Gliwice, 12.02.2021

Sprawozdanie z projektu PBL

Elastyczne systemy produkcyjne w przemyśle 4.0 – stanowisko laboratoryjne wspomagające projektowanie systemów logistyki wewnętrznej bazujących na pojazdach AGV (Autonomous Ground Vehicle)

Zespół projektowy:

Mateusz Szwedka – Informatyka SII/lider zespołu

Marek Daniszewski – Informatyka SII

Karol Knapik– Informatyka SII

Wojciech Stańczuk – Informatyka SII/Koło naukowe Industrum

Piotr Zielonka – Informatyka SII

Opiekunowie:

dr hab. inż. Rafał Cupek – RAu8/główny opiekun naukowy

dr hab. inż. Adam Ziębiński – Rau8/pomocniczy opiekun naukowy

mgr inż. Tomasz Stęclik – Wspólna Szkoła Doktorska/pomocniczy opiekun naukowy

Projekt zrealizowany dzięki dofinansowaniu przyznanemu w ramach I konkursu finansowania projektów studenckich kół naukowych w ramach programu „Inicjatywa Doskonałości – Uczelnia Badawcza”

Spis treści

1. Cel i zakres projektu

Celem projektu jest stworzenie stanowiska laboratoryjnego, którego poszczególne moduły będą realizować następujące funkcje:

a/ Warstwa sprzętowa i oprogramowanie systemu wbudowanego

– celem podzadania było pozyskanie wybranych danych z laboratoryjnego modelu AGV. Dane są zbierane przez serwer OPC UA zainstalowany na kontrolerze Raspberry Pi;

- format danych odpowiada formatowi stosowanemu w podzadaniach b i c tak aby dane można było wykorzystać w modułach eksploracji i wizualizacji;

- dane obejmują zużycie energii, pracę napędów oraz dodatkowe informacje związane z pracą AGV;

- moduł umożliwia sterowanie pracą AGV;

- moduł akwizycji danych komunikuje się ze środowiskiem ROS dla Raspberry Pi:

b/ Moduł komunikacji, symulacji i wizualizacji

- komunikacja z AGV i modułami analitycznymi realizowana jest poprzez interfejs OPC UA z zastosowaniem narzędzi wspomagającymi przegląd modelu danych dostępnego na serwerze OPC UA pojazdu AGV oraz tworzenie wzorców wizualizacji dla środowiska Wonderware

- moduł symulacji umożliwia generację tras pojazdu AGV i w połączeniu z warstwą sprzętową i oprogramowaniem wbudowanym (a) pozwala na przygotowanie danych dla modułu (c) bez konieczności przeprowadzenia fizycznych testów na platformie AGV

- podzadanie wizualizacji polega na wizualizacji logistyki wewnętrznej opartej na AGV dla środowiska Wonderware Archestra, oraz na implementacji wybranych elementów tego systemu;

- wizualizacja AGV w Intouch obejmie prezentację podstawowych informacji o AGV na layoucie (położenie, realizowane zadania, statusy itp.) oraz informacje szczegółowe prezentowane w postaci stacyjek sterowania dla wybranego AGV. Stacyjki pozwalają nie tylko na wizualizację danych szczegółowych, ale także na manualne sterowanie pracą AGV z wykorzystaniem interfejsu OPC UA;

- layout trasy AGV jest importowany z plików xml i rysowany w środowisku Intouch, procedura importu została zautomatyzowana tak aby można łatwo tworzyć okna wizualizacji dla różnych wariantów systemów logistyki wewnętrznej;

c/Moduł klasteryzacji i predykcji

- Smart Battery Charger – blok realizujący analizę zużycia energii celem znalezienia odpowiedzi na pytania dotyczące planowania cykli pracy AGV, realizacji zleceń logistycznych i ładowania baterii AGV;

- AGV Safety Module – blok wspomagający analizę związaną z bezpieczeństwem systemów opartych o AGV: analiza zatrzymań ze względu na pojawienie się przeszkód na trasie, analiza czy dana trasa, staje się coraz bardziej bezpieczna czy wręcz odwrotnie, analiza usług logistycznych floty AGV pod kątem optymalizacji przebiegu tras logistycznych tak ze względu na bezpieczeństwo jak i efektywność energetyczną;

1.1 Stan realizacji projektu na koniec semestru zimowego r.a. 20/21 i zadania zaplanowane do realizacji w semestrze letnim r.a. 20/21

a/ Warstwa sprzętowa i oprogramowanie systemu wbudowanego

W semestrze zimowym zrealizowano warstwę sprzętową dla układu pomiaru zużycia energii i układu śledzenia stanu naładowania baterii. Zrealizowano także część sprzętową symulacji zużycia energii przez pojazd AGV pozwalającą na weryfikację działania układu pomiarowego i oprogramowania wbudowanego. Szczegółowy zakres prac wykonanych w semestrze zimowym został opisany w rozdziale 2.

Na semestr letni zaplanowana jest integracja układu pomiarowego z pojazdem AGV. W zakresie symulacji planowane jest przeprowadzenie pełnych testów symulatora oraz porównanie wyników uzyskanych w rozwiązaniu symulacyjnym z rzeczywistymi pomiarami, które zostaną przeprowadzone dla pojazdu AGV. Celem planowanych prac jest ocena dokładności zaproponowanego algorytmu symulacji zużycia energii przez pojazd AGV.

b/ Moduł komunikacji, symulacji i wizualizacji

W semestrze zimowym zrealizowano uproszczony serwer OPC UA dla pojazdu AGV, podstawowy algorytm symulacji i makietę systemu wizualizacji. Przeprowadzono testy serwera OPC w oparciu o oprogramowanie symulacyjne i połączono serwer OPC UA z modułem wizualizacji. Szczegółowy zakres prac wykonanych w semestrze zimowym został opisany w rozdziale 3.

Na semestr letni zaplanowana jest rozbudowa serwera OPC tak aby uwzględnił on pełną listę sygnałów, które wynikają z rozwiązań zastosowanych w części (a). Planowana jest także rozbudowa modułu o funkcje dynamicznego ładowania scenariusza symulacji. W zakresie wizualizacji do realizacji pozostaje powiązanie ekranów prezentowanych w środowisku Archestra z plikami .xml zawierającymi opisy tras AGV.

c/Moduł klasteryzacji i predykcji

W semestrze zimowym przygotowano oprogramowanie testowe wykorzystujące algorytmy klasteryzacji K-means i DBSCAN, algorytm regresji liniowej k-NN. Ze względu na brak rzeczywistych danych z pojazdów AGV, działanie modułu przetestowano na dwóch zbiorach danych testowych typu brenchmark: Iris i Wines. Szczegółowy wykonanego oprogramowania oraz wyniki testów przeprowadzonych na zbiorach testowych opisano w rozdziale 4.

Na semestr letni planowane są testy i strojenie algorytmów eksploracji danych z wykorzystaniem pomiarów uzyskanych przez moduł sprzętowy zrealizowany w części (a). Dane zostaną pobrane za pomocą serwera OPC UA zrealizowanego w części (b). W trakcie symulacji wygenerowane zostaną dane testowe dla znanych wariantów przejazdu pojazdu AGV. Dodatkowo wykorzystane zostaną dane zebrane w czasie rzeczywistych przejazdów AGV. W trakcie badan wyznaczone zostaną miary czułości i dokładności klasyfikacji oraz parametr F1 zarówno dla uzyskanych na drodze symulacji jak i dla danych rzeczywistych.

2. Warstwa sprzętowa i oprogramowanie systemu wbudowanego

W ramach projektu należało zaprojektować oraz utworzyć moduł pozwalający określić stan naładowania akumulatora pojazdu AGV. W ramach pierwszej części projektu określono metodę i ogólną specyfikę sprzętu potrzebnego do wykonania zadania. Zdecydowano się na metodę zliczania ładunku tzw. „Coulomb counting method". Następnie należało dokonać niezbędnych zakupów i zbudować moduł pomiarowy. Niestety przez charakter zdalny studiów wywołany pandemią nie możliwe były testy moduł w rzeczywistym robocie, przez co zdecydowano się na utworzenie stanowiska sztucznego obciążenia akumulatora, które miało współpracować z symulatorem pojazdu, wizualizacją oraz modułem analizy danych.

Założenia przy budowaniu stanowiska sztucznego obciążenia:

- możliwość pełnego obciążenia akumulatora
- wykorzystanie jak największego zakresu pomiarowego czujników prądowych
- duża rozdzielczość obciążenia
- łatwa sterowalność poborem prądu
- dostępność elementów składowych

2.1 Opis systemu

Moduł SOC opiera się o metodę zliczania ładunku który wpłyną i został pobrany z akumulatora. Ładunek zgodnie ze wzorem:

$$Q = It$$

ładunek można określić iloczynem natężenia prądu *I* wypływającego/wpływającego do akumulator w czasie *t*.

Komputer z kartą pomiarową ma za zadnie całkować prąd wpływający i wypływający. Dane z komputer o stanie naładowania baterii są dostępne za pośrednictwa serwera OPC UA.
Układ sztucznego obciążenia generujący informacje dla modułu wyznaczającego SOC, składa się z:

1) Akumulatora – z wbudowanym układem BMS, który zabezpiecza układ przed zbyt dużymi różnicami pomiędzy napięciami na poszczególnych ogniwach. Pozwala on także balansować cele podczas ładownia i rozładowywania akumulatora.
2) Układu ładowania - przetwornica o mocy pozwalającej ładować akumulatora oraz zasilać układ obiciążający – symulacja ładowarki pojazdu AGV.
3) Układu sterowania – komputer, który na podstawie informacji steruje układem obciążenia, wywołując określone obciążenie.
4) Układu obciążenia – układ elementów biernych pozwalających na rozproszenie mocy. W przypadku tego projektu posłużono się paskami z adresowalnymi diodami LED.

Każdy z układów ma inną specyfikę elektryczna. Akumulator generuje napięcie od około 12 do 17 V w zależności od poziomu naładowania. Układ sterujący wymaga stabilnego napięcia 5V. Układ obciążenia w przypadku diod LED potrzebuje napięcia 5V, jednak w odróżnieniu od układu sterowania moc będzie znacznie większa. Ładowanie musi być w stanie pokryć zapotrzebowanie mocy wszystkich układów.

W celu zapewnienia optymalnych warunków bezpiecznej pracy oraz mając na uwadze straty w układach, należy użyć elementów o większej mocy niż jest w stanie maksymalnie pobrać dany zespół elementów. Nie można także zbytnio przewymiarować mocy elementów, gdyż może mieć to negatywne skutki pod względem wydajności.

2.2 Określenie parametrów podzespołów

Przy założeniu wykorzystania jak największego zakresu pomiarowego wcześniej kupionych czujników prądowych należało wybrać akumulator pozwalający na pobranie ponad 10 A. Przy założeniu, że akumulator powinien pracować w standardowym zakresie wydajności prądowej, dobrano akumulator Litowo-jonowy o pojemności 10,4Ah. Wyszukano akumulator w konfiguracji 4S2P (4 cele po 2 ogniwa). Napięcie nominalne to 14,4 V, a wiec posiadamy 149,76 Wh energii.

Zakładając, że maksymalna moc układu może wynosić około 150W należy dobrać obciążenie, które wraz z układem sterowania nie przekroczy tej wartości. Maksymalna moc jednego metra paska LED z 144 diodami to 43,2 W. Dla 2 metrów to 86,2 W. Dla komputera raspberrypi 4B założono maksymalną moc 20W. Do komputera podłączona będzie karta ADC oraz sensory i sygnał sterujący diodami, co nie powoduje dużego zużycia. Zakładając, że ładowanie odbywać się będzie z napięciem 16.8 V i prądem 10,4 A, moc potrzebna do ładowania to 174,72 W. W celu zapewnienia działania układu podczas ładowania należy dodać 86,2 W potrzebne na działanie diod oraz 20W dla komputera. Łączna moc ładowarki nie powinna być mniejsza niż 280,92 W.

2.3 Komponenty warstwy sprzętowej i ich specyfikacja

Akumulator pakiet 14,4V 10,4Ah LI-ION

Po rozpatrzeniu kilku opcji zdecydowano się na gotowy akumulator z wbudowanym układem BMS ze względu na cenę oraz łatwość implementacji. Wybrano akumulator firmy Amelectronics, ponieważ nie znaleziono dostępnego, konkurencyjnego produktu w tym czasie.

Rysunek 1 Źródło: https://amelectronics.pl/produkt/akumulator-pakiet-144v-104ah-li-ion-bms-10-4s2p/

- Typ ogniwa: LI-ION

- Napięcie: 14,4V
- Pojemność: 10 400mAh
- Ilość ogniw: 8
- Pakiet z wyprowadzonymi przewodami, do samodzielnego montażu
- BMS 10A 4S2P

Zasilacz ładowarki

Rysunek 2 Źródło https://botland.com.pl/zasilacze-montazowe/18507-zasilacz-montazowy-do-tasm-i-paskow-led-12v-29a-350w-5900804092056.html

Specyfikacja techniczna zasilacza impulsowego

- Model: CP12350B20
- Typ zasilacza: impulsowy, montażowy
- Maksymalna moc: 350 W
- Napięcie wyjściowe: 12 V DC
- Prąd wyjściowy: 29 A
- Napięcie zasilania: od 175 V do 240 V AC / 50 / 60 Hz
- Podłączenie elektryczne: listwa zaciskowa
- Zabezpieczenia:
    - Przed przeciążeniem
    - Przed wzrostem napięcia
    - Przed zwarciem
- Montaż: do wbudowania
- Liczba wejść: 3
- Temperatura pracy: od -25°C do 70°C

- Wymiary: 215 x 110 x 50 mm

Przetwornica ładowarki

Rysunek 3 Źródło https://abc-rc.pl/product-pol-12108-Przetwornica-DC-DC-1200W-20A-STEP-UP-8-60V-12-80V.html

Parametry

- Napięcie wejściowe: 8V - 60V
- Napięcie wyjściowe (regulowane): 12V - 80V
- Prąd wejściowy: max 20A
- Prąd wyjściowy: max 20A
- Moc wyjściowa: 1200W
- Bezpiecznik wejściowy: 30A
- Sprawność do 95%
- Temperatura pracy: -45C - +85
- Prąd spoczynkowy: 15mA
- Przetwornica: DC-DC
- Wymiary (dł. x szer. x wys.): 133mm x 52mm x 43mm

Niestety ograniczeniem tego modułu jest maksymalny prąd wejściowy. Nie znaleziono przetwornicy o lepszych parametrach przez co maksymalna moc ładowania to około 240 W.

Przetwornica układu obciążenia

Specyfikacja

- sprawność do 95%
- chłodzenie pasywne
- Napięcie wyjściowe: 5V
- Maksymalny prąd wyjściowy: 30A
- Dokładność napięcia: ±1.5%

- IP68
- Wymiary: 74*74*30mm
- waga: 0.28kg
- zakres temperatur pracy: -40~85°C
- certyfikaty: CE, ROHS

Rysunek 4 Źródło: https://www.daygreen.com/collections/12-24v-to-5v/products/12v-24v-to-5v-30a-150w-dc-dc-step-down-converter-voltage-regulator

Adresowalne diody LED

Podłączenie

Urządzenie posiada 4-pinowe złącze składające się z następujących wyprowadzeń:

- GND (przewód biały) - masa modułu
- V (przewód czerwony) - napięcie zasilania, dla jednej matrycy należy podłączyć źródło o wydajności prądowej co najmniej 3,5 A na każdy metr łańcucha
- D0 (przewód zielony - data) - cyfrowy sygnał sterujący z mikrokontrolera
- B0 (przewód niebieski - clock) - cyfrowy sygnał sterujący z mikrokontrolera - podłączony w zależności od potrzeby programu

Specyfikacja

- Napięcie zasilania: 5 V
- Moc: 43,2 W/m
- Zastosowane diody: LED RGB SMD5050 WS2813
- Adresowany piksel: 1 dioda RGB
- Odporność na warunki zewnętrzne: IP30
- Długość łańcucha: 1 m

Rysunek 5 Źródło https://botland.com.pl/paski-led-adresowane/8737-pasek-led-rgb-ws2813-cyfrowy-adresowany-144-ledm-432wm-5v-1m-ip30.html

Przetwornica układu sterowania

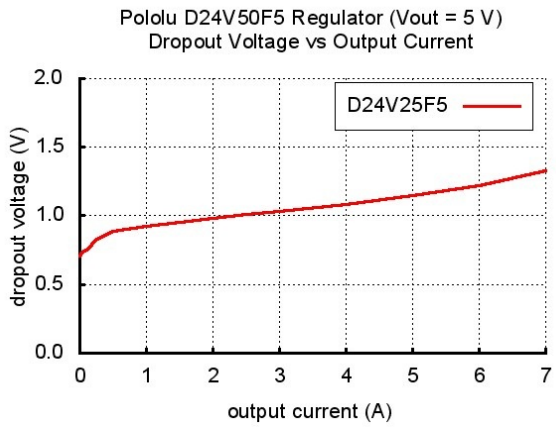
Rysunek 6 Źródło: https://botland.com.pl/przetwornice-step-down/2754-d24v50f5-przetwornica-step-down-5v-5a-pololu-2851.html

Specyfikacja przetwornicy step-down Pololu D24V50F5

- Napięcie wejściowe: 6* V do 38 V
- Napięcie wyjściowe: 5 V
- Dokładność napięcia wyjściowego: 4 %
- Prąd wejściowy maksymalny: 5 A
- Sprawność na poziomie 85 - 95 %
- Zintegrowane zabezpieczenie przed zbyt wysokim prądem, napięciem i temperaturą
- Dwa otwory montażowe o średnicy 2,2 mm

- Wymiary: 20 x 18 x 9 mm
- Masa (bez złącz): 3,0 g

Na poniższym wykresie widać spadek napięcia w funkcji prądu – wymagany będzie pomiar napięcia zasilania w celu kompensacji błędu. Dla komputerów oraz sensorów jest to niekorzystna sytuacja z powodu zmiennej skali pomiarowej. Komputer RaspberryPi jest czuły na spadki napięcia przez co jego wydajność może spaść. W razie problem z wydajnością możliwe jest zastosowanie zasilacza 5 V.

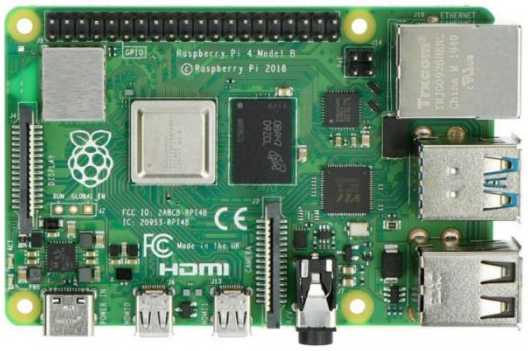


Rysunek 7 Źródło: https://botland.com.pl/przetwornice-step-down/2754-d24v50f5-przetwornica-step-down-5v-5a-pololu-2851.html

Raspberry Pi 4 B (komputer)

Specyfikacja:

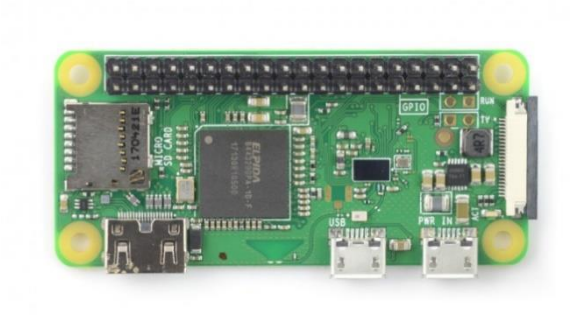
- 8 GB RAM
- Procesor BCM2711 z 64-bitowym rdzeniem Quad-core ARM-8 Cortex-A72 CPU, taktowany 1,5 GHz
- 2 złącza monitora microHDMI do 4K (4K i 60 kl/s dla jednego monitora oraz 4K i 30 kl/s dla dwóch monitorów).
- 4 złącza USB. Dwa w wersji 2.0 oraz dwa USB 3.0.
- Prędkość Ethernet do 1000 Mbit/s
- Złącze zasilania USB C
- Złącze zasilania zostało dostosowane do najnowszych standardów. Raspberry posiada złącze USB C. Producent zaleca zasilanie 5 V / 3 A.
- Bluetooth 5.0
- Interfejs WiFi Dual Band 2,4 GHz i 5 GHz 802.11 b/g/n/ac
- Komunikacja UART, SPI, I2C, GPIO



Rysunek 8 Źródło

https://botland.com.pl/moduly-i-zestawy-raspberry-pi-4b/16579-raspberry-pi-4-model-b-wifi-dualband-bluetooth-8gb-ram-15ghz-765756931199.html

W pierwszej fazie testów zakładano wykorzystanie także Raspberry Pi Zero W



Rysunek 9 Źródło: https://botland.com.pl/moduly-i-zestawy-raspberry-pi-zero/9749-raspberry-pi-zero-wh-512mb-ram-wifi-bt-41-ze-zlaczami.html

Domyślnie słabszy komputer Pi Zero miał być odpowiedzialny wyłącznie za sterowanie adresowalnymi diodami LED. Jednak testy funkcjonalności biblioteki oraz wydajnościowe oraz wykazały, że lepszym rozwiązaniem będzie wyłącznie Raspberry pi 4 B.

Sensory prądowe

Czujniki działają na podstawie efektu Halla. Możliwa jest dodatkowa filtracja ponieważ układy scalone posiadaj wewnętrzny filtr o rezystancyjny 1,7k kΩ, na płytce znajduje się kondensator o wartości 1nF, co razem tworzy filtr dolnoprzepustowy o częstotliwości granicznej 90 kHz.

Użytkownik może modyfikować filtr montując kondensator w wyprowadzenia o nazwie "filter". Częstotliwość graniczną można obliczyć ze wzoru:

$$F = 1 / (2\pi RC) = 1 / (11k\Omega * (1\,nF + Cf))$$

gdzie Cf jest pojemnością kondensatora wlutowanego w wyprowadzenia "filter".

