

4.1.2 MediaPipe - Elementy charakterystyczne

Biblioteka MediaPipe o otwartym źródle, udostępnia wieloplatformowe oraz konfigurowalne rozwiązania wykorzystujące uczenie maszynowe w dziedzinie rozpoznawania, segmentacji oraz klasyfikacji obiektów wizji komputerowej. Niektórymi z rozwiązań są:

- Rozpoznawanie twarzy
- Segmentacja włosów oraz twarzy
- Rozpoznawanie oraz określanie rozmiarów obiektów trójwymiarowych na podstawie obrazu dwuwymiarowego.

Model modułu MediaPipe pozwala na wyznaczenie pozycji 21 elementów charakterystycznych dłoni. Współrzędne X i Y są znormalizowane względem rozdzielczości obrazu kamery. Współrzędna X względem liczby pikseli w osi X, a współrzędna Y względem liczby pikseli w osi Y. Oś Z jest prostopadła do osi X i Y, z punktem początkowym w punkcie określającym pozycję nadgarstka. Współrzędna Z jest znormalizowana względem szerokości obrazu kamery, tak jak współrzędna X.



Rysunek 4.1: Elementy charakterystyczne dłoni

Pozycje nadgarstka, paliczków oraz stawów dłoni zostaną wykorzystane do

obliczenia obrotu dłoni względem punktu 0 oraz do wytrenowania modeli uczenia maszynowego, których zadaniem będzie rozpoznawanie wybranych gestów.

4.1.3 Generowanie grafiki dłoni

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

4.2 SciKit Learn - uczenie maszynowe

Biblioteka SciKit Learn to biblioteka ... tu opis ...

4.2.1 Budowa programu

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

- Zebranie i przetworzenie danych.
- Przygotowanie algorytmów klasyfikacji i znalezienie najdokładniejszego.
- Zapis modelu do pliku typu **pickle**

4.2.2 Zebranie danych

Przygotowanie danych do przetworzenia będzie wymagało paru operacji matematycznych. Współrzędne opisujące pozycje elementów charakterystycznych są znormalizowane względem wielkości obrazu pobranego z kamery, z czego wynika, że środek układu współrzędnych jest w prawym górnym rogu obrazu pobranego z kamery. W takim wypadku należy przeprowadzić transformację, tutaj akurat przesunięcie układu współrzędnych do pozycji nadgarstka. Taka operacja pozwoli na pozbycie się uniezależnienia zmiennych od pozycji elementów dłoni na obrazie. Ostatecznie pozbywamy się pozycji nadgarstka z wektora danych, ponieważ jest ona środkiem nowego układu współrzędnych.

Macierz przesunięcia

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

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

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

Dane w postaci jednowymiarowej mają postać następującego wektora.

$$A_n = [x_1 \ y_1 \ z_1 \ x_2 \ y_2 \ \cdots \ y_{21} \ z_{21}]$$

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

Normalizacja

$$\frac{[x_1 \ y_1 \ z_1 \ x_2 \ y_2 \ \cdots \ y_{21} \ z_{21}]}{\max() [x_1 \ y_1 \ z_1 \ x_2 \ y_2 \ \cdots \ y_{21} \ z_{21}]}$$

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

4.2.3 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.

W celu wybrania najlepszej metody klasyfikacji, zostanie wybranych kilka algorytmów. Każdy z nich stworzy swój model, a ostatecznie zostanie sprawdzona ich poprawność z wykorzystaniem grupy testowej. Model z najlepszym 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.

4.2.4 Ponowne wykorzystanie modelu

Gotowy model pobieramy i testujemy w przykładowym programie.

4.3 Paczka PyPi

4.3.1 Budowa paczki

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

4.3.2 Załadowanie paczki do repozytorium

Rozdział 5

[Właściwy dla kierunku - np. Specyfikacja zewnętrzna]

Jeśli to Specyfikacja zewnętrzna:

- wymagania sprzętowe i programowe
- sposób instalacji
- sposób aktywacji
- kategorie użytkowników
- sposób obsługi
- administracja systemem
- kwestie bezpieczeństwa
- przykład działania
- scenariusze korzystania z systemu (ilustrowane zrzutami z ekranu lub generowanymi dokumentami)



Politechnika
Śląska

Rysunek 5.1: Podpis rysunku po rysunkiem.

