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, listopad 2021

Spis treści

1 Streszczenie **1**

Streszczenie **1**

2 Wstęp **3**

2.1 Wprowawdzenie w problem 4

2.2 Cel pracy 4

2.3 Osadzenie problemu w dziedzienie 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 Zebranie danych 10

4.2.3 Metody klasyfikacji - uczenie maszynowe 11

4.2.4 Ponowne wykorzystanie modelu 11

4.3 Paczka PyPi 11

4.3.1 Budowa paczki 11

4.3.2 Załadowanie paczki do repozytorium 11

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 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

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

Cele projektowe można podzielić na trzy części.

- 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.

Rozdział 4

Wymagania i narzędzia

- wymagania funkcjonalne i niefunkcjonalne
- 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 wyznacznie 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 kontrolerolera odbywa się poprzez główną metodę **main**().

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

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

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

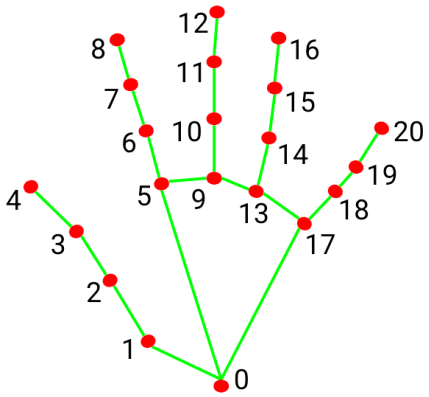
```
    def main(self):
        """
        Main function that runs the core of the program.
        """

        hand_type = None
        if self.cap.isOpened():
            ret, frame = self.cap.read()

            #BGR to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            #Flip horizontal
            image = cv2.flip(image, 1)

            #Set flag
            image.flags.writeable = False

            #Detections
            self.results = self.hands.process(image)

            #Set flag back to True
            image.flags.writeable = True

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

4.1.2 MediaPipe - Elementy charakterystyczne

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

- Rozpoznawanie twarzy
- Segmentacja włosów oraz twarzy
- Rozpoznawnie 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 znomralizowane 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.



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 4.1: Elementy charakterystyczne dłoni

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.

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 bilbioteka ... 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 tworzenie takiego modelu można podzielić na trzy kroki.

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

4.2.2 Zebranie danych

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

Tutaj równanie transformacji

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

Normalizacja

Każdy nowy wektor zostaje zapisany do pliku CSV z odpwowiednią 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, 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.

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. Wygama to przygotowania odpowiednich plików konfiguracyjnych oraz zostosowania stosownych narzędzi do stworznie 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)

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

Krótka 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
            fuzzy set. */
18       virtual double getMembership (double x) const;
19
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.

	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.

```
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
                   number of iteration or minimal difference --
                   epsilon.");
10         if (_nIterations > 0 and _epsilon > 0)
11             throw std::string ("Both number of iterations and
                   minimal epsilon set -- you should set either
                   number of iterations or minimal epsilon.");
12
13         auto mX = ds.getMatrix();
14         std::size_t nAttr = ds.getNumberOfAttributes();
15         std::size_t nX    = ds.getNumberOfData();
16         std::vector<std::vector<double>> mV;
17         mU = std::vector<std::vector<double>> (_nClusters);
18         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              mU = mUnew;
41          } while (frob > _epsilon);
42      }
43      mV = calculateClusterCentres(mU, mX);
44      std::vector<std::vector<double>> mS =
            calculateClusterFuzzification(mU, mV, mX);
45
46      partition part;
47      for (int c = 0; c < _nClusters; c++)
48      {
49          cluster cl;
```

```
50            for (std::size_t a = 0; a < nAttr; a++)
51            {
52               descriptor_gaussian d (mV[c][a], mS[c][a]);
53               cl.addDescriptor(d);
54            }
55            part.addCluster(cl);
56         }
57         return part;
58      }
59      catch (my_exception & ex)
60      {
61         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex.what());
62      }
63      catch (std::exception & ex)
64      {
65         throw my_exceptionn (__FILE__, __FUNCTION__, __LINE__
               , ex.what());
66      }
67      catch (std::string & ex)
68      {
69         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex);
70      }
71      catch (...)
72      {
73         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               "unknown expection");
74      }
75   }
```

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

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