Sensor ACHS-7121 +/- 10 A

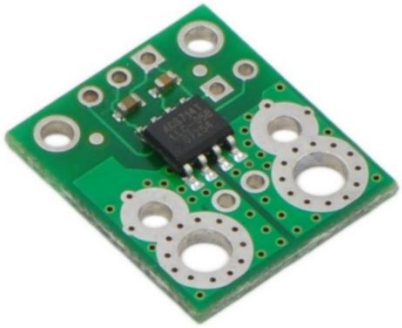
Rysunek 10 Źródło: https://botland.com.pl/czujniki-pradu/14236-czujnik-pradu-achs-7121-10a-do-10a-pololu-4030.html

Specyfikacja:

- Napięcie zasilania: od 4,5 V do 5,5 V
- Przystosowany do stosowania w systemach 5 V
- Czułość zasilania: 0,185 V/A
- Prąd zasilania 15 mA
- Wymiary: 17,8 x 20,3 mm

Sensor ACS714 +/- 5A

- Specyfikacja:
- Napięcia zasilania części logicznej: od 4,5 V do 5,5 V
- Pobór prądu części logicznej: 13 mA
- Czułość dla zasilania 5V: 0,185 V/A
- Masa modułu 1,3 g
- Wymiary: 18 x 20,5 mm



Rysunek 11 Źróło: https://botland.com.pl/czujniki-pradu/523-czujnik-pradu-acs714-5a-do-5a-pololu-1185.html

Dzielnik napięcia

Rysunek 12 Źródło: https://botland.com.pl/seria-gravity/11261-dfrobot-gravity-dzielnik-napiecia-15.html

Specyfikacja:

- Napięcie zasilania: od 3,3 V do 5 V
- Zakres pomiarowy: od 0,0245 V do 25 V
- Napięcie wyjściowe: proporcjonalne do wejściowego w zakresie od 0 V do 5 V
- Rezystor R1: 7,5 kΩ
- Rezystor R2: 30 kΩ
- Stosunek Uwy/Uwe: 5 do 1

- Wymiary: 22 x 30 mm

Karta pomiarowa

Rysunek 13 Źródło: *https://botland.com.pl/rozszerzenia-gpio-do-raspberry-pi/4478-ads1256-dac8552-przetwornik-ac-i-ca-2416-bit-spi-nakladka-dla-raspberry-pi-waveshare-11010.html*

Specyfikacja:

- Podłączenie przez piny GPIO Raspberry Pi
- Wbudowane przetworniki:
    - ADS1256 - 8-kanałowy, 24-bitowy przetwornik A/C (4 wejścia różnicowe)
    - DAC8552 - 2-kanałowy. 16-bitowy przetwornik C/A
- Moduł posiada złącza goldpin oraz śrubowe

Na cel projektu wykorzystano wyłącznie przetwornik ASD1256

Biblioteka oraz dokumentacja dostępna na stronie producenta:
https://www.waveshare.com/wiki/High-Precision_AD/DA_Board

2.4 Opis modułu wyznaczającego SOC

Algortym

Do opracowania algorytmu posłużono się opracowaniem dostępnym pod linkiem: https://www.analog.com/media/en/technical-documentation/technical-articles/a-closer-look-at-state-of-charge-and-state-health-estimation-techniques.pdf

Wybrana metoda Coulomb Counting Method opiera się o całkowanie prądu.

$$SOC = SOC(t_0) + \int_{t_0}^{t_0+\tau} (I_b - I_{loss})dt$$

I_b - aktualny prąd wychodzący z akumulatora

I_{loss} – wartość prądu reakcji strat (w naszym przypadku zakładamy zero)

Za zliczanie ładunku odpowiada funkcja: BatteryMonitor::update

```
void BatteryMonitor::update(double voltage, double current)
{
 if (!initialized_)
 {
  initialized_ = true;
  cout << "setting up init in feed function\n";
  lastUpdate_ = std::chrono::steady_clock::now();

  return;
 }
 updateMeanCurrent(current);
 std::chrono::steady_clock::time_point time_now = std::chrono::steady_clock::now();
 double dt = std::chrono::duration<double>(time_now - lastUpdate_).count() / 3600.0; //hours
 battery_.parameters.time_stamp = time_now;
 double C = dt * current;
 lastUpdate_ = time_now;

 if (current > 0.0)
 { /////////////// CHARGING ///////////////
  // battery_.parameters.power_supply_status =
batteryMsg_.POWER_SUPPLY_STATUS_CHARGING;
  battery_.parameters.c_owned += C;

  if ((current < nominalLoadCurrent_) && !calibrate)
  {
   if (underCurrnetCounter >= 100)
   {
    calibrate = true;
```

```
      cout << "calibration set\n";
    }
    underCurrnetCounter++;
  }
  else
  {
    underCurrnetCounter = 0;
  }
}
else if (current <= 0.0)
{ /////////////// DISCHARGING ///////////////
  if (calibrate)
  {
    cout << "calibrating!!!\n";

    if (battery_.parameters.c_rated > battery_.parameters.c_owned)
    {


      cout << "DEGRADACJA OGNIWA! "
           << "diff :" << battery_.parameters.c_rated - battery_.parameters.c_owned << "\n";
    }

    battery_.parameters.c_rated = battery_.parameters.c_owned;
    calibrate = false;
  }

  if (battery_.parameters.c_owned > 0.0)
  {
    battery_.parameters.c_owned += C; // odejmowanie c jest ujemne (prąd jest ujemny)
  }
  else
  {
    battery_.parameters.c_owned = 0.0;
  }

  if (battery_.parameters.c_owned <= 0.0)
  {
    battery_.parameters.c_rated += abs(C);
  }
}
```

```
 if (battery_.parameters.c_owned > (battery_.parameters.c_rated * 1.5))
 {
  battery_.parameters.c_rated = battery_.parameters.c_owned;
  cout << "calibrating!!!\n";

  calibrate = false;
 }
 // obliczanie parametrów
 if (battery_.parameters.c_rated != 0.0)
 {
  battery_.parameters.SOC = battery_.parameters.c_owned / battery_.parameters.c_rated;
 }

 if (battery_.parameters.c_nominal != 0.0)
 {
  battery_.parameters.SOH = battery_.parameters.c_rated / battery_.parameters.c_nominal;
 }
 if ((std::chrono::duration<double>(time_now - lastDump_).count()) > 5.0)
 {
  lastDump_ = std::chrono::steady_clock::now();
  battery_.save();
  cout << "saving to bin file\n";
 }
}
```

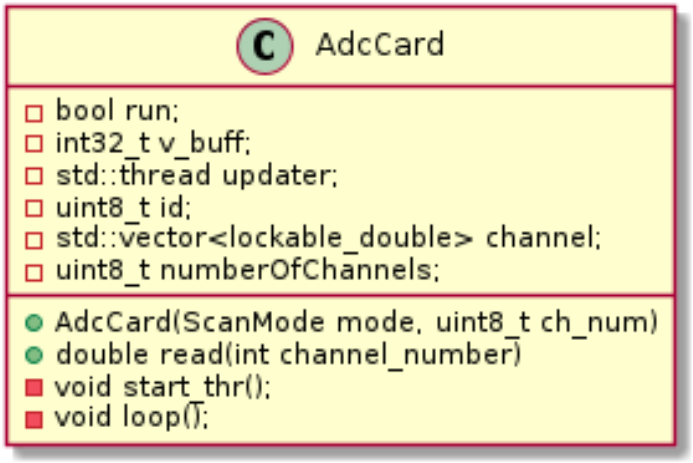
Zapis danych

Jeśli w systemie zarejestrowano ładowanie baterii, zwiększana jest zmienna battery_.parameters.c_owned i od niej w pierwszej kolejności odejmowany jest ładunek wychodzący z akumualtora. Jeśli zmienna battery_.parameters.c_owned jest równa 0, i z akumulatora nadal pobierany jest ładunek oznacza to, że pobierany jest ładunek, którego nie zarejestrowano, należy wiec zwiększyć pojemność akumulatora, ponieważ ładunek ten znajdował się tam zanim system został włączony. Dlatego dla najlepszej jakości działania zalecane jest jak najczęstsze rozładowanie i ładowanie baterii do maksimum. Nie określono początkowego stanu naładowania pobieranego z napięcia. Funkcjonalność ta wymaga testów z akumulatorem.

Z uwagi na możliwość nagłego odcięcia zasilania należało przewidzieć system bezpiecznego zapisywania informacji, który będzie odporny na błąd przy zapisie w chwili nagłego wyłączenia. Problem rozwiązano przez cykliczne zapisywanie plików z parametrami w postaci binarnej z timestamp'em. W chwili włączenia system wyszukuje pliki z parametrami, następnie określa, który z nich posiada „najstarsze" dane. Jeśli plik byłby uszkodzony, timestamp, który jest ostatnim rekordem nie będzie poprawny i plik nie będzie brany pod uwagę.

Pomiar prądu i napięć

Biblioteka przygotowana przez producenta karty pomiarowej, implementuje komunikację oraz odczyt danych. Wykorzystując tą bibliotekę przygotowane klasę umożliwiającą asynchroniczne pomiary i odczyt danych.



Rysunek 14 Klasa AdcCard

Klasa wymaga podania trybu:

- single_channel – pomiar względem napięcia referencyjnego
- differentia – pomiar napięcia pomiędzy 2 sąsiadującym wejściami

oraz ilość kanałów jakie chcemy (założono, że kolejne sygnały będą podłączane w kolejności numerów wejść) Obiekt po stworzeniu jest gotowy do pobrania danych za pomocą funkcje read, jako argument przekazujemy nr kanału, który chcemy odczytać.

Sterownik adresowalnych diod LED

Do sterowania poszczególnymi diodami wykorzystano język python3 oraz bibliotekę rpi_ws281x. W celach testowych napisano program wykorzystujący ROS. Skrypt tworzy 3 tematy odpowiedzialne za 3 kolory (czerwony, zielony,niebieski). Publikując liczbę z przedziału 0 do 288 (ilość diod na paskach), program załączy tyle diod o danym kolorze.

```
#!/usr/bin/env python3

import rospy
import time
from std_msgs.msg import UInt16
from rpi_ws281x import *



class RgbStrip:

    def __init__(self, pixelsNum, ledPin,  brightest, dma=10, led_channel=1):
```

```
    self.led_count = pixelsNum
    self.led_pin = ledPin
    self.dma_channel = dma
    self.led_brightness = brightest
    self.strip = Adafruit_NeoPixel(
        self.led_count, self.led_pin, 800000, self.dma_channel, False, self.led_brightness, led_channel)

    self.strip.begin()
    # print(self.strip.)
    rospy.Subscriber('r', UInt16, self.RedCallback)
    rospy.Subscriber('g', UInt16, self.GreenCallback)
    rospy.Subscriber('b', UInt16, self.BlueCallback)

def __clamp(self, value, min_value, max_vale):
    return max(min_value, min(value, max_vale))

def __del__(self):
    self.colorWipe(Color(0, 0, 0))

def colorWipe(self, color, wait_ms=1):
    for i in range(self.strip.numPixels()):
        self.strip.setPixelColor(i, color)
        self.strip.show()
        time.sleep(wait_ms/1000.0)

def getColorArray(self, i):
    color = self.strip.getPixelColor(i)
    r = (color >> 16 & 0xff)
    g = (color >> 8 & 0xff)
    b = (color & 0xff)
    return [r, g, b]

def RedCallback(self, data):
    red_leds = self.__clamp(data.data, 0, self.led_count)
    for i in range(red_leds):
        color = self.getColorArray(i)
        color[0] = 255
        self.strip.setPixelColorRGB(i, color[0], color[1], color[2])
    for i in range(red_leds, self.led_count):
        color = self.getColorArray(i)
        color[0] = 0
        self.strip.setPixelColorRGB(i, color[0], color[1], color[2])
```

```
        self.strip.show()

    def GreenCallback(self, data):
        red_leds = self.__clamp(data.data, 0, self.led_count)
        for i in range(red_leds):
            color = self.getColorArray(i)
            color[1] = 255
            self.strip.setPixelColorRGB(i, color[0], color[1], color[2])
        for i in range(red_leds, self.led_count):
            color = self.getColorArray(i)
            color[1] = 0
            self.strip.setPixelColorRGB(i, color[0], color[1], color[2])

        self.strip.show()

    def BlueCallback(self, data):
        red_leds = self.__clamp(data.data, 0, self.led_count)
        for i in range(red_leds):
            color = self.getColorArray(i)
            color[2] = 255
            self.strip.setPixelColorRGB(i, color[0], color[1], color[2])
        for i in range(red_leds, self.led_count):
            color = self.getColorArray(i)
            color[2] = 0
            self.strip.setPixelColorRGB(i, color[0], color[1], color[2])

        self.strip.show()

if __name__ == "__main__":

    rospy.init_node('led_controller', anonymous=True)

    leds = RgbStrip(pixelsNum=288, ledPin=13, brightest=255)
    rospy.spin()
```

Poniżej zdjęcia w czasie działania kodu testowego z wykorzystaniem Raspberrry Pi Zero

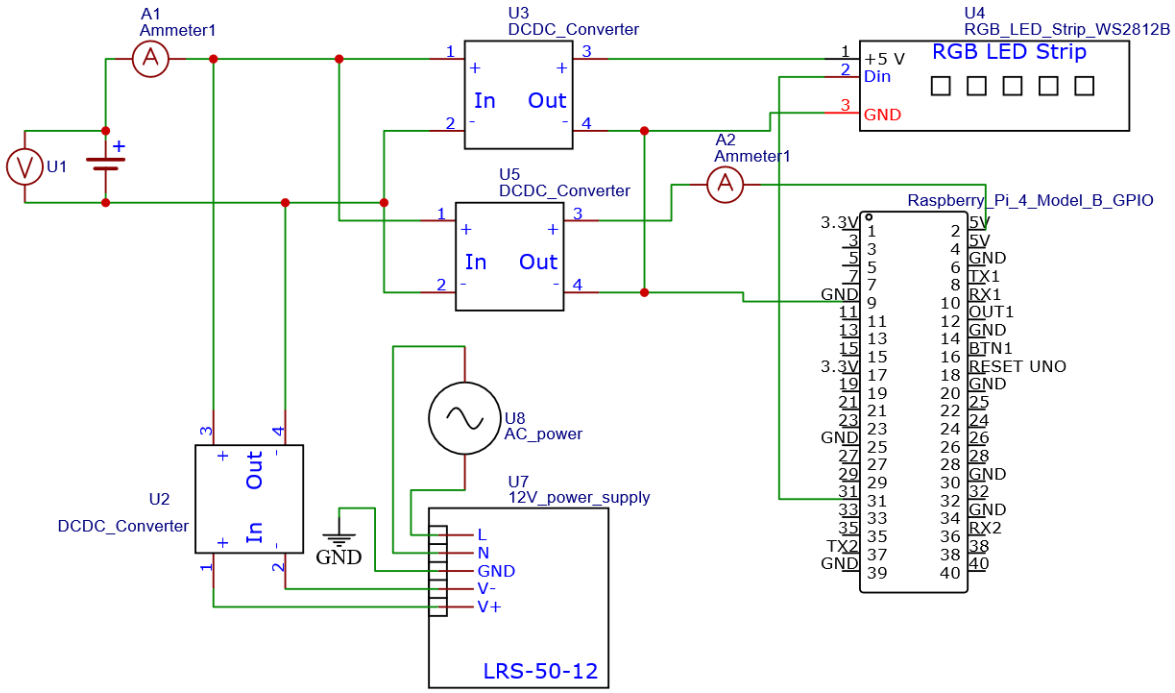
Rysunek 15 Działanie diod adresowalnych

Podczas testów zauważono miganie diod (nagła zmiana koloru, następnie powrót do normalnej pracy). Nie zdiagnozowano ostatecznej przyczyn tego błędu, jednak nie wpływa ona na funkcjonalność układu. Najprawomocniej przyczyna mogą być :

- drgania styków lub
- błędy biblioteki, która nie poprawnie obsługuje szybkie zmiany dużej ilości diod
- niestabilne zasilanie
- wadliwe złącza

2.5 Schemat układu

Na schemacie w celu zachowania czytelności nie zaznaczono wyprowadzeń sygnałowych z czujników do karty pomiarowej. Obecność tych połączeń na nie jest kluczowa dla czytelności oraz nie została jeszcze jednoznacznie określona.



- A1 - Sensor ACHS-7121 +/- 10 A
- A2 - Sensor ACS-714 +/- 5 A
- U1 dzielnik napięcia
- U2 – przetwornica ładowarki
- U3 – przetwornica układu obciążenia
- U5 – przetwornica układu sterowania

3. Komunikacja, symulacja i wizualizacja

Symulator AGV jest elementem projektu, którego celem jest zastąpienie rzeczywistego pojazdu wirtualnym modelem, a co za tym idzie umożliwienie prowadzenia badań zdalnie. Głównym założeniem tej części jest generacja sygnałów i zbiorów danych, umożliwiających wizualizację stanu jednostki AGV wraz z jej położeniem i orientacją na płaszczyźnie (x,y) oraz symulację zużycia energii podczas pracy, na podstawie informacji o aktywnych systemach (napęd, oświetlenie itp.).

Część symulatora opisana w tej części skupia się wokół softwareowej strony modelu, obejmując zagadnienia komunikacji OPC, programowanej logiki oraz sygnałów, składających się na wszystkie informacje na temat stanu wirtualnego AGV.

Sama zasada działania symulatora bazuje na pętli pracy sterownika PLC:

Odczyt wejść -> Wykonanie -> Wystawienie sygnałów wyjściowych

Sygnały wyjściowe wystawiane są na serwer OPC, co umożliwia ich odczyt i obróbkę przez pozostałą część zespołu.

Sterowanie symulacją odbywa się poprzez obsługę parametrów symulacyjnych na serwerze OPC (nadpisywanie wartości również z poziomu klienta): wybór numeru polecenia przejazdu, szybkość symulacji, zatrzymanie przejazdu.

3.1 Wykorzystane narzędzia

Do zbudowania symulacji pojazdu AGV zastosowano zintegrowane środowisko programistyczne PyCharm dla języka programowania Pyhon firmy JetBrains.

Wykorzystane moduły w obrębie projektu Python:

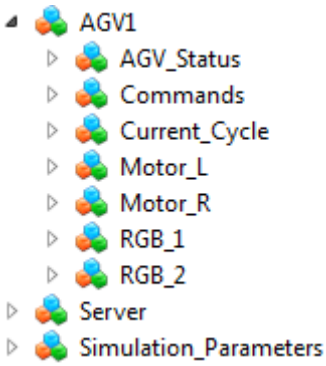
- time - wykorzystanie RTC (zegara systemowego) do koordynacji operacji
- numpy - operacje na macierzach
- matplotlib - kreślenie wykresów
- opcua - komunikacja OPC

Z powyższych modułów jedynym modułem niestandardowym jest ogólnodostępna biblioteka opcua, pozwalająca na obsługę komunikacj.

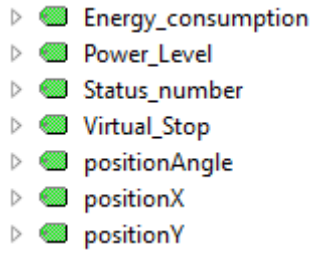
3.2 Lista sygnałów pojazdu AGV

Sygnały są całością informacji o wirtualnym pojeździe. Determinują one całą strukturę, na której opiera się model. To na ich podstawie tworzona jest wizualizacja, a niesione dane wykorzystywane są dalej w procesie badania zużycia energii.

Poniżej przedstawiono podglądy sygnałów w programie **UaExpert**. Podzielone one zostały na kilka głównych grup:



- **AGV Status:**



Energy consumption - aktualny poziom zużycia energii

Power Level - aktualny poziom naładowania baterii

Status number - status (1 = postój, 2 = załadunek, 3 = wyładunek, 4 = przejazd, 5 = ładowanie, 6 = awaria)

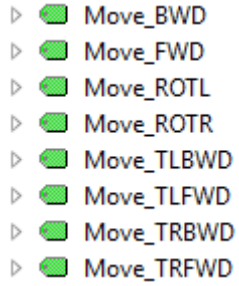
Virtual_Stop - zatrzymanie

PositionAngle - aktualna orientacja

PositionX - aktualna pozycja X

PositionY - aktualna pozycja Y

- **Commands (są to flagi 0/1) – aktywne polecenie jazdy:**



Move BWD - ruch do tyłu

Move FWD - ruch do przodu

Move ROTL - skręt w lewo w miejscu

Move ROTR - skręt w prawo w miejscu

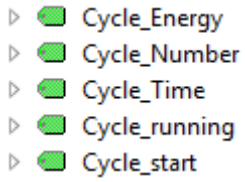
Move TLBWD - skręt w lewo podczas jazdy do tyłu

Move TLFWD - skręt w lewo podczas jazdy do przodu

Move TRBWD - skręt w prawo podczas jazdy do tyłu

Move TRFWD - skręt w prawo podczas jazdy do przodu

- **Current Cycle – informacje na temat aktualnego przejazdu:**



Cycle Energy - energia zużyta w trakcie przejazdu

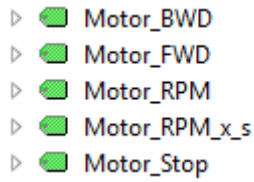
Cycle Number - numer porządkowy przejazdu

Cycle Time - czas trwania przejazdu

Cycle running - flaga oznaczająca wykonywanie przejazdu

Cycle start - flaga oznaczająca rozpoczęcie przejazdu (aktywna na kilka chwil)