Rozdział 6

[Właściwy dla kierunku - np.Specyfikacja wewnętrzna]

Jeśli to Specyfikacja wewnętrzna:

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

Krótką wstawka kodu w linii tekstu jest możliwa, np. **descriptor**, a nawet **descriptor_gaussian**. Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rysunku 6.1, a naprawdę długie fragmenty – w załączniku.

```
1 class descriptor_gaussian : virtual public descriptor
2 {
3     protected:
4         /** core of the gaussian fuzzy set */
5         double _mean;
6         /** fuzzyfication of the gaussian fuzzy set */
7         double _stddev;
8
9     public:
10        /** @param mean core of the set
11            @param stddev standard deviation */
12        descriptor_gaussian (double mean, double stddev);
13        descriptor_gaussian (const descriptor_gaussian & w);
14        virtual ~descriptor_gaussian();
15        virtual descriptor * clone () const;
16
17        /** The method elaborates membership to the gaussian
18            fuzzy set. */
19        virtual double getMembership (double x) const;
20 };
```

Rysunek 6.1: Klasa **descriptor_gaussian**.

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.

ζ	metoda						
	alg. 1	alg. 2	alg. 3			alg. 4, $\gamma = 2$	
			$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
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 stopień 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.

```
1 partition fcm_possibilistic :: doPartition
2                               (const dataset & ds)
3 {
4     try
5     {
6         if (_nClusters < 1)
7             throw std::string ("unknown_number_of_clusters");
8         if (_nIterations < 1 and _epsilon < 0)
9             throw std::string ("You should set a maximal
10                                number_of_iteration_or_minimal_difference_or_
11                                epsilon.");
12         if (_nIterations > 0 and _epsilon > 0)
13             throw std::string ("Both number_of_iterations_and_
14                                minimal_epsilon_set_or_you should set either_
15                                number_of_iterations_or_minimal_epsilon.");
16
17         auto mX = ds.getMatrix();
18         std::size_t nAttr = ds.getNumberOfAttributes();
19         std::size_t nX    = ds.getNumberOfData();
20         std::vector<std::vector<double>> mV;
21         mU = std::vector<std::vector<double>> (_nClusters);
22         for (auto & u : mU)
```

```
19         u = std::vector<double> (nX);
20     randomise(mU);
21     normaliseByColumns(mU);
22     calculateEtas(_nClusters, nX, ds);
23     if (_nIterations > 0)
24     {
25         for (int iter = 0; iter < _nIterations; iter++)
26         {
27             mV = calculateClusterCentres(mU, mX);
28             mU = modifyPartitionMatrix (mV, mX);
29         }
30     }
31     else if (_epsilon > 0)
32     {
33         double frob;
34         do
35         {
36             mV = calculateClusterCentres(mU, mX);
37             auto mUnew = modifyPartitionMatrix (mV, mX);
38
39             frob = Frobenius_norm_of_difference (mU, mUnew)
40                 ;
41             mU = mUnew;
42         } while (frob > _epsilon);
43     }
44     mV = calculateClusterCentres(mU, mX);
45     std::vector<std::vector<double>> mS =
46         calculateClusterFuzzification(mU, mV, mX);
47
48     partition part;
49     for (int c = 0; c < _nClusters; c++)
50     {
51         cluster cl;
```


Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła \LaTeX owe i końcowa wersja w pdf),
- źródła programu,
- dane testowe.

Spis rysunków

4.1	Elementy charakterystyczne dłoni	9
5.1	Podpis rysunku po rysunkiem.	14
6.1	Klasa descriptor_gaussian	16

Spis tablic

7.1	Opis tabeli nad nią.	18
-----	------------------------------	----