- **Motor L – lewy silnik napędowy:**



Motor BWD - nakaz ruchu wprzód

Motor FWD - nakaz ruchu w tył

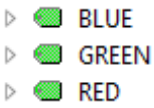
Motor RPM - obr/min napędu

Motor RPM xs - przebyty dystans

Motor Stop - zakaz ruchu

- **Motor R – prawy silnik napędowy, analogicznie do lewego**

- **RGB 1 – lewy sygnalizacyjny pasek ledowy:**



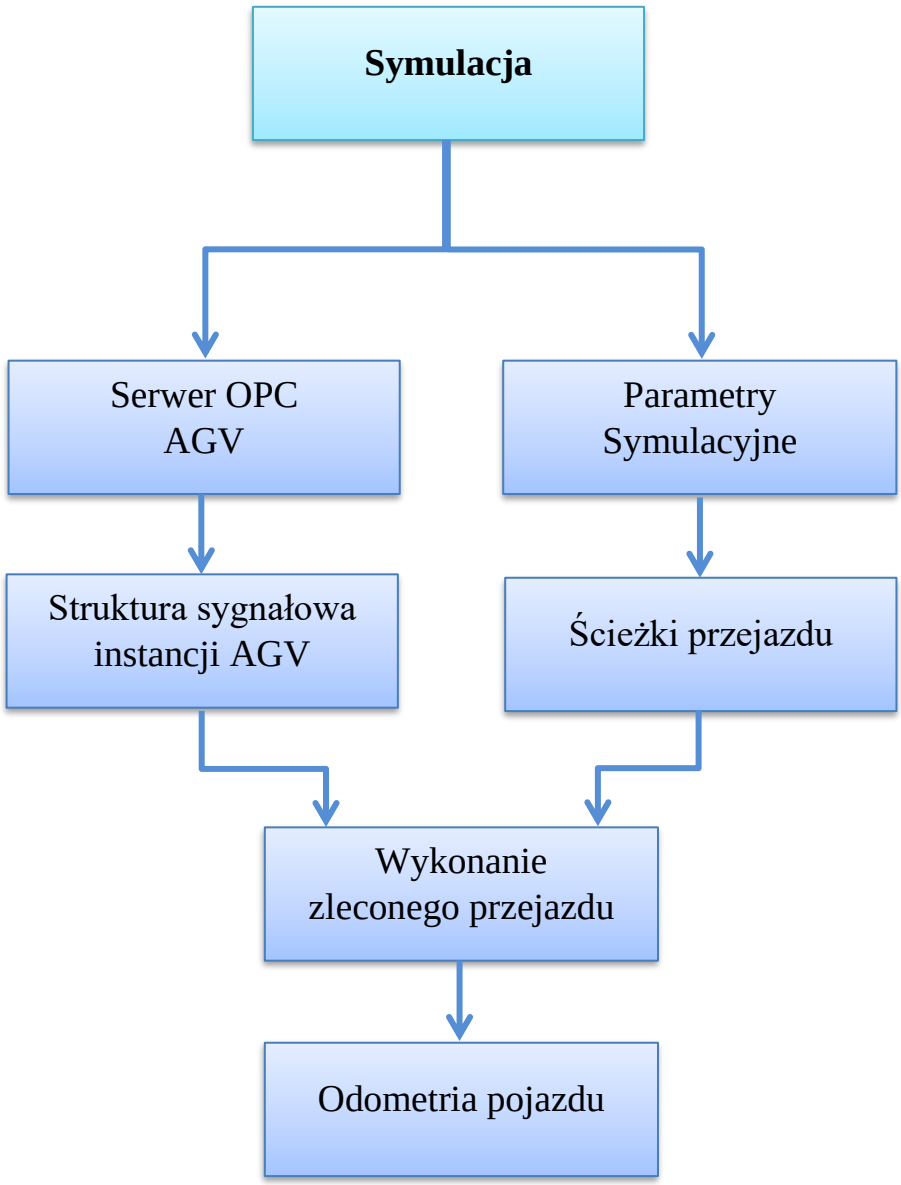
BLUE - natężenie koloru niebieskiego

RED - natężenie koloru czerwonego

GREEN - natężenie koloru zielonego

- RGB 2 – prawy sygnalizacyjny pasek ledowy, analogicznie do lewego

3.3 Struktura i opis programu



3.4 Symulacja

Główny moduł opiera się o tworzony za pomocą klasy ServerManager() serwer OPC. Na niego wprowadzana jest instancja klasy AGVInstance(), będąca odpowiednikiem pojedynczego AGV. Po utworzeniu obiektu AGV, na zasadzie działania sterownika PLC w pętli odczytywane są zmienne i parametry wejściowe, na podstawie których ustawiane są zmienne wyjściowe. Wszystko to dzieje się w trakcie jednego cyklu, którego długość trwania determinowana jest parametrem sim_rate o jednostce Hz (czas trwania jednego cyklu[s] = 1/sim_rate). Poniżej zaprezentowano utworzenie serwera, instancji AGV oraz zapoczątkowanie pętli rutyny pojazdu AGV.

```
import time
import Classes

# Available routes creation
route = Classes.Route()
route.example()  # fill route class with exemplary routes



# OPC UA server model
server = Classes.ServerManager()

#AGV instance create
AGV1 = Classes.AGVInstance(server.addspace, server.master_node, "AGV1")

#Simulation Parameters
Sim = Classes.Simulation(server.addspace, server.master_node)
Sim.sim_rate.set_value(5)# default sampling rate in [Hz]

# ====================== R O U T I N E ===========================

while True:  # AGV routine

    route.select_path(Sim)    # select route to be executed
    start = time.time()    # cycle time start after selecting path

    route.execute_path(AGV1, Sim)

    elapsed_time = round(time.time() - start)
    print('cycle time: {}s'.format(elapsed_time))
```

Serwer OPC AGV

Serwer, będący obiektem klasy ServerManager, tworzony jest przy wykorzystaniu modułu biblioteki opcua na podstawie adresu url (w tym przypadku localhost) serwera oraz przestrzeni adresowej (tutaj „OPCUA_AGV_SIMULATOR_SERVER”). Znając wspomniany adres url możemy się z nim połączyć i podejrzeć (lub nawet nadpisać w przypadku parametrów symulacyjnych) wszystkie sygnały symulatora.

```
#========================== Server class ==================================
class ServerManager:
    def __init__(self):
        self.server = Server()
        self.url = "opc.tcp://127.0.0.1:48030"
        self.server.set_endpoint(self.url)
        name = "OPCUA_AGV_SIMULATOR_SERVER"
        self.addspace = self.server.register_namespace(name)  # model namespace
        self.master_node = self.server.get_objects_node()
        self.server.start()
        print('Server started at {}'.format(self.url))
```

Parametry symulacyjne

Parametry symulacyjne służą niejako do sterowania symulacją i zaimplementowane zostały w klasie AGV i służą do ustawienia: szybkości symulacji, wybranego zlecenia przejazdu. Sterować można nimi dzięki umożliwieniu nadpisywania ich wartości na serwerze OPC.

```
class Simulation:
    def __init__(self, addspace, master_node):
        # Root AGV Node
        self.sim_node = master_node.add_object(addspace, "Simulation_Parameters")

        self.sim_rate = self.sim_node.add_variable(addspace, "Sim_rate", 1)
        self.sim_rate.set_writable(writable=True)
        self.chosen_path = self.sim_node.add_variable(addspace, "Chosen_path", 0)
        self.chosen_path.set_writable(writable=True)
```

(Dodane zostaną również parametry samego AGV: maksymalna prędkość obrotowa silników, przyspieszenie, średnica kół, szerokość pojazdu + dodatkowe opcje sterowania symulacją: zatrzymanie przejazdu, restart symulacji itp.)

Struktura sygnałowa instancji AGV

W obiekcie klasy AGVInstance generowana jest struktura sygnałów, opisanych w rozdziale 2. Tak jak zostało to przedstawione, tworzą one drzewo węzłów(nodes) z podziałem na grupy główne (metodą add_object). Poniżej zaprezentowano implementację węzłów głównych oraz podgrupę zmiennych (metodą add_variable) wraz z wartościami początkowymi przykładowo dla węzła „AGV_Status”:

```python
# ============ AGV instance class ========================================================
class AGVInstance:

        # ++++++++++++++++++++++ CONSTRUCTOR ++++++++++++++++++++++++++++++
        # ====================== CONSTRUCT NODES ==========================
    def __init__(self, addspace, master_node, name):
        # Root AGV Node
        self.AGV_node = master_node.add_object(addspace, name)

        # Main Nodes
        self.AGV_Status = self.AGV_node.add_object(addspace, "AGV_Status")
        self.AGV_Actions = self.AGV_node.add_object(addspace, "Commands")
        self.AGV_Current_Cycle = self.AGV_node.add_object(addspace, "Current_Cycle")
        self.Motor_L = self.AGV_node.add_object(addspace, "Motor_L")
        self.Motor_R = self.AGV_node.add_object(addspace, "Motor_R")
        self.RGB_1 = self.AGV_node.add_object(addspace, "RGB_1")
        self.RGB_2 = self.AGV_node.add_object(addspace, "RGB_2")

        # AGV_Status node
        self.eng_cons = self.AGV_Status.add_variable(addspace, "Energy_consumption", 0)
        self.power_lvl = self.AGV_Status.add_variable(addspace, "Power_Level", 0)
        self.status_no = self.AGV_Status.add_variable(addspace, "Status_number", 1)
        self.positionX = self.AGV_Status.add_variable(addspace, "positionX", 0.0)
        self.positionY = self.AGV_Status.add_variable(addspace, "positionY", 0.0)
        self.positionAngle = self.AGV_Status.add_variable(addspace, "positionAngle", 0.0)
        self.virtual_stop = self.AGV_Status.add_variable(addspace, "Virtual_Stop", True)
```

Ścieżki przejazdu

Ścieżka przejazdu w rozumieniu symulatora składa się z odcinków elementarnych – oznacza to, że w jednym kroku symulacji wykonywane są czynności potrzebne do przebycia zadanego odcinka elementarnego. Kolejnymi odcinkami elementarnymi kodowana jest lista poleceń ścieżki, gdzie ‘F’ to odcinek znajdujący się na wprost ‘R’ i ‘L’ odcinki wymuszające skręt odpowiednio w prawo lub w lewo.

W przypadku wyboru jednej ze ścieżek, przepisywana jest ona do listy active[].

Ścieżki przejazdu AGV zawarte są w obiekcie klasy Route. Przechowywane są one w liście route[]. Dodawanie nowej ścieżki odbywa się poprzez metodę add_path(), gdzie jako argument należy podać ciąg

znaków, odpowiadających odcinkom elementarnym (dla ułatwienia wpisanie np. liczby 10 poskutkuje wprowadzeniem do listy ścieżki dziesięciu poleceń jazdy do przodu ‘F’).

Dwie przykładowe ścieżki podano w metodzie example() poniżej:

```
36  #========================== Route class ============
37  class Route:
38      def __init__(self):
39          self.executing = False
40          self.idle = True
41          self.route = []
42          self.active = []
43
44          # odometria
45          self.positionX = [0.0]
46          self.positionY = [0.0]
47          self.angle = [270.0]
48
49      def add_path(self, path):
50          self.route.append([])
51          for i in path:
52              if type(i).__name__ == 'int' and i > 0:
53                  for x in range(i):
54                      self.route[-1].append('F')
55              else:
56                  for x in range(12):
57                      self.route[-1].append(i)
58
59      def example(self):
60          r1 = [10, 'L', 5, 'R']
61          r2 = [5, 'R', 'R']
62          self.add_path(r1)
63          self.add_path(r2)
```

Wykonanie przejazdu

Metoda odpowiedzialna za symulację wykonania zlecenia przez AGV wywoływana jest w głównym module symulatora, w pętli gdzie metoda select_path() oczekuje na zmianę parametru chosen_path z przedziału [1; liczba ścieżek zapisanych w obiekcie klasy Route].

Po wyborze numeru ścieżki następuje wykonanie przejazdu po kolei dla każdego odcinka elementarnego ścieżki. Przy każdym kroku (odcinku) w czasie zdefiniowanym parametrem symulacyjnym sim_rate wykonywane są trzy metody, odpowiedzialne za aktualizację sygnałów na serwerze OPC:

routine_command() – aktualizacja sygnałów sterowania (commands)

routine_motors() – aktualizacja prędkości obrotowych silników

routine lights() – aktualizacja ledów sygnalizacyjnych

Przejazd symulowany jest w metodzie execute_path() klasy Route().

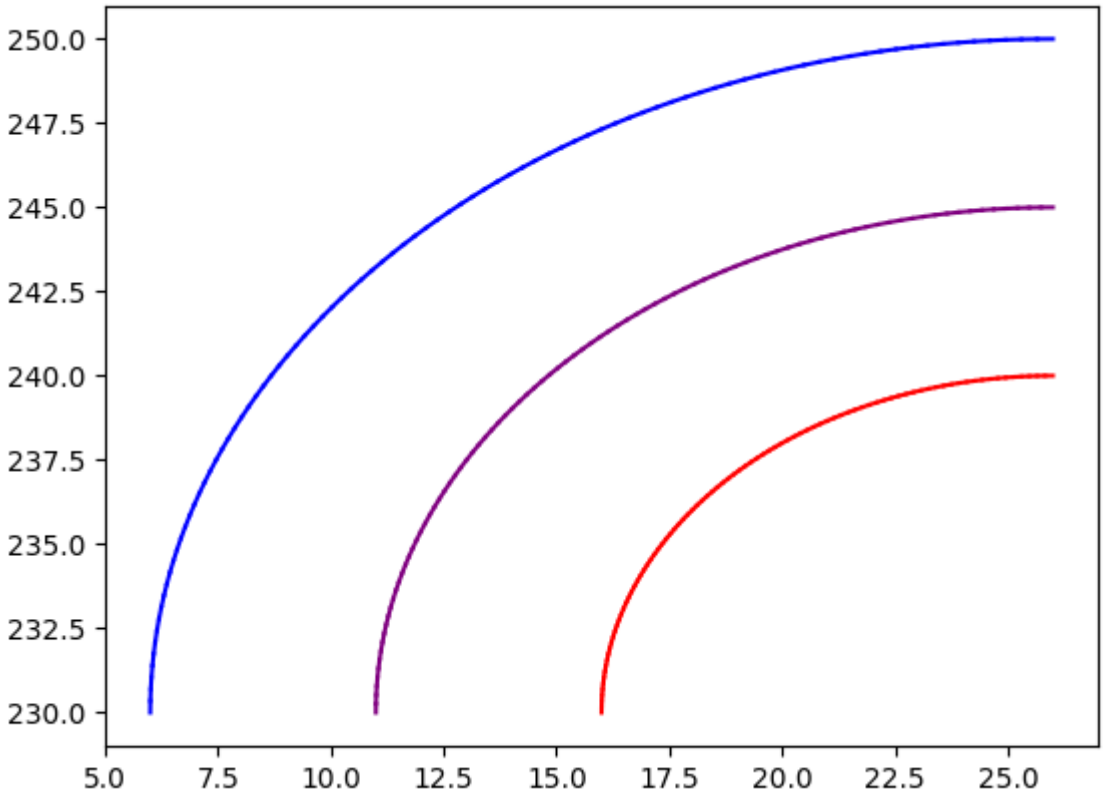
```
 99      def execute_path(self, AGV, sim):
100          AGV.cycle_time.set_value(0)
101          AGV.cycle_no.set_value(AGV.cycle_no.get_value()+1)
102          AGV.virtual_stop.set_value(False)
103          AGV.cycle_start.set_value(True)
104          time.sleep(5 / sim.sim_rate.get_value())
105          AGV.cycle_time.set_value(5)
106          AGV.cycle_start.set_value(False)
107          AGV.cycle_running.set_value(True)
108
109          self.positionX = [10.0]
110          self.positionY = [0.0]
111          self.angle = [0.0]
112          for r in self.active:  # route execution
113              AGV.routine_command(r)
114              AGV.routine_motors()
115              AGV.routine_lights()
116              time.sleep(1 / sim.sim_rate.get_value())
117              AGV.cycle_time.set_value(AGV.cycle_time.get_value()+1)
118          AGV.brake()
119          AGV.cycle_running.set_value(False)
120          AGV.virtual_stop.set_value(True)
```

Odometria pojazdu

Pojazd AGV porusza się po płaszczyźnie opisanej osiami X, Y. Zaimplementowany został moduł „odometria”, pozwalający na ustalenie końcowych pozycji(x,y) i orientacji w stopniach dla aktualnych prędkości kół lewego i prawego oraz zadanej szerokości pojazdu i warunków początkowych PositionX (wartość pozycji x), PositionY (wartość pozycji y) oraz PositionAngle (orientacja wyrażona w stopniach, gdzie N=północ=0, W=wschód=90, S=południe=180, E=zachód=270).

Poniżej przykładowa implementacja dla prędkości v1=50 (lewe koło), v2=25 (prawe), pozycja startowa [11,230], szerokość=10, orientacja=0.

```python
##################################################################
##########  Algorytm dla pojedynczego skretu  ###################
def turn(l_wheel, r_wheel, veh_center, veh_width, v1, v2):
    c = array(veh_center)
    l_wheel = array(l_wheel)
    r_wheel = array(r_wheel)
    w = veh_width/2

    # promien skretu (dwukolowca)
    r, L1, L2 = turn_radius(v1,v2,w)

    # srodek okregu obrotu
    a_b = line_coefficients(l_wheel, r_wheel) # wspolczynniki a,b prostej y
    r = r_sign(a_b)*r # 1*r lub -1*r
    turn_center = line_point(c, a_b, r)

    right = czy_w_prawo(v1,v2)
    starting_angle = -orientation(l_wheel, r_wheel)
    finishing_angle = end_turn(starting_angle)

    PL,PR,PC = turn_points(l_wheel, starting_angle, finishing_angle, turn_center,r,w,right)
    # PL=PointWheelLeft, PR=PointWheelRight, PC=PointWheelCenter
    X = PC[0]
    Y = PC[1]
    angle = orientation(PL, PR)
    return X,Y,angle
```



Rys. 2 Przykładowy wynik przejazdu elementarnego dla RPM lewego koła=50, RPM prawego koła=25

W procesie wyznaczania wartości wyjściowych pozycji skorzystano wyłącznie z matematycznych funkcji trygonometrycznych środowiska Python. Ze względu na dużą objętość, lecz zarazem prostotę obliczeń podfunkcje nie zostały przedstawione.

3.5 Wizualizacja

Część wizualizacji ma za zadanie zobrazować położenie robota, wydawanie podstawowych komend oraz przedstawić aktualny stan robota. Dodatkowo zakłada się wykorzystanie w systemie wizualizacji interpretacji danych generowanych w formacie XML przedstawiających zaplanowaną trasę przejazdu. Pierwszy semestr prac nad projektem został ukończony w programie InTouch z rodziny programów Wonderware jako niezależny projekt prezentujący położenie robota na hali oraz odczyt danych z pliku XML wyznaczającego kolejne punkty trasy ruchu robota.

Główne kierunki rozwoju realizacji projektu:

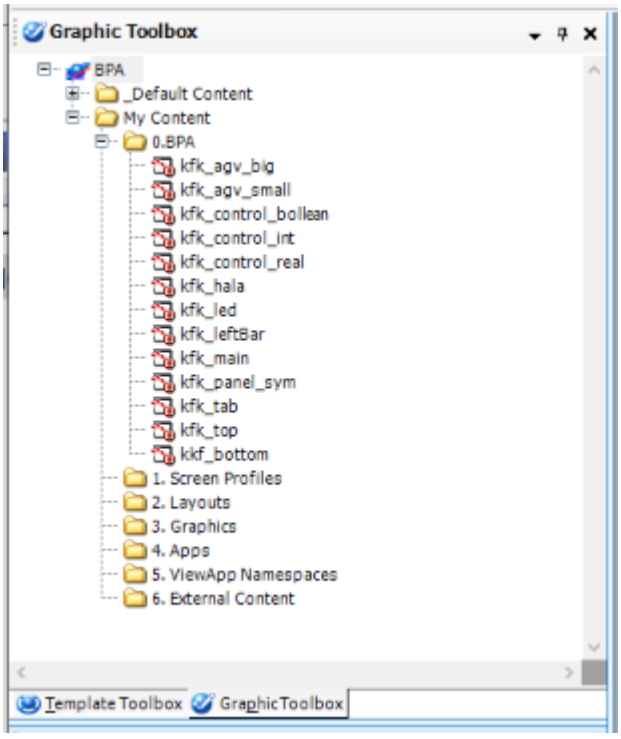
- Przeniesienie projektu z programu InTouch do rozbudowanego środowiska Archestra IDE. Oba programy należą do rodziny programów Wonderware tworzących jedno rozbudowane środowisko do wizualizacji procesów produkcyjnych.
- Implementacja programu realizującego odczyt danych z plików XML napisanego już w programie InTouch

- Nawiązanie komunikacja z serwerem OPC jako głównego medium wymiany danych między robotami a serwerami aplikacyjnymi

Realizacja założeń:

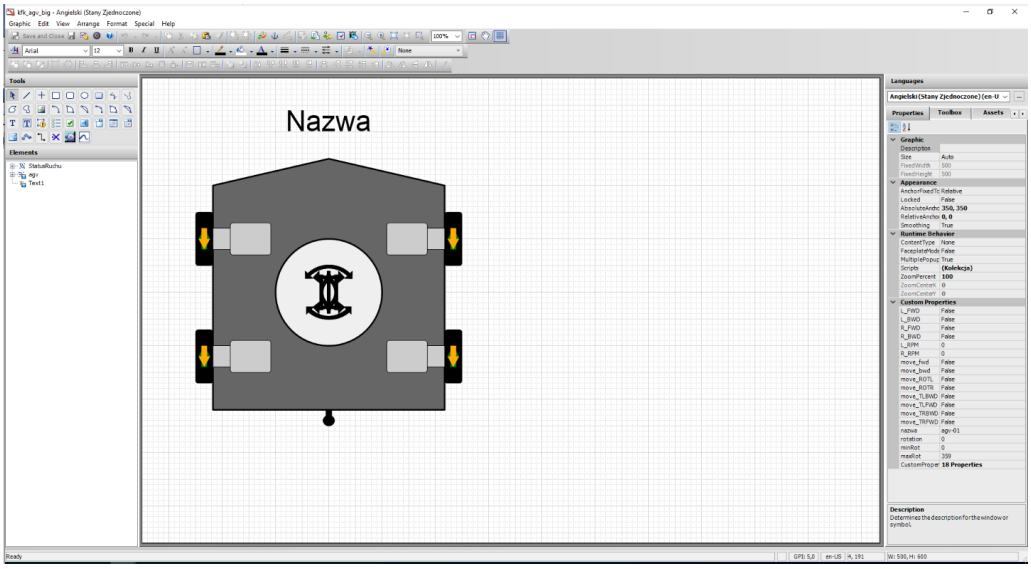
- Przeniesienie rozwiązania zaimplementowanego w programie InTouch pozwala na prostą rozbudowę programu o dodatkowe roboty czy hale produkcyjne tak aby wszystko było spójnie i hermetycznie zamknięte w jednym systemie oraz dawało prosty i szybki podgląd stanu procesu produkcyjnego. Dodatkowo środowisko projektowe Archestra IDE daje większe możliwości tworzenia stricte nowych obiektów jak i generowania indywidualnych szablonów dopasowanych do tych obiektów, które reprezentują.
- Implementacja napisanego okazała się kłopotliwa ze względu na różnice w funkcjach przygotowanych w obu środowiskach i całość kodu na podstawie istniejącej logiki musi zostać napisana od nowa
- Nawiązanie komunikacji z serwerem OPC pozwoliło przetestować główne medium wymiany danych pomiędzy poszczególnymi modułami oraz możliwość wydawania poleceń robotowi poprzez w/w medium.

Biblioteka graficzna

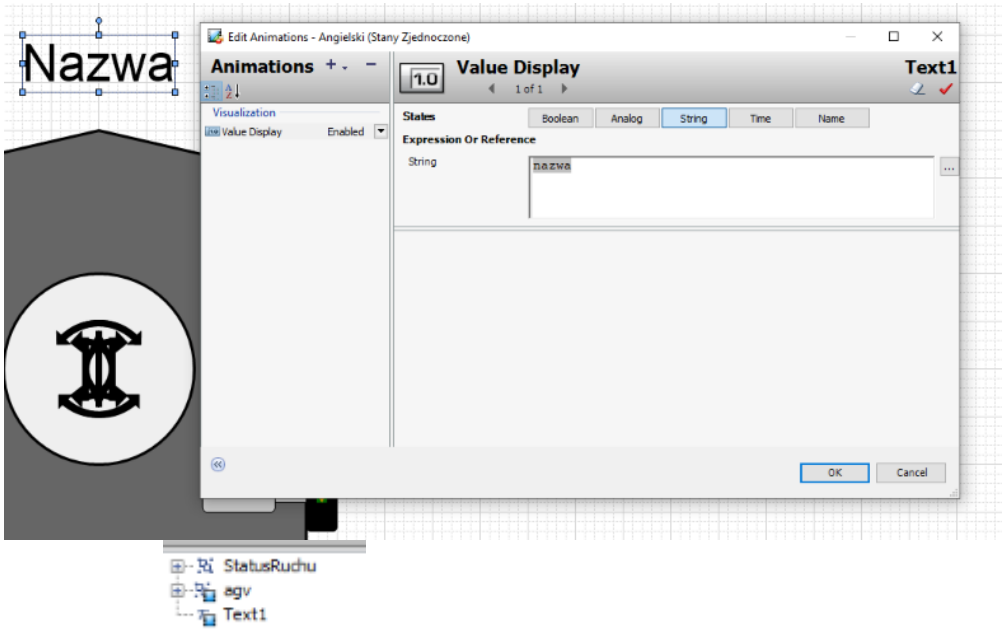


W systemie wonderware istnieje predefiniowana biblioteka zawierająca zbiór gotowych rozwiązań w zakładce „Graphic Toolbox”. Na potrzeby projektu został założony nowy folder „0.BPA”, w którym zostały zamieszczony wzory interaktywnych obrazów przygotowanych specjalnie dla tego projektu.

Poniżej przedstawiony został zrzut ekranu prezentujący okno projektowe jednego z przygotowanych wzorów. Wzór zawiera obraz robota AGV wraz z podstawowymi informacjami o jego stanie i realizowanych zadaniach.



Taką bibliotekę obrazków przygotowuje się w celu jej prostej implementacji w przygotowanych obiektach. Grafika oprócz samego obrazu posiada funkcje przypisane do widocznych elementów oraz przygotowane zmienne wewnętrzne które podpięte do rzeczywistych sygnałów mogą sterować zaimplementowanymi funkcjami.

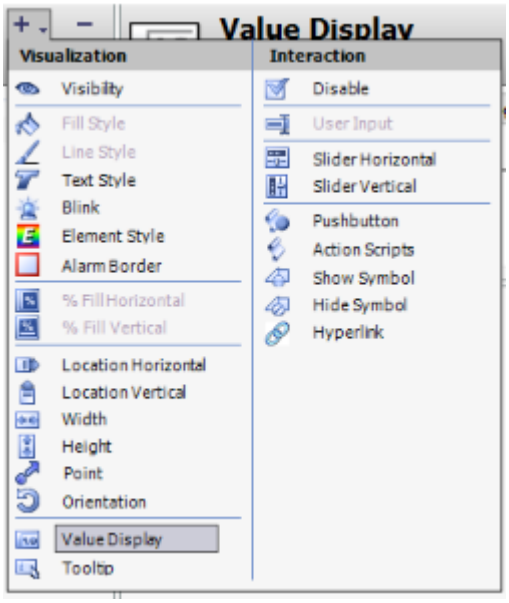


jasny niebieski kwadrat widoczny przy elemencie lub grupie elementów zebranych w jeden obiekt informuje że istnieje do niego przypisana przynajmniej jedna funkcja.

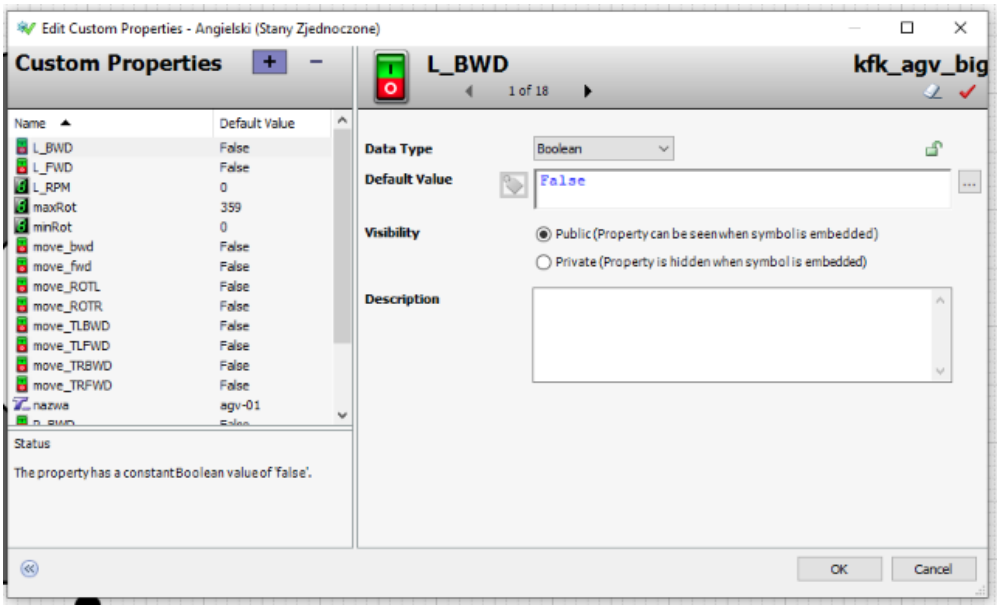
Obiekt „Text1” posiada jedną funkcję widoczną na zdjęciu poniżej:

Działanie funkcji to podmiana tekstu widniejącego w oknie projektowym na nazwę w zmiennej „nazwa”.

Poniżej zaprezentowany został zbiór wszystkich dostępnych funkcji możliwych do wykorzystania podczas animacji obiektów:



Zmienna zaprezentowana w powyższym przykładzie jest zdefiniowana dla całego okna graficznego. Zmienne tworzone są w oknie „Custom Properties”



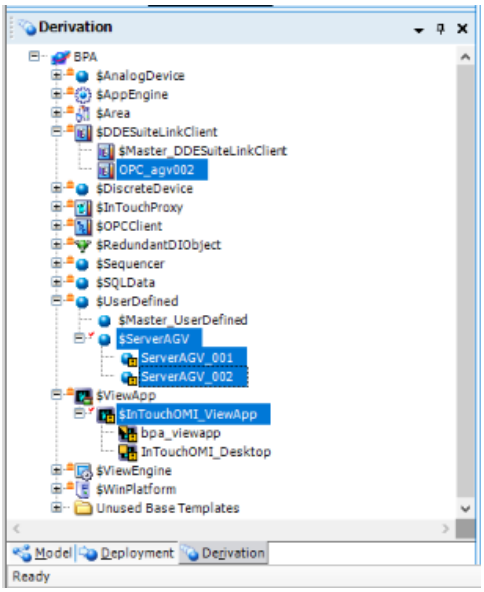
Można tam określić ich typ oraz wartości początkowe. Do tych zmiennych w oknach obiektów zostają przypisane rzeczywiste sygnały.

Obiekty

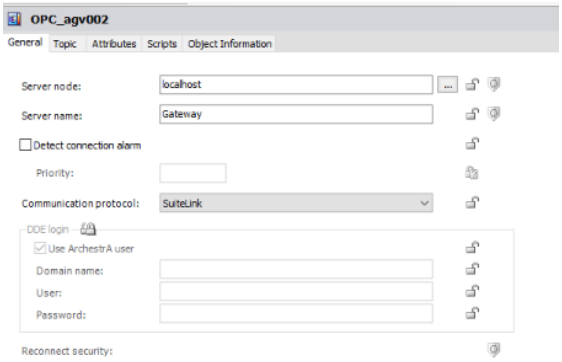
W środowisku oprócz biblioteki obrazów zostały przygotowane już obiekty dopasowane do odpowiednich zadań. W projekcie zostały użyte trzy główne kategorie obiektów:

- Komunikacyjny – server OPC
- Obiektowy – robot AGV
- Aplikacyjny – środowisko InTouch

Na zdjęciu poniżej zostały zaznaczone wszystkie elementy przygotowane na potrzeby projektu:

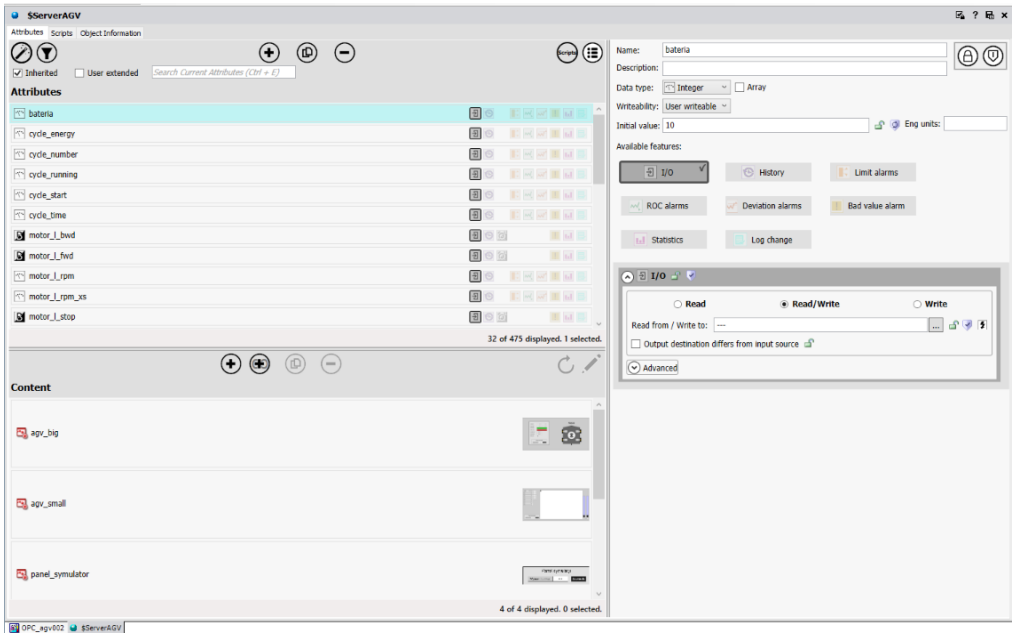


Obiekt OPC_agv002 służy do komunikacji z wybranym serwerem testowym. Każdy podłączony serwer posiada przynajmniej jeden taki obiekt do niego przypisany. Poniżej zamieszczony został panel funkcyjny obiektu DDE SuiteLink:



Szczegółowe informacje o jego konfiguracji najlepiej zaczerpnąć z filmu przygotowanego przez dystrybutora znajdującego się pod linkiem: https://www.youtube.com/watch?v=1U8cxXH5CWM

$ServerAGV to obiekt, który stanowi szablon dla obiektów $ServerAGV_001 oraz $ServerAGV_002. W nim jest zapisany przepis na obiekty z niego tworzone oraz wszystkie reguły. Jeżeli jakaś opcja zostanie zablokowana w szablonie to nie ma możliwości modyfikacji jej w obiekcie zbudowanym na tym szablonie. Okno konfiguracji szablonu i gotowego obiektu wyglądają identycznie i prezentują się następująco:



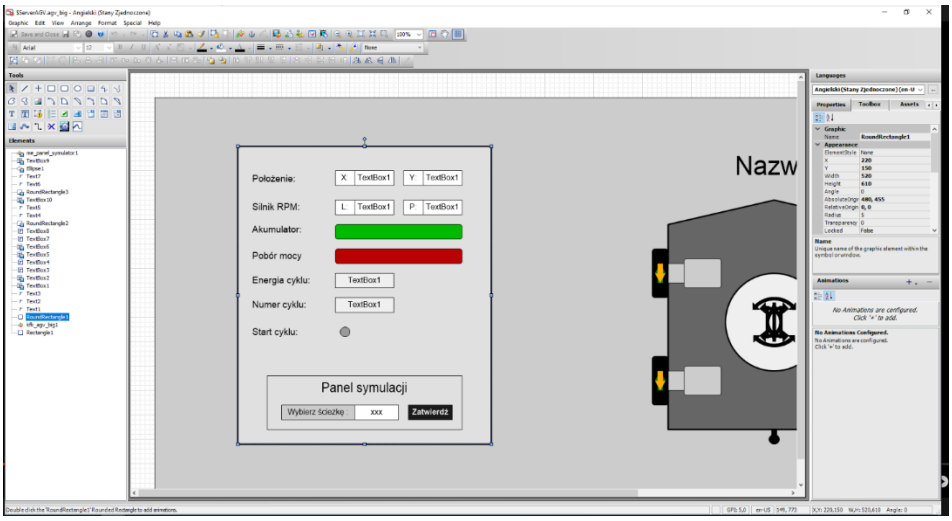
Na konfigurację składają się trzy zakładki. Z czego w projekcie została użyta tylko jedna „Atributes” i to na niej skupi się opis.

Z lewej strony w górnej części okna zostały wylistowane atrybuty przypisane do danego obiektu lub szablonu. Natomiast każdy z atrybutów może posiadać dodatkowe funkcje takie jak:

- adres I/O – adresy na serwerze OPC czy sterowniku – używana w projekcie
- typ
- wartość domyślna
- nazwa
- opis

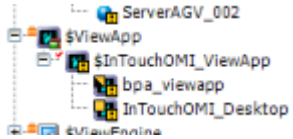
Wyżej opisane funkcje są widoczne po prawej stronie i jest ich o wiele więcej, opisane zostały tylko te wykorzystane w projekcie.

Pod listą atrybutów znajdują się utworzone okna dla danego obiektu. Okno edycji tych okien wygląda identycznie jak dla obrazków z „Graphic Toolbox”



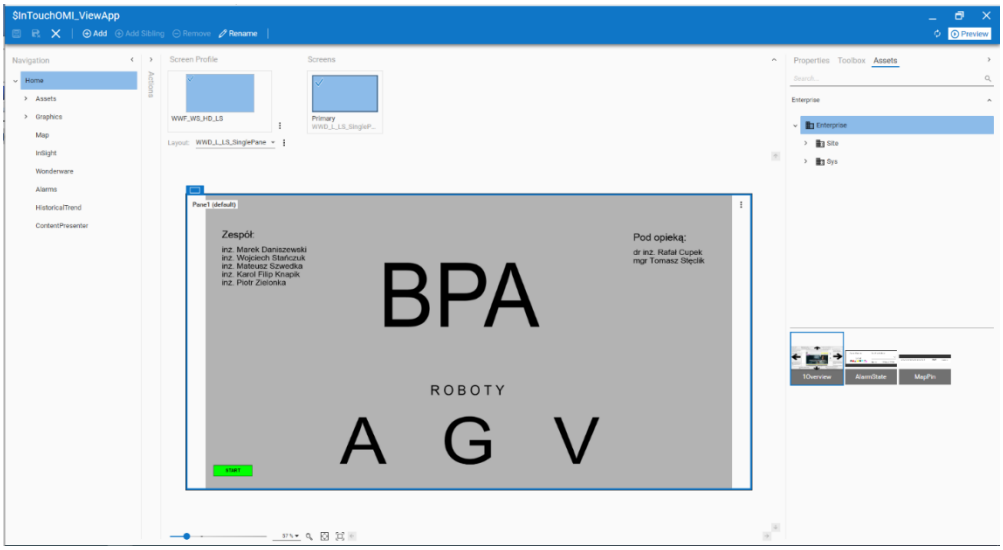
Okna te poza wyglądem dzielą wszystkie opisane funkcje, można w nich dodatkowo tworzyć zmienne dla okna obiektu oraz przypisywać wartości obiektu do tych zmiennych lub bezpośrednio do zmiennych zaciągniętych wraz z przygotowanymi szablonami w celu sterowania zaprojektowanymi animacjami.

Obiekt aplikacji $InTouchOMI_ViewApp



Widoczny w drzewku aplikacji służy do otwierania aplikacji w trybie „Runtime” na serwerze do testów funkcjonalności przygotowanej aplikacji. Taką aplikację można uruchomić jednocześnie na wielu urządzeniach które są reprezentowane, np. przez „bpa_viewapp”. Instancje pod jedną aplikacją, uruchomione na różnych urządzeniach podpiętych do systemu dają wgląd do procesu produkcyjnego tej samej aplikacji oraz możliwość otwierania na nich różnych okien. Wpłynięcie jednak na zmianę wartości parametrów na jednej z nich powoduję realne zmiany w całym prezentowanym systemie. Dzięki możliwości podpięcia wielu urządzeń do jednej aplikacji jest możliwość prezentowania tych samych danych w jednym czasie, w różnych miejscach, obsłudze hali, kierownikowi działu czy dyrektorowi całego zakładu.

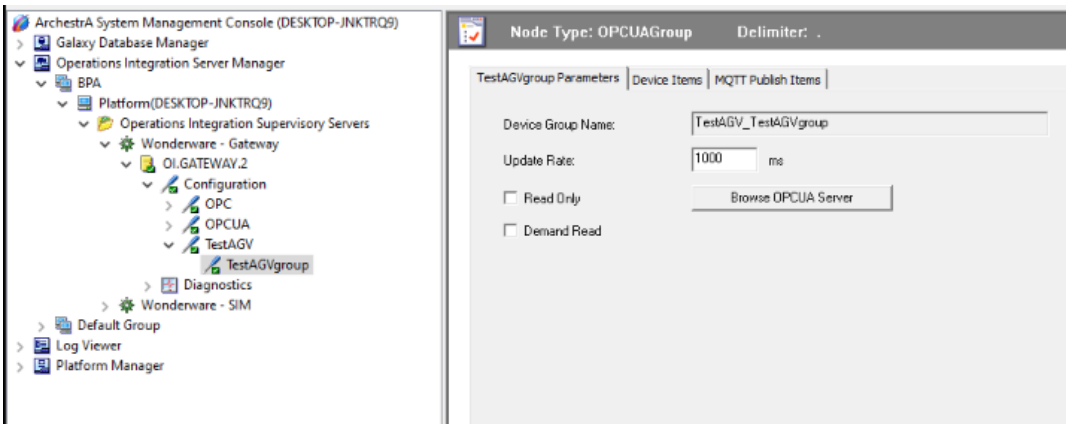
Każda nowa hala będzie posiadała swój obiekt $InTouchOMI_ViewApp wraz z instancjami podpiętych urządzeń. Po kliknięciu w obiekt aplikacji na serwerze zostaje uruchomione okno InTouch:



Po naciśnięciu przycisku „Preview” w prawym górnym rogu zostaje uruchomiona aplikacja w trybie runtime prezentująca wszystkie funkcje. Pomimo, że służy ona do testów developerskich to po podpięciu rzeczywistych sygnałów do systemu jest ona pełnoprawną wersją aplikacji i pozwala ona w pełni na kontrolę nad systemem.

System Managment Console

Ostatni element systemu który w połączeniu obiektem komunikacyjnym pozwolił na podłączenie serwera OPC. Dokładny opis konfiguracji został pokazany w załączonym linku do filmu gdzie wytłumaczono cały proces. Poniżej zamieszczone zostało zdjęcie prezentujące wynik końcowy dla opisywanego projektu:



Wnioski

Podczas tego semestru udało się przenieść rozwiązanie przygotowane w poprzednim semestrze ze środowiska InTouch do środowiska ArchestraIDE wraz z implementacją wszystkich głównych zalet nowego środowiska które można określić mianem „a’la obiektowym” ze względu na cechy takie jak:

- Hermetyzacja danych
- Przeciążenia zmiennych i funkcji
- Klasyfikacja

Ostatnim minusem który został i jest nie do przeskoczenia w tego typu rozwiązaniu jest konieczność z góry określonej liczby sygnałów. W konsekwencji tego ograniczenia nie można w trakcie działania programu dodać kolejnego robota poprzez jedno kliknięcie jeżeli nie został on, jego obiekt i wszystkie instancje pochodne potrzebne do jego prawidłowego działania wcześniej utworzone w systemie.
Jak widać w raporcie wybrane środowisko jest środowiskiem graficznym i cechuje się wysokim poziomem parametryzacji, a kody występujące w aplikacji są jej marginalnym elementem. Niestety w trakcie semestru nie udało się przenieść implementacji kodu interpretującego tekst w kodzie XML do odczytywania zaplanowanej trasy przejazdu robota. Powodem jest poziom skomplikowania przygotowania odpowiedniej konfiguracji systemu środowiska Wonderware jak również środowiska uruchomieniowego Windows 10. Do tego potrzebna była wiedza specjalistyczna wiedza nabyta w zakładzie pracy, doświadczenie nabyte podczas prac w różnego rodzaju projektach produkcyjnych oraz kontakt z polskim dystrybutorem oprogramowania Wonderware, firmą Astor.

Na kolejny semestr głównymi założeniami prac rozwojowych nad projektem są:

- implementacja programu odczytującego pliki XML
- testy na rzeczywistych pojazdach AGV
- uzupełnienie funkcji wizualizacji o prezentację informacji koniecznych do łatwej obsługi systemu

4. Moduł klasteryzacji i predykcji

Proces klasteryzacji (zwanej również analizą skupień lub grupowaniem) opiera się na podzieleniu danego zbioru wartości na określoną liczbę klastrów (rozłącznych grup) w taki sposób, aby każde z otrzymanych w jego rezultacie skupisk zawierało obiekty możliwie podobne do siebie (zgodnie z przyjętym kryterium), a zarazem niepodobne do obiektów z pozostałych podzbiorów. Całość przebiega w sposób nienadzorowany. Oznacza to, że grupy dobierane są tylko i wyłącznie na podstawie zmierzonych wartości cech opisujących dane obiekty. Wynika, więc z tego, że aby dokonać klasteryzacji nie trzeba mieć wstępnej wiedzy o ich przynależności do jednej ze znanych wcześniej grup. Pomaga ona jednak przy interpretacji wyników klasteryzacji.

Można więc stwierdzić, że klasteryzacja jest metodą o charakterze odkrywczym. Oznacza to, że może być ona wykorzystana do wykrycia pewnych nieznanych wcześniej zależności między obserwowanymi obiektami na podstawie posiadanych na ich temat informacji. Ponadto, zastosowanie klasteryzacji pomoże w zredukowaniu dużej liczby pierwotnie posiadanych danych do paru podstawowych kategorii, których można użyć jako przedmioty dalszej analizy.

4.1 Ogólny schemat przebiegu klasteryzacji

Pierwszym krokiem jest przeprowadzenie pomiaru wartości cech, które opisują badane przez nas obiekty. Następnie należy dokonać klasteryzacji wektorów tych cech i bazując na pozyskanej wiedzy, podjąć próbę nadania tym obiektom interpretacji. Mimo prosto brzmiącej teorii, proces ten jest trudny w praktycznej realizacji, zwłaszcza w przypadku, gdy mamy do czynienia z dużymi zbiorami danych.

Wybrane metody:

K-means (k-centroidów)

Metoda ta zaliczana jest do grupy algorytmów niehierarchicznych, tzn. takich dla których konieczne jest wcześniejsze zadeklarowanie ilości klastrów. Polega ona na tworzeniu k różnych skupień możliwie od siebie odmiennych i przenoszeniu między nimi obiektów, dopóki zmienności wewnątrz grup oraz między nimi nie zostaną w pełni zoptymalizowane.

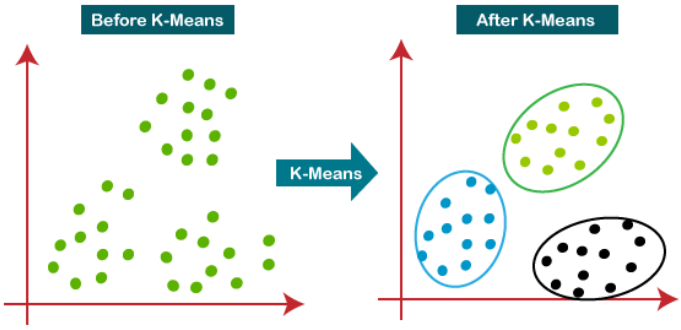


Fig. 1, diagram przedstawiający idee metody k-means, źródło: javatpoint.com

Schemat działania algorytmu:

1. **Ustalenie liczby klastrów**
   Ilość grup można zdefiniować w sposób umowny, a na późniejszym etapie dokonywać korekty w celu poprawienia otrzymanych wyników. Wybór ten można również oprzeć na podstawie innych analiz
2. **Założenie wstępnych środków skupień**
   Dobór centroidów, czyli środków skupień, odbywa się na kilka sposobów:
   a. Wybór k pierwszych obserwacji
   b. Losowy wybór k obserwacji
   c. Dobór oparty o zmaksymalizowanie odległości skupień
3. **Wyliczenie odległości obiektów od centroidów**
   Etap ten wpływa bezpośrednio na to, jakie obserwacje będą uznane za podobne, a jakie za zbyt zróżnicowane względem siebie. Jest to więc istotna faza algorytmu. Najczęściej stosuje się odległość euklidesową lub odległość Czebyszewa.
   Odległość euklidesowa:

$$d(x,\ y)\ = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

   Odległość Czebyszewa:

$$d(x,\ y)\ =\ max_i|x_i - y_i|$$

   gdzie:
   x_i, y_i *– wektory wartości cech porównywanych obiektów w przestrzeni p-wymiarowej*
4. **Przypisanie obiektów do grup**

W tym kroku następuje porównanie odległości od wszystkich skupień dla wybranej obserwacji, a następnie przypisanie jej do tej, której środek znajduje się najbliżej

5. **Zdefiniowanie nowych środków skupień**
   W większości przypadków nowym środkiem jest punkt, którego współrzędne to średnia arytmetyczna współrzędnych punktów wchodzących w skład bieżącego klastra.
6. **Powtarzanie kroków 3-5, dopóki warunek zatrzymania nie zostanie spełniony**
   Powszechnie stosowanym warunkiem zatrzymania jest ilość iteracji, która została założona na początku lub brak przesunięć między klastrami.

Grupowanie hierarchiczne

Nazwa tej metody pochodzi od struktury hierarchicznej (tworzonej przez stosowane w niej algorytmy) przyjmującej formę drzewa przedstawianego na wykresie zwanym dendrogramem. Przedstawia ono cały proces działania algorytmu oraz tworzenia się grup na skutek scalania lub dzielenia. Grupy należą do osi X, natomiast odległość między łączonymi skupiskami przedstawia oś y.
Metody hierarchiczne dzielą się na dwie grupy:

1. **Aglomeracyjne (łączące)**

W tym przypadku algorytm ma swój początek w momencie, gdy liczba grup jest równa liczbie obserwacji, a każda z obserwacji stanowi odrębną grupę. Grupy te są łączone w sposób taki, by ich wewnętrzna wariancja była możliwie jak najmniejsza, natomiast w stosunku do innych grup możliwie duża. Sposób działania zaimplementowany w tego typu algorytmach znajduje swoje zastosowanie w przypadku poszukiwania małych grup, czyli gdy proces może się zakończyć po kilku iteracjach.

2. **Deglomeracyjne (dzielące)**

Ta metoda rozpoczyna swoje działanie, gdy wszystkie obserwacje wchodzą w skład jednej grupy. Następnie wraz z kolejnymi iteracjami jej działania, grupy te są dzielone z uwzględnieniem tych samych kryteriów co w przypadku metody aglomeracyjnej, czyli miary odległości i wariancje między grupami. W przeciwieństwie do algorytmów łączących, metoda ta sprawdza się przy poszukiwaniu dużych grup.

Algorytmy te są jednak mało efektywne, ze względu na swoją złożoność obliczeniową $O(n^3)$ i pamięciową $O(n^2)$.

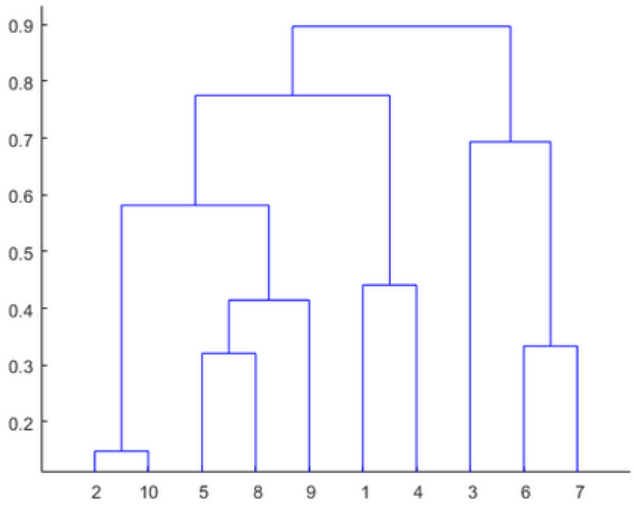


Fig. 2, dendogram przedstawiający idee działania grup hierarchicznych, źródło: mathworks.com

Grupowanie gęstościowe

Metoda ta polega na tworzeniu segmentów w oparciu o informacje na temat zagęszczenia obserwacji w przestrzeni, czyli odległości, które dzielą poszczególne obserwacje w danym obszarze. Z uwagi na to najlepiej sprawdza się ona w przypadku identyfikacji segmentów o nieregularnych kształtach. Nie umożliwia jednak wcześniejszego zdefiniowania liczby grup, można natomiast nieznacznie wpłynąć na to poprzez zmianę parametrów modelu. Do atrybutów, które należy przed rozpoczęciem działania algorytmu ustalić, zalicza się minimalna liczba obserwacji, które są potrzebne do zbudowania grupy oraz minimalna odległość epsilon definiująca sąsiada. W grupowaniu gęstościowym nie wszystkie obserwacje muszą zostać powiązane z grupami, gdyż w przypadku, gdy któraś z obserwacji nie będzie spełniać założonych kryteriów, zostanie uznana za odstającą. Kluczowym aspektem tej metody jest dobór

odpowiednich (choć niewielu) parametrów modelu, w taki sposób, aby efektem zastosowania algorytmu była optymalna liczba grup, a obserwacje uznane za nietypowe nie stanowiły większości zbioru.

DBSCAN

Algorytm DBSCAN jest najpopularniejszym algorytmem spośród przedstawicieli metod grupowania gęstościowego. Do jego zalet można zaliczyć przede wszystkim to, że zwraca dobre rezultaty, przy jednoczesnym zachowaniu relatywnie szybkiego działania. Jest odporny na wpływ obserwacji odstających, bardzo dobrze radząc sobie przy tym z grupami o niewypukłych kształtach. Z kolei jego główną wadą jest problematyczność w doborze odpowiednich parametrów modelu, których optymalizacja potrafi być czasochłonna z uwagi na brak sprecyzowanej i sprawdzonej metody.

Głównymi parametrami algorytmu są:

- **Epsilon (promień sąsiedztwa),** czyli taka minimalna odległość dzieląca dwie obserwacje, która pozwoli uznać je za sąsiadów
- **Min_samples,** czyli minimalna liczba obserwacji, jaka jest potrzebna, aby dana obserwacja mogła być uznana za punkt centralny danej grupy

Schemat działania:

1. Znalezienie sąsiadów danej obserwacji dla wartości bliższych niż założony epsilon
2. Przyjęcie punktów centralnych, na podstawie obserwacji mających co najmniej min_samples sąsiadów w odległości mniejszej niż epsilon
3. Złączenie w jedną grupę obserwacji spełniających założenie z punktu 2.
4. Przyłączenie do istniejących grup obserwacji, które nie zostały uznane za punkty centralne, ale znajdujące się w odległości epsilon
5. Zakończenie działania algorytmu, po czym następuje:
    a. przypisanie obserwacji nie należących do grup oraz nie posiadających w zasięgu epsilon nowych obserwacji do obserwacji granicznych
    b. przyłączenie obserwacji które nie zostały dodane do żadnej z grup, do obserwacji odstających

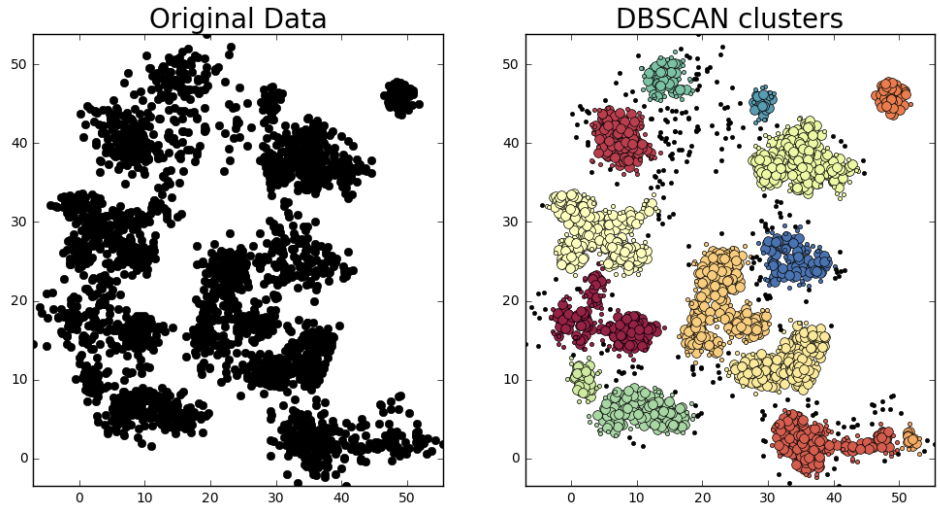


Fig. 3, diagram przedstawiający idee działania algorytmu DBSCAN, źródło: github.com

4.2. Wyniki badań klasteryzacji

W ramach projektu przeprowadzono badania algorytmów klasteryzacji, wykorzystując implementacje metod KMeans oraz DBSCAN w bibliotece scikit-learnco on chci dla języka python. W tym celu skorzystano z dwóch ogólnodostępnych zbiorów danych – Iris Dataset oraz Wine Dataset.

KMeans

Zgodnie z teorią dotyczącą tego algorytmu (zawartą we wcześniejszej części dokumentu), wymagane jest zdefiniowanie liczby klastrów (oznaczanej też literą "k") przed przystąpieniem do obliczeń. Można to zrobić doświadczalnie, poprzez zwiększanie liczby klastrów w każdej iteracji, jednak jest to mało wydajne rozwiązanie. Jednym ze skutecznych sposobów na jej wyznaczenie jest zastosowanie metody łokcia, polegającej na przedstawieniu wartości funkcji kosztu jaką uzyskuje się przy różnych wartościach k. Jeśli więc liczba ta wzrasta, średnie zniekształcenie zmniejszy się, a każdy klaster będzie składał się z mniejszej ilości obiektów składowych, będących jednocześnie bliżej swoich centroidów. Jednak wraz ze wzrostem, maleje poprawa w zakresie średniego zniekształcenia. Wartość, przy której zniekształcenie ulega znacznej zmianie nazywane jest łokciem, jest to moment, w którym należy zaprzestać dalszej iteracji.

Zdecydowano więc na implementację tego zagadnienia w kodzie programu. Ponadto, do wynikowego wykresu dodano również inny wykres, przedstawiający różnicę znormalizowanej wartości k pomniejszonej o odpowiadające jej zniekształcenie. Punkt, w którym różnica ta przyjmuje największą wartość jest jednoznaczny z optymalną wartością liczby klastrów.

Inną zastosowaną wartością kontrolną, jest współczynnik Silhouette wyznaczany poprzez średnią odległość wewnątrz klastrów i ich najbliższych sąsiadów. Wartości, które możemy otrzymać mieszczą się w przedziale <-1; 1>, gdzie wartości bliskie 1 są uznawane za dobrze zakwalifikowane, a bliskie –1 za źle. Jednym słowem, wartość ta jest miarą tego, jak bardzo dany obiekt jest podobny do swojego własnego klastra (w jakim stopniu jest z nim spójny) w porównaniu do innych.

Wartość Silhouette dla jednego punktu:

$$s = \frac{b - a}{max(a,\ b)}$$

gdzie:
a – średnia odległość danej obserwacji od wszystkich innych obserwacji w grupie
b – średnia odległość danej obserwacji od wszystkich innych obserwacji w innej najbliższej grupie
max(a, b) - większa wartość z wartości a i b.
Aby uzyskać współczynnik silhouette dla całego podziału, wystarczy wziąć pod uwagę średnią współczynników indywidualnych.

Iris Dataset

Fragment kodu odpowiedzialny za załadowanie danych i wyświetlenie pierwszych 10 linii:

```
iris = datasets.load_iris(as_frame=True)

    X = scale(iris.data)
    Y = iris.target
print(iris.data.head(10))
```

Fragment zbioru danych:

```
Iris Dataset - Kmeans Clustering
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
5                5.4               3.9                1.7               0.4
6                4.6               3.4                1.4               0.3
7                5.0               3.4                1.5               0.2
8                4.4               2.9                1.4               0.2
9                4.9               3.1                1.5               0.1
```

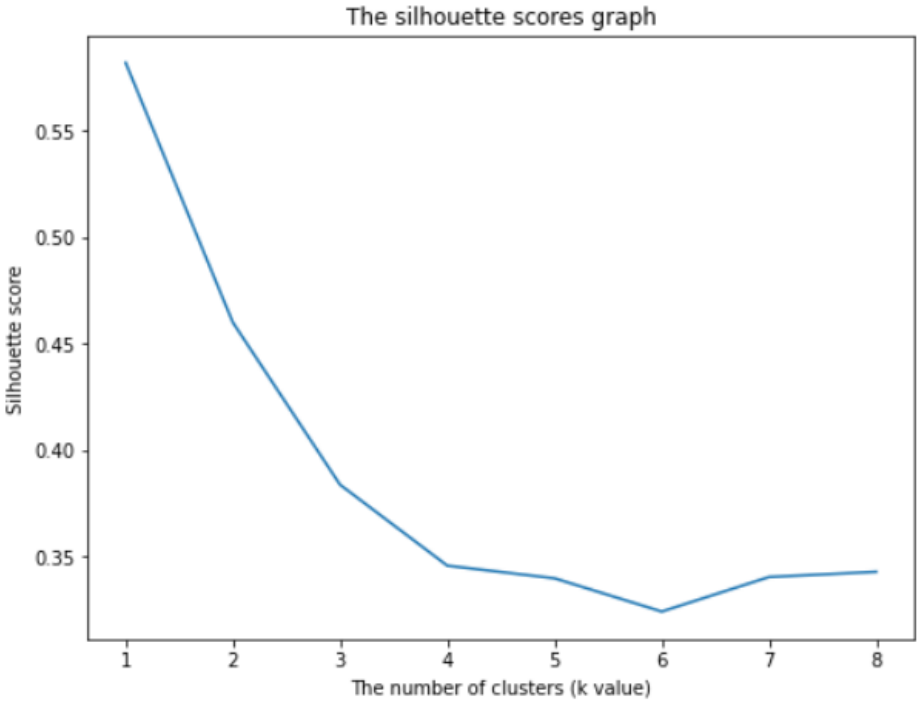
Fragment kodu odpowiedzialny za wywołanie metody łokcia, obliczenie współczynnika Silhouette dla całego przedziału k oraz wizualizację otrzymanych wyników:

```
distortions = []
silh = []
K = range(1, 10)


# Elbow method

for k in K:
    clustering = KMeans(n_clusters=k).fit(X)
    distortions.append(clustering.inertia_)


    # Silhouette
    if k >= 2:
        cluster_labels = clustering.fit_predict(X)
        silhouette_avg = silhouette_score(X, cluster_labels)
        print("For n_clusters =", k, "The average silhouette_score is :", silhouette_avg)
        silh.append(silhouette_avg)


# Count normalized K and distortion
```

```
norm_k = [((k-min(K))/(max(K)-min(K))) for k in K[::-1]]
norm_d = [((d-min(distortions))/(max(distortions)-min(distortions))) for d in distortions]

# Count difference between normalized K and distortion
res = [norm_k[i]-norm_d[i] for i in range(0,9)]
k = res.index(max(res))+1

print(f'\nOptimal number of clusters according to biggest different between normalized k and
corresponding distortion: {k}')


# Plot

fig, ax1 = plt.subplots()


color = 'tab:red'
ax1.set_xlabel('k')
ax1.set_ylabel('Distortion', color=color)
ax1.plot(K, distortions, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('diff', color=color)
ax2.plot(K, res, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()

plt.figure(figsize=(8,6))
plt.plot(range(1,9), silh)
plt.xlabel('The number of clusters (k value)')
plt.ylabel('Silhouette score')
plt.title('The silhouette scores graph')

plt.show()
```

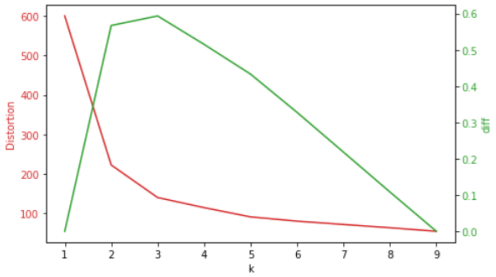
Otrzymane wartości współczynnika Silhouette:

```
Silhouette
For n_clusters = 2 The average silhouette_score is : 0.5817500491982808
For n_clusters = 3 The average silhouette_score is : 0.45994823920518646
For n_clusters = 4 The average silhouette_score is : 0.383850922475103
For n_clusters = 5 The average silhouette_score is : 0.3459012795948778
For n_clusters = 6 The average silhouette_score is : 0.33996630438958203
For n_clusters = 7 The average silhouette_score is : 0.32438404182471015
For n_clusters = 8 The average silhouette_score is : 0.340574718410144
For n_clusters = 9 The average silhouette_score is : 0.34301799509063646
```



Wyliczona wartość liczby klastrów:

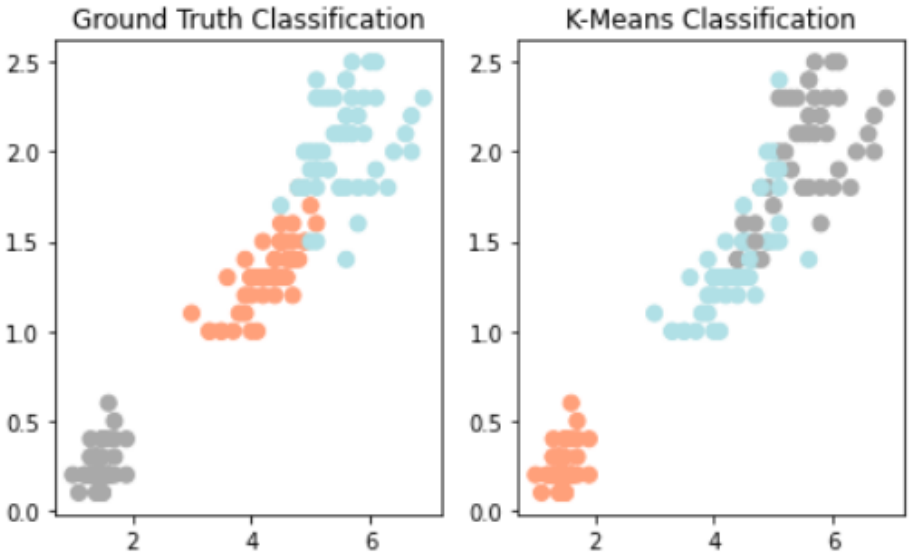
```
Optimal number of clusters according to biggest different between normalized k and corresponding distortion: 3
```



Fragment kodu odpowiedzialny za przeprowadzenie klasteryzacji na podstawie oblioczonej wartości k , wizualizację danych oraz porównanie wyniku klasteryzacji z danymi rzeczywistymi:

```
clustering = KMeans(n_clusters=k, random_state=5).fit(X)
df = iris.data
df.columns = [i.replace(' ', '_').replace('_(cm)', '') for i in iris.feature_names]


relabel = np.choose(clustering.labels_,[2,0,1]).astype(np.int64)
color_theme = np.array(['darkgray','lightsalmon','powderblue','red'])
plt.subplot(1,2,1)
plt.scatter(x=df.Petal_Length,y=df.Petal_Width,c=color_theme[iris.target], s=50)
plt.title('Ground Truth Classification')
plt.subplot(1,2,2)
plt.scatter(x=df.Petal_Length,y=df.Petal_Width,c=color_theme[clustering.labels_], s=50)
plt.title('K-Means Classification')

print(classification_report(Y, relabel))
```

Rozkład powstałych grup na wykresie:



Ocena poprawności otrzymanych wyników dla każdej z klas, poprzez porównanie z wartościami rzeczywistymi

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.74	0.78	0.76	50
2	0.77	0.72	0.74	50
accuracy			0.83	150
macro avg	0.83	0.83	0.83	150
weighted avg	0.83	0.83	0.83	150

Wartość współczynnika ***precision*** obliczana jest poniższym wzorem i oznacza jak wiele obiektów zostało przypisanych do klastra odpowiadającego klasie dla danych rzeczywistych

$$\frac{tp}{tp + fp}$$

gdzie:
tp –liczba wyników prawdziwie pozytywnych
fp – liczba wyników fałszywie pozytywnych

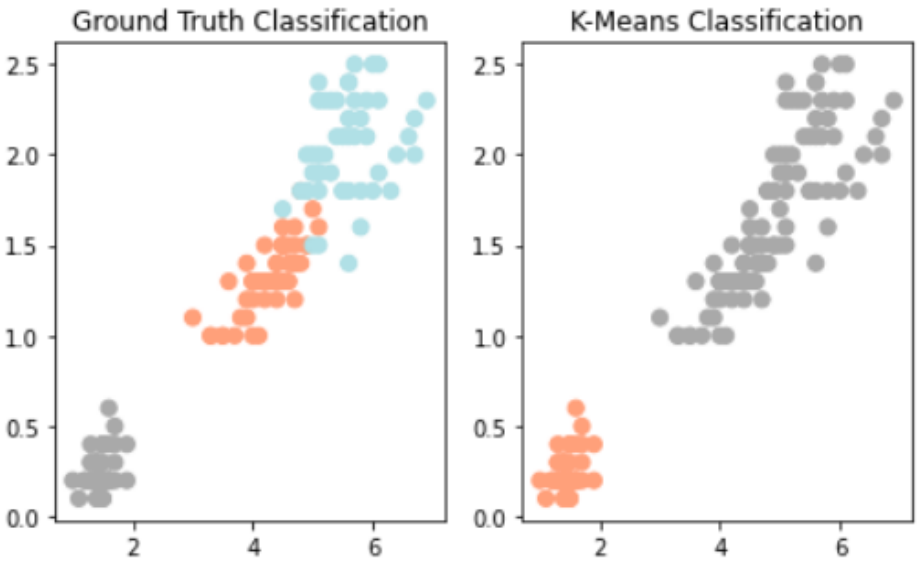
Wartość współczynnika ***recall*** należy interpretować jako ilość poprawnie dobranych obiektów klastru wśród danej klasy danych rzeczywistych, obliczana jest wzorem

$$\frac{tp}{tp + fn}$$

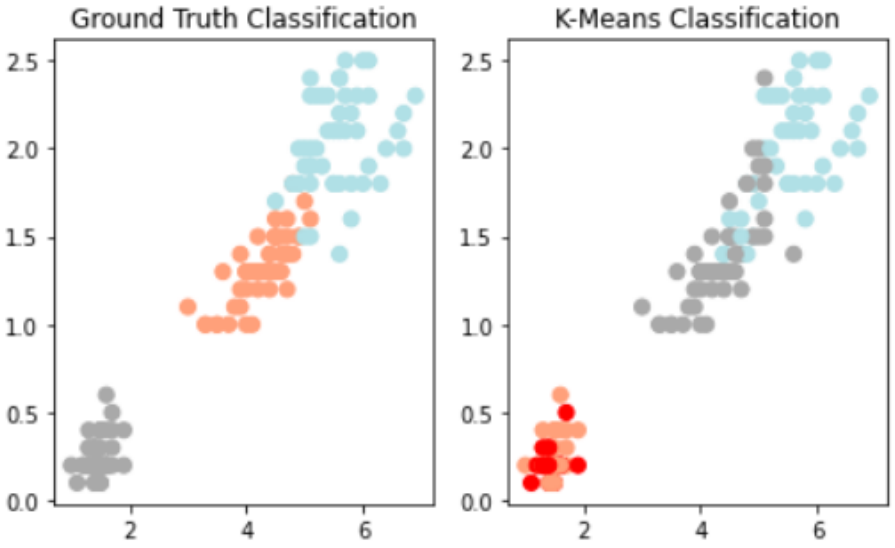
gdzie:
tp –liczba wyników prawdziwie pozytywnych
fn – liczba wyników fałszywie negatywnych

Wartość współczynnika ***f1-score*** może być interpretowana jako średnia harmoniczna ***precision*** i ***recall***, przy czym najlepszą wartość osiąga przy 1, a najgorszą przy 0.

Support oznacza liczbę wystąpiej każdej klasy dla wartości rzeczywistych.
Rozkład danych, przy założeniu istnienia dwóch klastrów



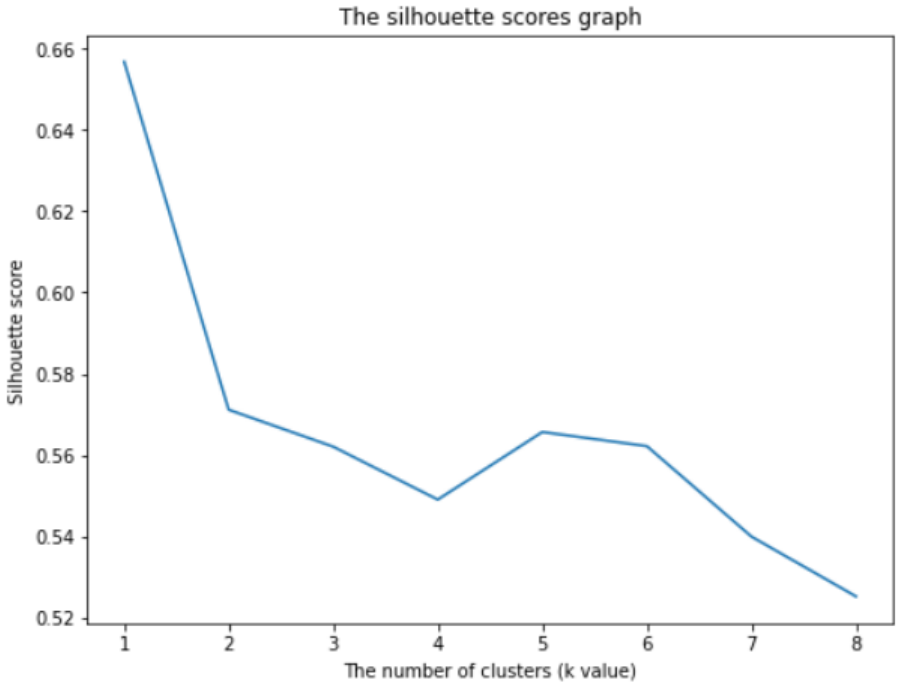
Rozkład danych, przy założeniu istnienia czterech klastrów



Wine Dataset

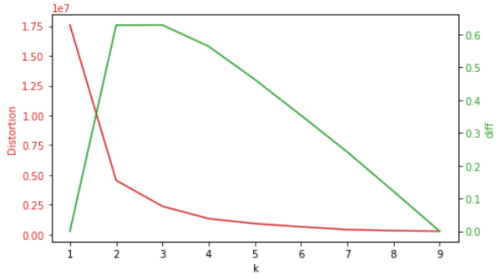
Otrzymane wartości współczynnika Silhouette:

```
Silhouette
For n_clusters = 2 The average silhouette_score is : 0.6568536504294317
For n_clusters = 3 The average silhouette_score is : 0.571138193786884
For n_clusters = 4 The average silhouette_score is : 0.5620323449580346
For n_clusters = 5 The average silhouette_score is : 0.5489993239795681
For n_clusters = 6 The average silhouette_score is : 0.5656413100386375
For n_clusters = 7 The average silhouette_score is : 0.5621677013863702
For n_clusters = 8 The average silhouette_score is : 0.5398971441034123
For n_clusters = 9 The average silhouette_score is : 0.525162492111064
```

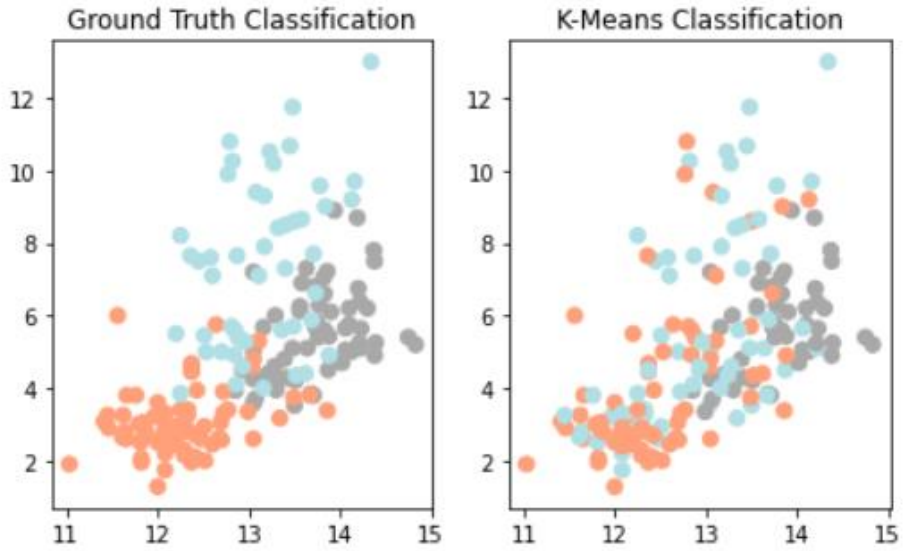


Wyliczona wartość liczby klastrów:

```
Optimal number of clusters according to biggest different between normalized k and corresponding distortion: 3
```



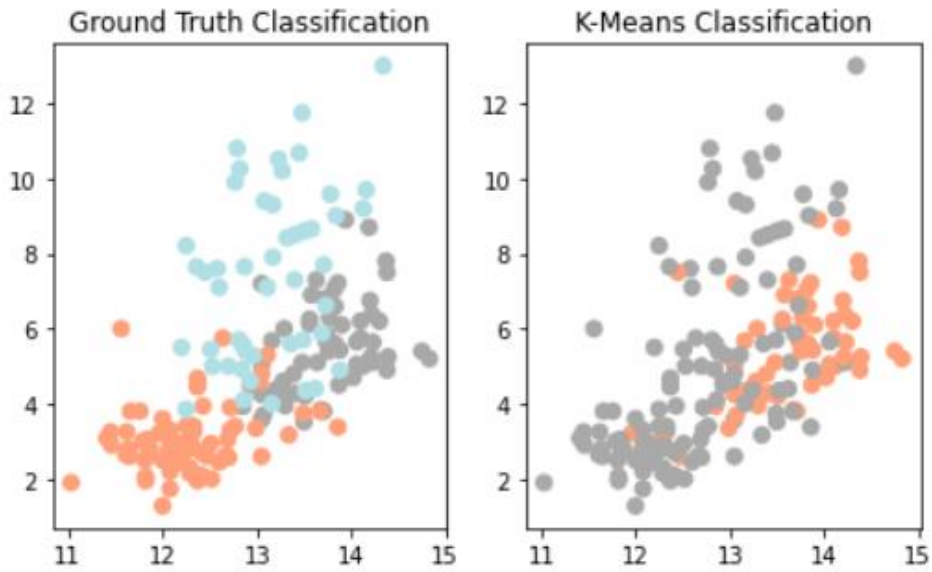
Rozkład powstałych grup na wykresie:



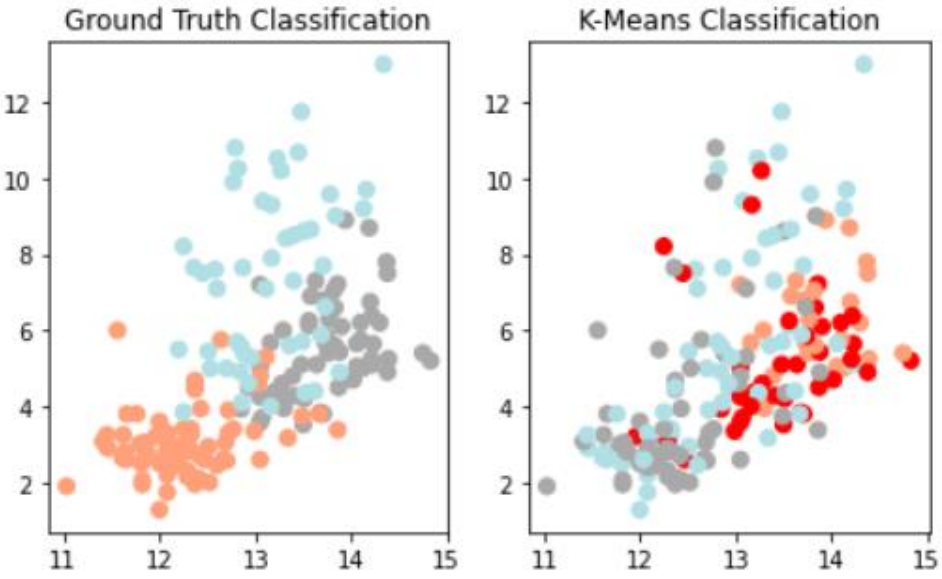
Ocena poprawności otrzymanych wyników dla każdej z klas, poprzez porównanie z wartościami rzeczywistymi

	precision	recall	f1-score	support
0	0.98	0.78	0.87	59
1	0.72	0.70	0.71	71
2	0.47	0.60	0.53	48
accuracy			0.70	178
macro avg	0.72	0.70	0.70	178
weighted avg	0.74	0.70	0.71	178

Rozkład danych, przy założeniu istnienia dwóch klastrów



Rozkład danych, przy założeniu istnienia czterech klastrów



DBSCAN

Celem porównania wyników z inną metodą klasteryzacji, zastosowano algorytm DBSCAN, stosując domyślne wartości, jakie funkcja przyjmuje w swojej implementacji dla biblioteki scikit-learn. W

kolejnych iteracjach wprowadzano zmiany dotyczące wartości epsilon oraz minimalnej liczbie próbek w zasięgu promienia sąsiedztwa, od której można uznać je za znalezioną grupę.

Wykorzystując domyślne wartości jakie przyjmuje funkcja DBSCAN dla biblioteki scikit-learn (epsilon = 0.5, minimalna liczba próbek = 5), otrzymano następujące wyniki dla zbioru danych Iris Dataset.

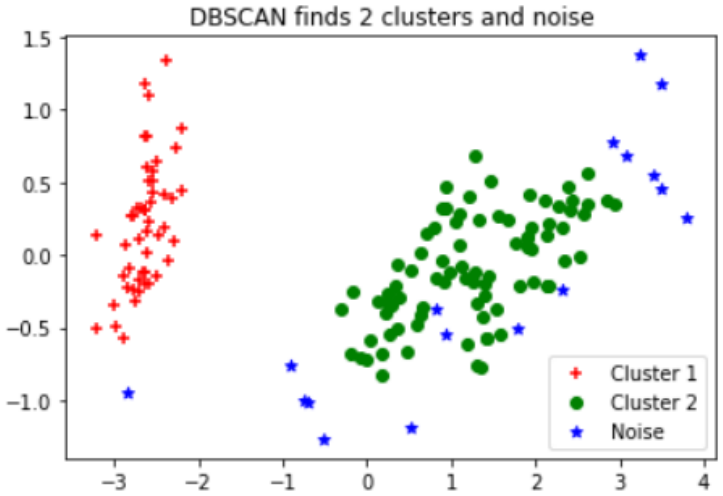
Fragment kodu odpowiedzialny za załadowanie danych i wywołanie metody DBSCAN dla domyślnych parametrów oraz wizualizację danych:

```
iris = load_iris(as_frame=True)
dbscan = DBSCAN().fit(iris.data)


core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True
labels = dbscan.labels_
labels_true = iris.target

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

pca = PCA(n_components=2).fit(iris.data)
pca_2d = pca.transform(iris.data)

for i in range(0, pca_2d.shape[0]):
    if dbscan.labels_[i] == 0:
        c1 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='r',
            marker='+')
    elif dbscan.labels_[i] == 1:
        c2 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='g',
            marker='o')
    elif dbscan.labels_[i] == -1:
        c3 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='b',
            marker='*')
plt.legend([c1, c2, c3], ['Cluster 1', 'Cluster 2',
       'Noise'])
plt.title(f'DBSCAN finds {n_clusters_} clusters and noise')
plt.show()
```

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
  0  0  1  1  1  1  1  1  1 -1  1  1 -1  1  1  1  1  1  1  1 -1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1  1  1 -1  1  1
  1  1 -1  1  1  1  1  1  1 -1 -1  1 -1 -1  1  1  1  1  1  1  1 -1 -1  1
  1  1 -1  1  1  1  1  1  1  1  1 -1  1  1 -1 -1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1]
```

Gdzie 0 i 1 oznaczają znalezione grupy, a –1 obiekty które nie pasują do żadnego z klastrów, nazywane szumem.

```
Estimated number of clusters: 2
Estimated number of noise points: 17
```



Dodatkowo zaimplementowano w kodzie pętlę szukającą takich ustawień epsilon i minimalnej liczby próbek, aby algorytm DBSCAN zwrócił trzy klastry I dodatkowym założeniem o minimalnej liczbie 30 punktów szumu. Fragment kodu odpowiedzialny za wywołanie pętli oraz obliczenie dokładności z jaką algorytm pogrupował dane:

```
for x in range(0, 10000):
    eps_ = uniform(0.1, 1.0)
    min_samples_  = randrange(1, 50)
    db = DBSCAN(eps=round(eps_, 2), metric='euclidean', min_samples=min_samples_).fit(iris.data)
    if 2 in list(db.labels_) and 3 not in list(db.labels_) and list(db.labels_).count(-1) <= 30 and
list(db.labels_).count(-1) >= 33:
        found =  f'Results for epsilon value {round(eps_, 2)} and min_samples {min_samples_}'
        labels = db.labels_
        core_sample_indices = db.core_sample_indices_
        comps = db.components_
zeros, ones, twos = [], [], []
```

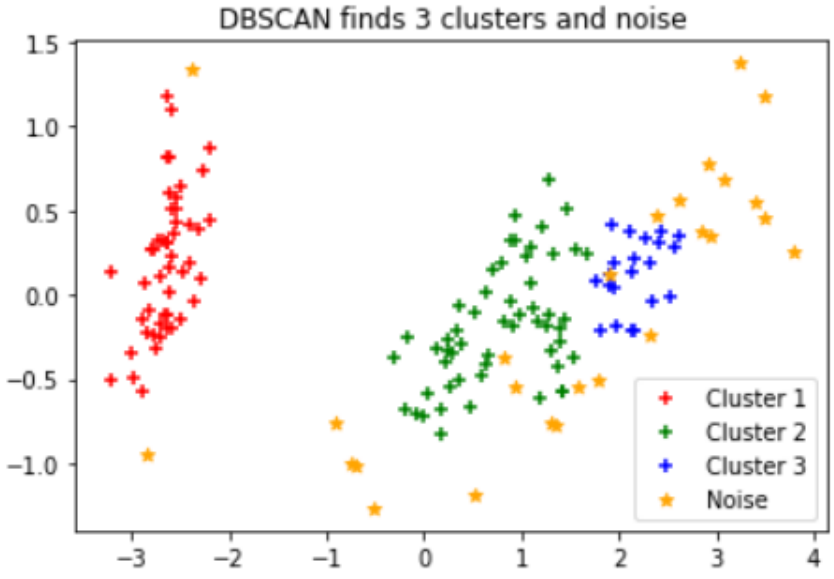
```
for i, v in enumerate(labels_true):
    if v == labels[i] and v == 0:
        zeros.append(v)
    elif v == labels[i] and v == 1:
        ones.append(v)
    elif v == labels[i] and v == 2:
        twos.append(v)
print(found)
print(f'DBSCAN precision for class 1: {len(zeros)/50}')
print(f'DBSCAN precision for class 2: {len(ones)/50}')
print(f'DBSCAN precision for class 3: {len(twos)/50}')

compare = [0 if v != labels[i] and labels[i] != -1 else 1 for i, v in enumerate(labels_true)]
```

Najlepsze otrzymane wartości dla DBSCAN :

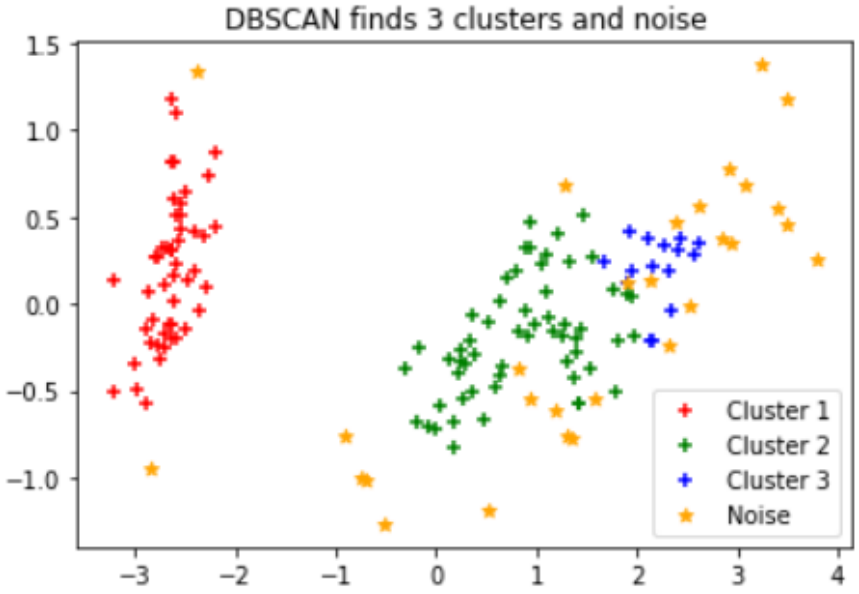
```
DBSCAN precision for class 1: 0.96
DBSCAN precision for class 2: 0.74
DBSCAN precision for class 3: 0.66
```

Niektóre z otrzymanych wyników:

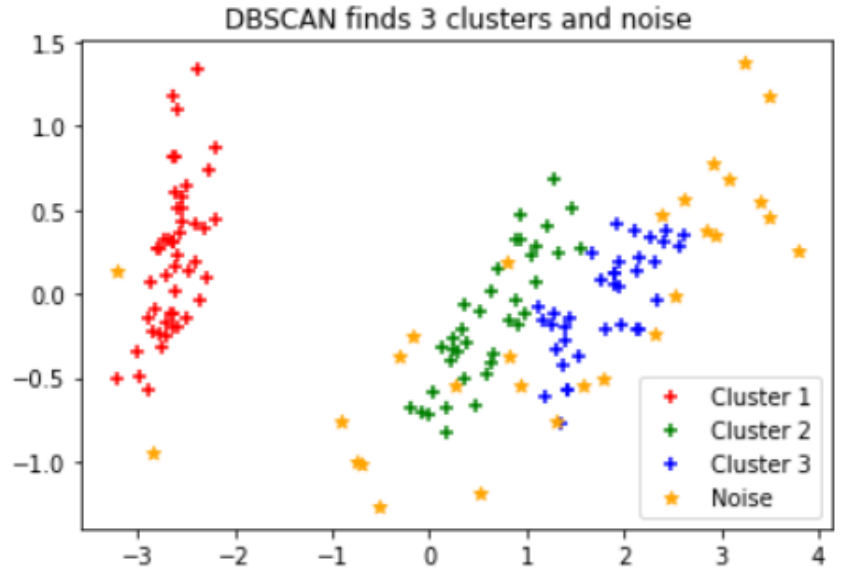
```
Results for epsilon value 0.53 and min_samples 14
Estimated number of clusters: 3
Estimated number of noise points: 26
```



```
Results for epsilon value 0.6 and min_samples 20
Estimated number of clusters: 3
Estimated number of noise points: 29
```




```
Results for epsilon value 0.42 and min_samples 5
Estimated number of clusters: 3
Estimated number of noise points: 29
```

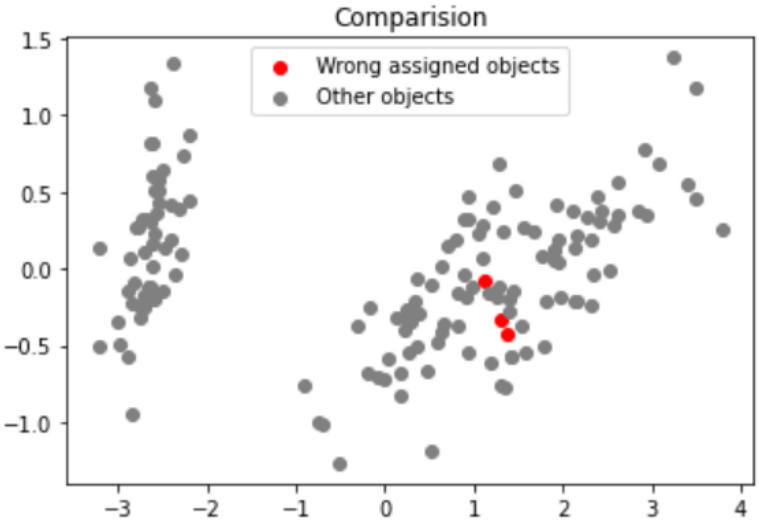


Fragment kodu odpowiedzialny za wyświetlenie wartości, które zostały przyporządkowane do innych klastrów w porównaniu do danych rzeczywistych

```
for i in range(0, pca_2d.shape[0]):
```

```
    p = pca_2d[i,0]
    q = pca_2d[i,1]
    if compare[i] == 0:
        b1 = plt.scatter(p,q,c='red', marker='o')
    elif compare[i] == 1:
        b2 = plt.scatter(p,q,c='gray', marker='o')

plt.legend([b1, b2], ['Class 1', 'Class 2'])
plt.title(f'Comparision')
plt.show()
```

Wizualizacja źle przyporządkowanych obiektów dla parametrów epsilon=0.42 i min_samples=5



4.3 Porównanie wyników

Jak widać, metoda kmeans w przypadku Iris Dataset pozwoliła uzyskać znacznie lepsze wyniki, które w dużym stopniu pokryły się z wartościami rzeczywistymi. W momencie zastosowania algorytmu DBSCAN, nie byliśmy w stanie otrzymać trzech klastrów przy niskiej liczbie punktów szumu.

Dalsze wykorzystanie metod w projekcie

Przedstawione w dokumencie metody, znajdą swoje zastosowanie w przyszłości, przy wyodrębnianiu grup spokrewnionych obiektów, dla danych opisujących pojedynczy przejazd platformy AGV.

Biblioteki python wykorzystane w projekcie

- **Scikit-learn**

- **KMeans**, wywołanie tej klasy zwróci jej obiekt o parametrach domyślnych lub zdefiniowanych przez użytkownika. Do istotnych parametrów możemy zaliczyć przede wszystkim liczbę klastrów. Metody w ramach klasy KMeans wykorzystane w projekcie:
  - fit(),
- **DBSCAN,** wywołanie tej klasy zwróci jej obiekt o parametrach domyślnych lub zdefiniowanych przez użytkownika. Parametry, które były modyfikowane w ramach projektu to wartość współczynnika espilon oraz min_samples. Wykorzystane metody:
  - fit()
- **load_iris**, loader zwracający zbiór danych dla Iris Dataset, jeśli przy jego wywołaniu przekażemy parametr as_frame jako True, zwrócone wartości przyjmą formę DataFrame biblioteki pandas.
- **load_wine**, identyczny loader, z tym, że zwraca zbiór danych dla Wine Dataset

- **Pandas**
  - **DataFrame,** podstawowa dwuwymiarowa struktura danych dla biblioteki Pandas o tabelarycznej strukturze danych z etykietowanymi wierszami i kolumnami.
- **Matplotlib,** biblioteka wykorzystana w celu wizualizacji danych

4.4. Predykcja

Pojęcie predykcji jest ściśle związane z przewidywaniem oraz wiąże się z takimi terminami jak regersja, klasyfikacja oraz uczenie nadzorowane. Sama regresja wykorzystywana jest do oszacowywania za pomocą jej technik wartości ciągłych lub uporządkowanych, natomiast klasyfikacja zajmuje się problematyką predykcji dyskretnych etykiet klas. W przeciwieństwie jednak do problemu klasyfikacji, który także związany jest z uczeniem maszynowym, w problemie predykcji najistotniejszą rolę gra wcześniej wspomniane szacowanie parametrów strukturalnych modelu na podstawie wyników próby. Mówiąc jaśniej: klasyfikacja działa w oparciu o predykcję etykiet (labels), a regresja działa w oparciu o predykcję ilości. Poniżej postanowiliśmy zamieścić tabelę, która w dosyć jasny sposób opisuje kilka wybranych metod, które wiążą się z realizacją projektu, a ściślej rzecz ujmując wyborem optymalnego rozwiązania dla naszego problemu.

Metoda	Typ	Zmienna niezależna	Założenia
Regresja liniowa	Regresja	Wszystkie ilościowe	Liniowa zależność, łatwa w implementacji
Naiwny klasyfikator Bayesa	Klasyfikacja	Tylko nominale (jakościowe)	Wymaga dużych zbiorów danych
k-NN	Regresja lub klasyfikacja	Wszystkie ilościowe	Nieliniowe zależności, odchylenia w danych

Sieci neuronowe	Regresja lub klasyfikacja	Wszystkie ilościowe	Model czarnej skrzynki

Tabela 1: Wybrane metody stosowane w uczeniu nadzorowanym

Tutaj pozwolimy sobie opisać jeden z najistotniejszych terminów zawartych w tabeli, którym będzie zmienna niezależna. Z definicji jest nazywana ona predyktorem. Podobnie jak zmienne zależne (decyzyjne) są zmiennymi opisującymi obiekty. Głównym założeniem modelu regresji jest ustanowienie, że zmienne niezależne 'x' nie są ze sobą silnie skorelowane, żadna ze zmiennych niezależnych nie powinna być kombinacją liniową innych zmiennych niezależnych, liczba obserwacji musi być większa od liczby parametrów do oszacowania, a także zakłada się istnienie modelu liniowego względem parametrów. (Stefanowski, 2009)

W naszym przypadku do predykcji wykorzystane zostaną metody k-NN (k Neareast Neighbours) oraz regersji liniowej. Nasze rozważania zostaną oparte o zbiory danych, które są dobrze znane w przypadku tematyki machine learningu.. Wiedza nabyta w trakcie tej analizy i przedstawiona tutaj w raporcie posłuży nam jako podstawa do pracy nad danymi otrzymywanymi z AGV w przyszłym semestrze.

Pojęcia podstawowe związane z predykcją

W tym podrozdziale opisane zostaną m.in. miary jakości, czym jest model, zbiór testowy, zbiór treningowy, itd. Jest to niezbędne w celu zrozumienia dalszych działań, w których wykorzystana zostanie przedstawiona tu wiedza w praktyce. Na samym początku należy krótko przedstawić czym jest Machine Learning, w dużym uproszczeniu jest to nauka na podstawie danych. Mówiąc jaśniej nie jest niezbędne pisanie dedykowanego programu w celu poradzenia sobie z pewnym problemem, zamiast niego stosuje się algorytmy, które otrzymują zebrane przez nas dane, budując tym samym własną logikę w oparciu o te dane. Istotne jest m.in. wydzielane cech, których algorytm rozpoznający lub regresji będzie się uczył w celu odkrywania interesującej programistę informacji.

Miary jakości

Opis pojęć rozpoczniemy od wytłumaczenia miar jakości stosowanych w ML. Poniżej znajdować będzie się opis tych miar, wraz z ich wytłumaczeniem. Na wstępie należy też zaznaczyć, że ewaluacja na podstawie miar jakości jest dla obydwu predykcji różna, na przykład: predykcje w oparciu o klasyfikację mogą być ewaluowane w oparciu o dokładność (accuracy), w przypadku regresji jest to niemożliwe, bowiem dla niej stosuje się m.in. wyliczenie błędu średniokwadratowego, którego natomiast nie stosuje się dla klasyfikacji.

Zatem dla problemu regresji zastosujemy następującą miarę, znaną dalej jako RMSE (Root mean squared error), którą zapisuje się następującym wzorem:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2}$$

,gdzie O_i oznaczane są obserwacje, S_i oznaczane są dane przewidywane, a „n” oznacza liczbę obserwacji dostępnych do analizy.

Poniżej zawarty też jest przykładowy kod napisany w języku Python wykorzystujący powyższy opis algebraiczny, dla przypadku w którym model predykcyjny dokonał dwóch predykcji, jedna wyniosła 1.5, gdzie wartością oczekiwaną jest 1, a inna wyniosła 3.3, gdzie oczekiwaną wartością jest 3.0:

```
RMSE = sqrt(average(error^2))

RMSE = sqrt(((1.0 - 1.5)^2 + (3.0 - 3.3)^2) / 2)

RMSE = sqrt((0.25 + 0.09) / 2)

RMSE = sqrt(0.17)

RMSE = 0.412
```

W przypadku klasyfikacji wyróżniamy następujące miary jakości (Fujarewicz, 2019):

	Decyzja klasyfikatora	
Wiedza eksperta	P_{klas}	N_{klas}
P_{exp}	TP	FN
N_{exp}	FP	TN

Błąd klasyfikatora	$Err = \frac{FP+FN}{TP+TN+FP+FN}$
Dokładność (precyzja) klasyfikatora	$Acc = \frac{TP+TN}{TP+TN+FP+FN} = 1 - Err$
Czułość klasyfikatora	$Sens = \frac{TP}{TP+FN}$
Specyficzność klasyfikatora	$Spec = \frac{TN}{TN+FP}$
Dodatnia wartość predykcyjna	$PPV = \frac{TP}{TP+FP}$
Ujemna wartość predykcyjna	$NPV = \frac{TN}{TN+FN}$

Głównym i najczęściej analizowanym w przypadku klasyfikacji estymatorem jest dokładność (accuracy). Definiowany jest jak iloraz sumy próbek prawdziwie pozytywnych oraz prawdziwie negatywnych dzielonych przez sumę próbek prawdziwie pozytywnych, prawdziwie negatywnych, fałszywie

pozytywnych oraz fałszywie negatywnych, zapisywanym inaczej jako różnica jedynki i błędu klasyfikatora.

Typy uczenia maszynowego

Uczenie maszynowe, którego jedną z gałęzi jest właśnie predykcja składa się z trzech podstawowych elementów, a są to:

1. Uczenie nadzorowane (supervised) – W tym podejściu posiadamy poetykietowane dane, model jest w stanie się wyuczyć tych etykiet i dokonać tym samym predykcji lub klasyfikacji.
2. Uczenie nienadzorowane (unsupervised) – Dla tego podejścia dane nie są poetykietowane, tzn. nie wiemy w jaki sposób dane opisują wybrane cechy naszego obiektu. Z tym przypadkiem ściśle związana jest klasteryzacja lub grupowanie, gdzie tworzone są grupy tych samych typów obiektów.
3. Uczenie ze wzmocnieniem (reinforcement) – W tym podejściu nauka przebiega na uczeniu się, na podstawie środowiska poprzez interakcję z nim i otrzymywanie nagród dla przeprowadzonych działań. Mówiąc jaśniej, stara się transformować do wybranego stanu poprzez wykonanie wybranej akcji, a nagrodę otrzymuje za działania pozytywne lub negatywne dla każdego ze swoich interakcji.

Terminy podstawowe przy budowie algorytmów

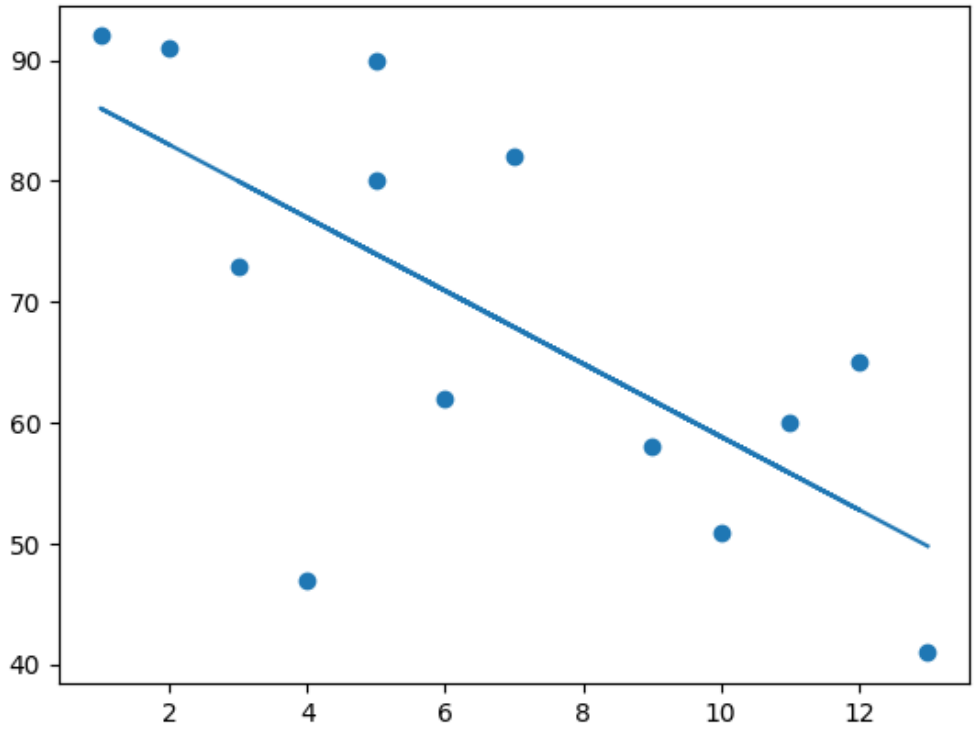
W tym rozdziale omówione zostaną podstawowe pojęcia, które będą używane w dalszej części raportu.

1. Model – Tworzony jest w trakcie procesu treningu, działa na zasadzie dostarczenia danych treningowych do algorytmów uczenia maszynowego, na jej podstawie algorytm dokonuje nauki i w ten sposób uzyskiwany jest wytrenowany model, który potem służy do analizy lub badań.
2. Zbiór danych treningowych oraz testowych – Dane doprowadzane do algorytmu uczącego nazywanego są zbiorami danych treningowych. Predykcja natomiast dokonywana jest na odseparowanym zbiorze danych zwanym dalej zbiorem testowym (najczęstszymi proporcjami jest wykorzystanie 90% danych w celach treningowych, a pozostałe 10% w celach testowych). Na tym ostatnim zbiorze dokonuje się analizy jakości wyszkolonego modelu.
3. Przeładowanie (overfitting) i niedopasowanie (underfitting) danych – Model nazywany jest przeładownym, gdy nauczanie zbiorów treningowych jest wysoce efektywne (np. accuracy równe 99%), ale niemożliwe jest na jego podstawie dokonanie generalizacji, czyli nie dokonuje dobrej predykcji, a co za tym idzie nie wykrywa, bądź niedostatecznie wykrywa dane testowe. O niedopasowaniu mówimy w momencie, gdy niemożliwe jest samodzielne wyuczenie modelu i tym samym zastosowanie go dalej do predykcji, oznacza to że taki model nie zadziała i nie dokona poprawnej predykcji.
4. Ścięcie (bias) oraz wariancja (variance) – Są to błędy, które mogą wystąpić w procesie uczenia. Pierwszy z nich występuje w przypadku nieprawidłowych założeń, a jego wynikiem jest niedopasowany model. Drugi błąd ma miejsce, gdy dojdzie do zjawiska przeładowania. Wiąże się to z zbyt dużą wrażliwością

modelu na małą zmienność w zbiorze testowym. Błędy te są od siebie ściśle zależne, a to oznacza że redukcja jednego wiąże się z zwiększeniem drugiego.

Opis algorytmu regresji liniowej

Jest to typ modelu regresji oraz algorytmu uczenia nadzorowanego, będąc jednym podstawowych modeli uczenia maszynowego. Najpierw odpowiemy na czym polega regresja. Jest to metoda modelowania wybranej wartości na podstawie niezależnych predyktorów. Jest używana głównie do przewidywania i odnajdywania powiązań między różnymi zmiennymi. Techniki te różnią się na podstawie liczby zmiennych niezależnych i typów powiązań miedzy zmiennymi niezależnymi oraz zależnymi. Poniżej przedstawienie prostego wykresu regresji liniowej uzyskanego przy pomocy funkcji stats.linregress(x,y) dla zbiorów danych x oraz y znajdującej się w bibliotece scipy



Rysunek 16 Przykładowy wykres regresji liniowej

Linia, która przechodzi przez środek, jest prostą najlepszego dopasowania do naszego zbioru danych. Dzięki niej widać właśnie omawianą wcześniej relację pomiędzy zmiennymi niezależnymi na osi poziomej z zmiennymi zależnymi na osi pionowej. Prosta taka jest definiowana nie inaczej jak $y(x) = a_1 * x + a_0$, czyli jako zwykła funkcja liniowa. Celem algorytmu regresji liniowej jest znalezienie optymalnych wartości a_1 oraz a_0.

Przy wyznaczaniu tych parametrów niezwykle istotna jest funkcja kosztu. Pozwala nam opracować i ustalić najlepsze możliwe wartości parametrów funkcji liniowej na podstawie punktów odpowiadających za dane w naszym zestawie. Poszukiwanie to jest problemem minimalizacji, gdzie konieczne jest zminimalizowanie błędu między wartością przewidywaną i wartością rzeczywistą. Zapis funkcji kosztu jest w tym wypadku następujący:

$$J = \frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2$$

Wzór ten był omawiany przy okazji opisywania miar jakości, jednak tam był on pierwiastkowany dając nam błąd średniokwadratowy.

Kolejnym koniecznym do omówienia terminem w przypadku regresji jest opisanie gradientu prostego, a mówiąć ściślej metody gradientu prostego. Jest definiowana jako algorytm numeryczny mający na celu znalezienie minimum zadanej funkcji celu. (Neursoft, 2021)

Jej zadaniem jest aktualizowanie parametrów funkcji liniowej w taki sposób, żeby zredukować funkcję kosztu. Działa na zasadzie rozpoczęcia z pewnymi wartościami parametrów a_1 oraz a_0, a następnie zmienianiem tych wartości iteracyjnie w celu zredukowania kosztu. Upraszczając: pomaga on przy optymalizacji tych wartości. Należy również wspomnieć, że opisywane parametry w realizacji praktycznej odpowiadają za współczynniki w regresji liniowej, natomiast w sieciach nueronowych są one nazywane wagami. Dodać powinno się, że czasami funkcja kosztu może nie być funkcją wypukłą i koniecznie w takiej sytuacji jest rozstrzygnięcie minimum lokalnego, jednak dla regresji liniowej taka sytuacja nie występuje.

Poniżej zamieszczamy opis algebraiczny, który wyjaśnia działanie tej metody.

Przy założeniu, że $S_i = a_0 + a_1 x_i$, otrzymamy następującą postać funkcji kosztu:

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 x_i - O_i)^2$$

Metoda ta opiera się na rozpatrywaniu pochodnych cząstkowych, które definuje się właśnie jako gradienty aktualizujące parametry funkcji liniowej. Poniżej dokonane zostało różniczkowanie powyższej funkcji po jednym i drugim parametrze:

$$\frac{\delta J}{\delta a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 x_i - O_i)$$

$$\frac{\delta J}{\delta a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 x_i - O_i)\cdot x_i$$

Wykorzystując powyższe pochodne cząstkowe wprowadzony zostaje hiperparametr (czyli parametr podawany przez użytkownika, w przeciwieństwie do „parametru", który jest samodzielnie wyliczany przez algorytm podczas uczenia, czyli na przykład wspomniane wcześniej współczynniki w regresji liniowej). Ten hiperparametr oznaczony zostanie jako α, a zdefiniowany jako współczynnik uczenia (learning rate). Mniejszy pozwoli precyzyjniej zbliżyć się do minimum, niestety dotarcie do niego wymaga więcej czasu. Większy natomiast wcześniej dotrze do minimum, ale niestety istnieje ryzyko przeskoczenia wartości minimalnej. To tłumaczy jak bardzo istotne jest dobranie odpowiedniego parametru α przez programistę. Poniżej wzory, które opisują wykorzystanie współczynnika uczenia:

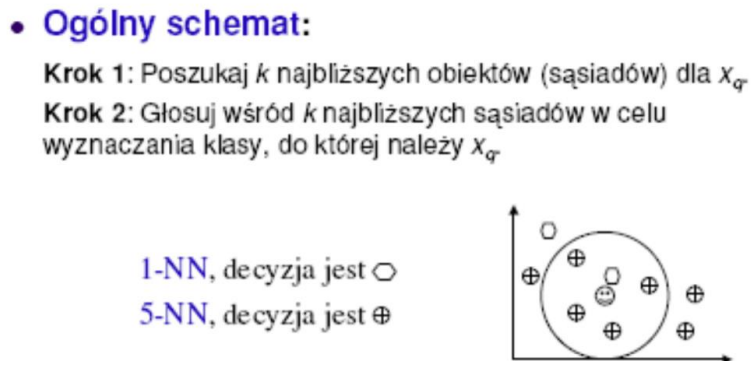
$$a_0 = a_0 - \alpha \cdot \frac{2}{n} \sum_{i=1}^{n} (a_0 + a_1 x_i - O_i)$$

$$a_1 = a_1 - \alpha \cdot \frac{2}{n} \sum_{i=1}^{n} (a_0 + a_1 x_i - O_i) \cdot x_i$$

Implementacja algorytmu regresji liniowej na wybranych zbiorach danych zostanie zaprezentowana w dalszej części raportu.

Opis algorytmu k-NN

Opisywany w tym podrozdziale algorytm, podobnie jak poprzedni jest przykładem rozwiązania dla problemu uczenia nadzorowanego. Tak jak we wstępie zostało wspomniane może on zostać użyty zarówno do problemów regresji, jak i klasyfikacji. Zakłada on istnienie tych samych zmiennych lub innych danych (inaczej nazywanych feature similarity) w bliskim sąsiedztwie, czyli istnienia takich samych obiektów obok siebie. Opis ten najłatwiej jest przedstawić graficznie.



Rysunek 17 Ogólny schemat k-NN

Analizując powyższy schemat oraz kroki algorytmu nietrudno dostrzec, że dla ikony buzi, gdy założymy klasyfikację 1-NN, czyli 1 obiektu w najbliższym sąsiedztwie to zakwalifikowane zostanie kółko. W przypadku 5-NN, powyższa klasyfikacja wybierze krzyżyk, z tego powodu że wykryje ich więcej w zaznaczonym przez okręg obszarze. Dystans do poszczególnych wartości wyliczany jest na podstawie dystansu nazywanego w literaturze euklidesowym. Znając położenie punktów w układzie kartezjańskim,

jesteśmy w stanie wyliczyć dystans, który jest definiowany jako pierwiastek sumy różnic kwadratowych pomiędzy nowym punktem x, a istniejącym punktem y. Jest to najprostszy, a zarazem jeden z najskuteczniejszych sposobów, który jest efektywnie wykorzystywany w opisywanym algorytmie. Innym sposobem jest wykorzystanie funkcji dystansu Manhattan, definiowanej jako wartość bezwzględna wektorów, będących sumą różnić bezwzględnych. Wzory te zostały zamieszczone poniżej.

$$Euclidean = \sqrt{\sum_{i=1}^{k} (x¿¿i - y_i)^2 ¿}$$

$$Manhattan = \sum_{i=1}^{k} ¿x_i - y_i \vee ¿¿$$

Najistotniejsze wydaje się określenie czynnika „k", a mówią precyzyjniej jego wartości optymalnej, który będzie determinował ilość sąsiadów, kiedy przypiszemy wartość do każdej nowej obserwacji. Zanim jednak opiszemy sposób dobrania wartosci „k" należy opisać kroki algorytmu, który składa się kilku etapów, jakie należy wykonać, aby móc zacząć z niego efektywnie korzystać. Opisany on został przez nas w punktach poniżej:

A. Załadowanie zbioru danych,
B. Inicjalizacja „k" dla wybranej liczby sąsiadów,
C. Dla każdego przykładu zawartego w danych:
a. Należy obliczyć dystans między nowym punktem, a istniejącym punktem w zbiorze danych,
b. Dodać dystans oraz indeks nowego punktu do uporządkowanego zbioru,
D. Dokonać sortowania zbioru dystansów i indeksów rosnąco na podstawie dystansów,
E. Pobrać pierwsze „k" wejść z posortowanego zbioru,
F. Uzyskać etykiety wybranych „k" wejść,
G. Dla problemu regresji, zwrócić średnią z „k" etykiet.

Nadal pozostaje jednak nierozwiązana sprawa doboru czynnika „k". Najprostszym sposobem jest wykorzystanie metodyki „prób i błędów" i obserwacja na podstawie kilku dobieranych wartości, która z nich zwróci najmniejszą liczbę błędów przy utrzymaniu zdolności algorytmu do możliwie najbardziej precyzyjnej predykcji, w momencie gdy otrzyma on dane których nigdy wcześniej nie zarejestrował. Przy tego typu metodzie jednak należy brać pod uwagę kilka aspektów, których nie powinno się pomijać, tj.: przy zmniejszaniu wartości czynnika „k", predykcje stają się co raz mniej precyzyjne. W przypadku odwrotnym, gdy czynnik będzie nadmiernie zwiększany wyniki staną się precyzyjniejsze (oczywiście do pewnego limitu), natomiast problemem może stać się w tej sytuacji rosnąca z tego tytułu liczba błędów. Oznacza to zbyt dużą wartość nadaną parametrowi „k".

Algorytm jest łatwy w implementacji i prosty w zrozumieniu z teoretycznego punktu widzenia, nie występuje w nim potrzeba budowy i strojenia różnych parametrów czy tworzenia dodatkowych założeń. Co więcej, jak zostało powiedziane jest on algorytmem uniwersalnym, tzn. można go z powodzeniem zastosować zarówno dla problemu klasyfikacji, jak i regresji. Niestety jego znaczącą wadą jest powolność w przypadku, gdy liczba przykładów lub predyktorów (zmiennych niezależnych) znacząco wzrasta co wymaga dużej mocy obliczeniowej urządzenia, na którym odbywa się predykcja.

Predykcja zbioru „Iris"

Rozdział ten będzie opierał się o analizę i implementację powyższych algorytmów predykcyjnych dla zbioru danych iris. Jako, że korzystamy podczas pracy z języka programowania Python, wymagane jest, aby zainstalować odpowiednie biblioteki, które pozwolą nam rozpocząć pracę z ww. zbiorem. W tym celu stworzone zostało wirtualne środowisko do którego zostały pobrane moduły pandas (do analizy danych), numpy (tworzenie tablic oraz macierzy), scikit-learn (zbiór iris), sklearn (do analizy problemu regresji), matplotlib (do wyświetlenia wykresów), seaborn (wizualizacja w oparciu o matplolib).

Zbiór danych iris składa się z trzech gatunków tego kwiatu. Odmiany te noszą nazwy Setosa, Versicolor oraz Virginica. Każda z odmian zawiera 50 wierszy, gdzie podane są długość oraz szerokość płatka kwiatu oraz długość i szerokość płatków kielicha. Na podstawie tych pogrupowanych danych dokonana zostanie predykcja z wykorzystaniem dwóch poniższych metod.

Jeden i drugi algorytm był przez nas analizowany w zeszycie Jupytera. Pierwszym co wykonaliśmy to stworzyliśmy klasę, która pozwoliła nam zarówno na załadowanie pliku danych zawartych we zbiorze, jak i stworzenie dwóch ramek danych. Pierwsza ramka danych przyjmuje do parametru data wszelkie dane związane z irisem dzieląc go przy tym na kolumny w oparciu o nazwę cech. Druga natomiast przyjmuje target i tworzy na jego podstawie 3 klasy, każda dla jednego z wybranych kwiatów.

```
# Załadowanie zbioru danych iris
iris = load_iris()
# Stworzenie ramek danych
iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
target_df = pd.DataFrame(data= iris.target, columns= ['species'])


def applier(iris_type):
    if iris_type == 0:
        return 'setosa'
    elif iris_type == 1:
```

```
        return 'versicolor'
    else:
        return 'virginica'


target_df['species'] = target_df['species'].apply(applier)


# łączenie ramek danych
iris_df = pd.concat([iris_df, target_df], axis= 1)
```

Metoda applier, która przyjmuje jako parametr typ irysa, konwertuje klasy na odpowiadające im nazwy kwiatów. W ten sposób, gdy dla target_df[‘species’] wywołamy powyższą metodę zamienimy numery klas na odpowiadające im nazwy. Na samym końcu połączyliśmy ramki danych w jedną iris_df, która zawiera zarówno dane z iris_df, jak i z target_df. Na tej klasie oprzemy resztę naszych rozważań.

Kolejnym co zrealizowaliśmy było wywołanie funkcji describe(), która to pozwoliła nam podejrzeć co znajduje się naszej ramce:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Rysunek 18 Zawartość ramki danych iris

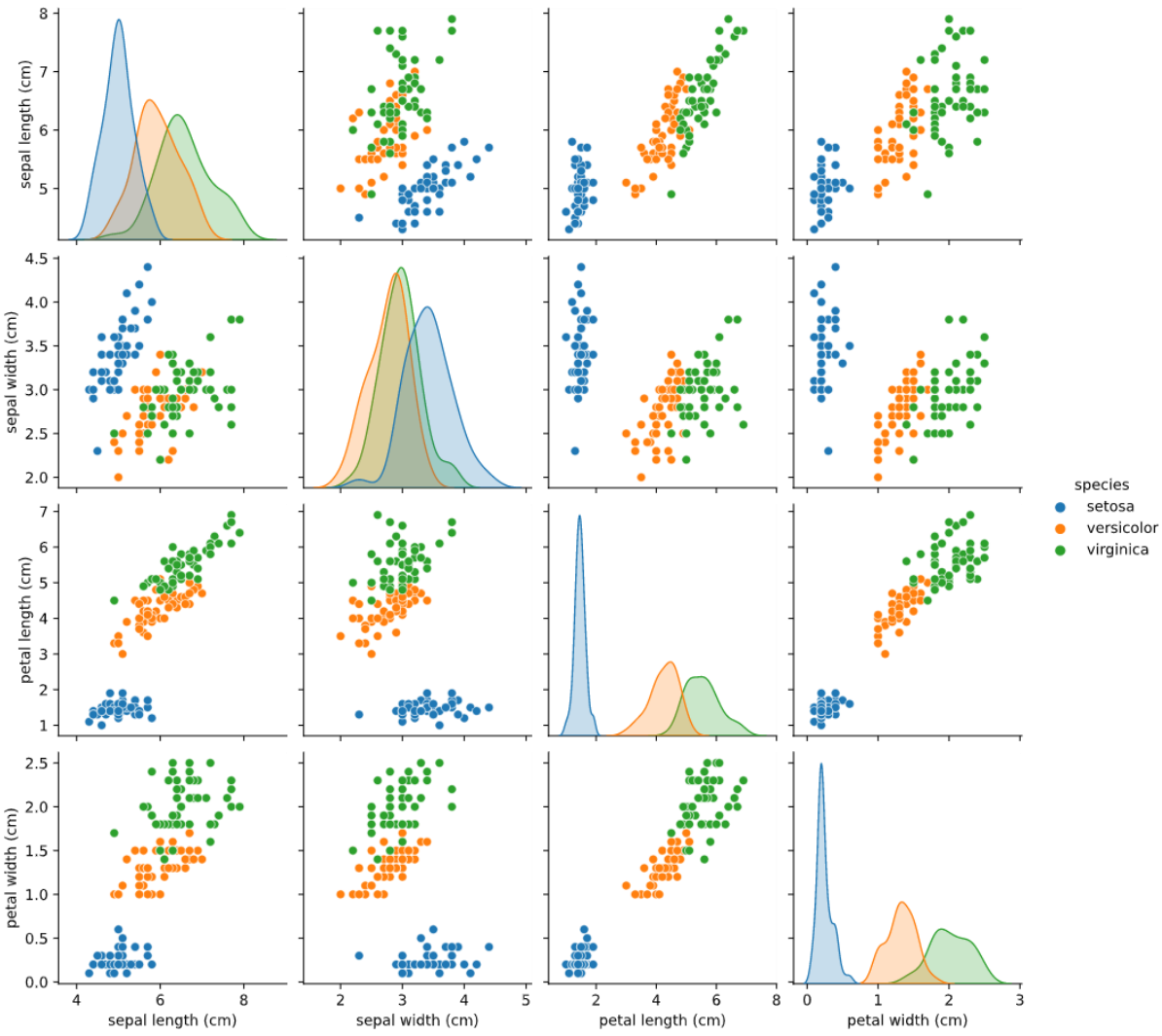
Powyższe parametry dostarczają nam wielu bezcennych informacji statystycznych. Otrzymujemy m.in. ilość danych dla każdej kolumny, czyli dla każdego elementu kwiatu. Znajdują się tam m.in. średnia, odchylenie standardowe, wartość maksymalna, itd. Tutaj możemy stwierdzić, że przy analizie danych otrzymywanych z AGV, chcielibyśmy przy pomocy klasteryzacji uzyskać zbliżony efekt, tj. posiadać taką ramkę danych, z takimi jak powyżej parametrami, które pozwolą nam dokonać efektywnej predykcji.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Rysunek 19 Informacje o ramce danych

Wywołanie funkcji info() dostarcza nam zwięzłych informacji o posiadanych przez nas danych. Jak widzimy dostajemy głównie dane numeryczne w 64-bitowym formacie zmiennoprzcinkowym. Zgodnie z tym co rozpatrywaliśmy w założeniach teoretycznych predykcja przy pomocy regresji liniowej jest zalecana do tego typu problemów. Podobnie będzie się to miało w sprawie pracy z AGV, tam również przewidywane jest otrzymanie zmiennych numerycznych, w podobnej postaci które będą nas informować o pewnych zachowaniach pojazdu. Species zostało zakwalifkowane jako obiekt, albowiem tam przechowywane są dane o gatunku irisa.

Przy pomocy bibliotek seaborn wygenerowaliśmy poniższe wykresy dla każdej z możliwości, grupując je na podstawie reprezentowanych przez dany kwiat cech. Wykres dostarczył nam użytecznych informacji, na jego podstawie jesteśmy w stanie w przejrzysty sposób określić zależność jednej cechy od drugiej, natomiast gdy dwie te same cechy spotykają się, jesteśmy w stanie ustalić, dla którego gatunku tych cech będzie najwięcej. Mając tą wiedzę, jesteśmy w stanie zająć się problemem predykcji.



Rysunek 20 Wykresy iris wraz z wizualizacją poszczególnych parametrów

Algorytm regresji liniowej (Iris)

W celu rozpoczęcia pracy dokonaliśmy konwersji obiektów na numeryczny typ danych, czyli zgodnie z teorią zrobiliśmy to czego wymaga problem regresji. W ten sposób w naszej ramce znajdują się tylko i wyłącznie dane numeryczne. Uzyskaliśmy to po dokonaniu „dropu", obiektu species z iris_df. Później utworzyliśmy zmienne, określając który element zostanie poddany predykcji. Wybór padł na „sepal length (cm)".

```
# Zmienne, gdzie x będą danymi treningowymi, a y danymi testowymi

X= iris_df.drop(labels= 'sepal length (cm)', axis= 1)

y= iris_df['sepal length (cm)']
```

Następnie podzieliliśmy zbiór danych określając dokładnie co jest danymi treningowymi, a co testowymi.

```
# Podzielenie zbioru danych

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33, random_state= 101)
```

Korzystając z modelu regresji liniowej postanowiliśmy dokonać predykcji otrzymując przy tym błędy, które są naszą główną miarą jakości.

```
Mean Absolute Error: 0.2595570975563035

Mean Squared Error: 0.10174529564238954

Mean Root Squared Error: 0.3189753840696638
```

Traktujemy je jako rezultat dopasowania naszego modelu. Te same miary jakości będziemy chcieli koniecznie uzyskać dla AGV, ponieważ tylko i wyłącznie na ich podstawie jesteśmy w stanie określić jakiej jakości model udało nam się uzyskać. Zgodnie z odczytanmi miarami nasz model ma potencjał do efektywnego wyuczenia się w oparciu o odstarczony zbiór danych.

Ostatnie co nam pozostało to przetestować nasz zaimplementowany model, wiedząc że aktualna długość płatka wynosi 4.6 cm ostatecznie otrzymaliśmy przewidywaną wartość, która wyniosła 5.46 cm.

```
Predicted Sepal Length (cm): 5.461145872156033

Actual Sepal Length (cm): 4.6
```

Jest to wartość dosyć spora (prawie trzykrotnie większa od mean absolute error) i jeżeli taka sytuacja będzie miała w przypadku analizy danych z AGV, istotne będzie poważne przyjrzenie się wynikom otrzymanym metodą regresji liniowej.

Algorytm k-NN (Iris)

W tym podrozdziale zajmiemy się klasyfikacją w oparciu o k-NN. Nieco zmienimy podejście, ponieważ zaczniemy opierać się na klasach. To pozwoli nam ocenić, która z metod predykcji zaproponowanych dla tego datasetu jest lepsza i efektywniejsza.

```
# podzielenie danych na zbiór treningowy i testowy, random state ustawiony nisko ponieważ dla analogic
znego jak w regresji daje 100% pewność

X_train, X_test, y_train, y_test = train_test_split(iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal leng
th (cm)', 'petal width (cm)']],iris_df['species'], random_state=0)
```

Ponownie dokonaliśmy podziału na zbiór testowy oraz treningowy, tym razem jednak nie wykonywaliśmy dropu wartości, wobec której chcieliśmy zastosować predykcję. Random state został ustawiony na 0, ponieważ w przypadku analogicznym jak w regresji otrzymywaliśmy 100% pewność w klasyfikacji, co jest efektem nieporządanym.

Podczas pracy do klasyfiaktora k-NN implementowaliśmy różne wartości ‘n’, począwszy od pierwszej, a skończywszy na setnej. W ten sposób dla jednego, trzech lub pięciu otrzymwaliśmy wynik na poziomie

97% dla problemu predykcji. Dla stu natomiast otrzymaliśmy wynik 62% pewności klasyfikacji. Pierwszy wynik grozi nam overfittingiem, natomiast drugi underfittingiem. Podczas pracy testowaliśmy różne wartości ‘n’ ostatecznie wybierając n = 37. Dla tej wartości otrzymujemy wartość na poziomie 92% co naszym zdaniem jest wystarczające dla naszego problemu. Poniżej tabela zawierająca dany gatunek oraz wartość przewidywaną dla ustalonej przez nas optymalnej wartości.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	Predicted
114	5.8	2.8	5.1	2.4	virginica	virginica
62	6.0	2.2	4.0	1.0	versicolor	versicolor
33	5.5	4.2	1.4	0.2	setosa	setosa
107	7.3	2.9	6.3	1.8	virginica	virginica
7	5.0	3.4	1.5	0.2	setosa	setosa
100	6.3	3.3	6.0	2.5	virginica	virginica
40	5.0	3.5	1.3	0.3	setosa	setosa
86	6.7	3.1	4.7	1.5	versicolor	virginica
76	6.8	2.8	4.8	1.4	versicolor	virginica
71	6.1	2.8	4.0	1.3	versicolor	versicolor
134	6.1	2.6	5.6	1.4	virginica	virginica
51	6.4	3.2	4.5	1.5	versicolor	versicolor

Rysunek 21 Tabela predykcji - k-NN

Naszym zdaniem klasyfikator k-NN jest prostszy w implementacji i pozwala na bardziej szeroką manipulację naszymi danymi. Najprawdopodbniej w trakcie pracy z AGV użyjemy tej metody stosując predykcję opartą o klasyfikację. Jeżeli uda nam się poprawnie pogrupować dane, czyli rozwiązać problem klasteryzacji będziemy w stanie lepiej sobie poradzić z przewidywaniem trasy jaką poruszać się będzie pojazd autonomiczny. Wniosek jest wysunięty na podstawie jednego, względnie nieskomplikowanego zbioru danych. W kolejnym rozdziale zajmiemy się predykcją win, tam cech jest zdecydowanie więcej i to pozwoli nam podjąć ostateczną decyzję.

Predykcja zbioru zawierającego wina

Rozdział ten będzie opierał się o analizę i implementację algorytmów predykcyjnych opisanych w rozdziale pierwszym dla zbioru danych zawierającego wina. Tak jak zostało wcześniej wspomniane, w przypadku win danych wejściowych jest więcej, a są to kolejno:

1. stała kwasowość (fixed acidity),
2. lotna kwasowość (volatile acidity),
3. kwas cytrynowy (citric acid),
4. cukier resztkowy (residual sugar),
5. chlorki (chlorides),
6. wolny ditlenek siarki (free sulfur dioxide),

7. ditlenek siarki ogółem (total sulfur dioxide),
8. gęstość (density),
9. pH,
10. siarczany (sulphates),
11. alkohol (alcohol).

Daną wyjściową tego zbioru jest jakość wybranego wina (quality), która umieszczona jest w skali od 0 do 10. Poniżej znajduje się tabela kilku pierwszych wartości zawartych w analizowanym zbiorze.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Rysunek 22 Pierwsze 4 elementy zbioru danych zawierającego wina

Algorytm regresji liniowej (Wines)

Jako, że rozpatrywać będziemy problem regersji liniowej, czyli nie uwzględniamy żadnych skategoryzowanych zmiennych, a to oznacza iż każda cecha jest oznaczona liczbą. Dając tym samym zbiór wartości określających cechy musimy przewidzieć jakość wybranego wina. Istotne było dla nas znalezienie korelacji między każdą z cech z naszą wartością oczekiwaną, czyli jakością. W tym celu wywołaliśmy na początku poniższą funkcję corr(), do której wrzuciliśmy nasz df, a tym samym usuwając kolumnę „quality".

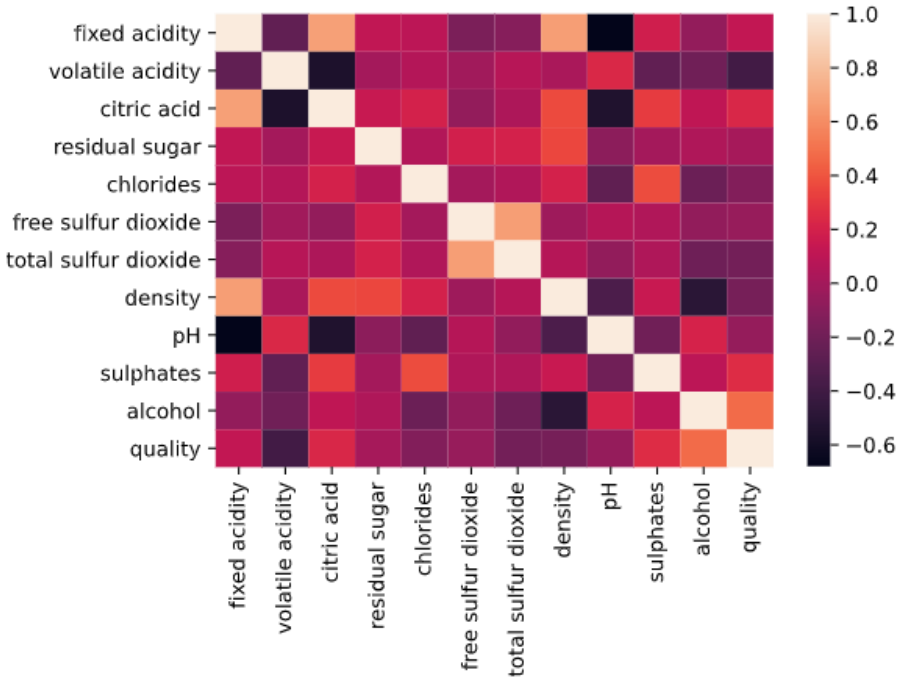
```
correlations = df.corr()['quality'].drop('quality')

print(correlations)
```

Oznaczenie „df" oznacza oczywiście ramkę danych utworzoną na podstawie zaciągniętego pliku .csv zawierającego wina. Ostatecznie po rzuceniu „quality" otrzymaliśmy poniższy wynik:

```
fixed acidity          0.124052
volatile acidity      -0.390558
citric acid            0.226373
residual sugar         0.013732
chlorides             -0.128907
free sulfur dioxide   -0.050656
total sulfur dioxide  -0.185100
density               -0.174919
pH                    -0.057731
sulphates              0.251397
alcohol                0.476166
Name: quality, dtype: float64
```

Rysunek 23 Korelacja między każdą z cech, a zmienną oczekiwaną

Na podstawie powyższego wywołaliśmy mapę ciepła, aby zobaczyć powyższą korelację w ujęciu graficznym.



Rysunek 24 Graficzne przedstawienie korelacji

W celu zmniejszenia ilości danych stworzyliśmy funkcję, która wydziela tylko te cechy, których wartość jest powyżej ustalonego przez nas progu. Następnie utworzony został wektor x zawierający opisane wcześniej cechy wejściowe wraz z wektorem y, który zawiera zmienną jakości. W x zawieramy wszystkie cechy z wyłączeniem cukru resztkowego. Próg korelacji wynosi w tym przypadku 5%.

Następnie podobnie jak dla iris zajęliśmy się stworzeniem datasetu przy pomocy funkcji train_test_split(), a następnie wykorzystaliśmy funkcję LinearRegression(). Na samym końcu wykonaliśmy predykcję. Reszta czasu została poświęcona na analizę. W tym celu ponownie obliczyliśmy miary jakości RMSE, MSE oraz MAE. Ostatecznie otrzymaliśmy poniższe wartości.

Mean Absolute Error: 0.48443407559847174

Mean Squared Error: 0.3938041346292098

Root Mean Squared Error: 0.627538153923098

Nasz model regresji liniowej dokonuje predykcji z RMSE równym około 63% dla zbioru testowego. Jest to co prawda wynik lepszy niż w przypadku iris natomiast dalej jesteśmy wobec niego sceptyczni, stąd też chylimy się ku rozwiązywaniu takich problemów predykcyjnych z wykorzystaniem klasyfikacji.

Algorytm k-NN (Wines)

Ostatnim analizowanym przez problemem predykcji było wykorzystanie algorytmu k-NN analogicznie jak w przypadku iris do działania na zbiorze zawierającym wina.

Podzieliliśmy nasze dataset na trzy klasy wina, tj. złe, średnie oraz dobre. Napisaliśmy w tym celu osobną funkcję, która by nam to obsłużyła.

```
quality = df["quality"].values
category = []
for num in quality:
    if num<5:
        category.append("Bad")
    elif num>6:
        category.append("Good")
    else:
        category.append("Medium")
```

Zgodnie z przypusczeniem najwiecej win zostało zaklasyfikowanych jako średnie, czyli znalazło się gdzieś pośrodku, zatem klasa mid jest klasą zawierającą najwięcej obiektów.

```
Medium    1319
Good       217
Bad         63
Name: category, dtype: int64
```

Analiza klasyfiaktora k-NN ponownie wiązała się z sprawdzaniem różnych wartości ‘n’, czyli liczby sąsiadów, znowuż począwszy od pierwszej, a skończywszy na setnej. W ten sposób udało nam się otrzymać wynik 82% dla jednego najbliższego sąsiada, 85% dla pięciu oraz 87% dla dwunastu najbliższych sąsiadów co też okazało się najwyższym możliwym do uzyskania wynikiem.

Biorąc pod uwagę to co wykonaliśmy w temacie predykcji będziemy zapewne opierać się w naszych rozważaniach głównie o k-NN. Być może wykorzystamy regresję jako algorytm walidacyjny, jeżeli jednak wyniki będą niesatysfakcjonujące spróbujemy wykorzystać jeszcze jeden wstępnie przez nas badany algorytm, tj. SVM czyli maszynę wektorów pomocniczych (Support Vector Machine).

Zgodnie z poprzednimi ustaleniami, naszą predykcję oprzemy o algorytm k-NN, ponieważ jest on łatwo zrozumiały, daje naszym zdaniem lepsze wyniki i jest prostszy w analizie i poszukiwaniu ewentualnych błędów.