



**Wykorzystanie optymalizacji za pomocą roju cząstek do rozpoznawania wzorców na obrazie**

Projekt zaliczeniowy na laboratorium z przedmiotu **Inteligencja Obliczeniowa** prowadzone przez **dr inż. Wojciecha Chmiela**

**Agnieszka Gazda  
Aleksandra Byra  
Radosław Wątroba**

Akademia Górniczo-Hutnicza  
Wydział Elektrotechniki, Automatyki,  
Informatyki i Inżynierii Biomedycznej  
Automatyka i Robotyka, spec.  
Inteligentne Systemy Sterowania  
(Neurocybernetyka), semestr II

Kraków, 2020

<b>Wstęp</b>	<b>4</b>
1.1 Opis zawartości dokumentacji	4
<b>Opis problemu</b>	<b>4</b>
<b>Algorytm PSO</b>	<b>5</b>
3.1 Opis algorytmu	5
3.2 Schemat (kroki) algorytmu (+ wzory)	6
3.3 Usprawnienia algorytmu	6
<b>Implementacja rozwiązania</b>	<b>7</b>
<b>Wyniki oraz testowanie</b>	<b>8</b>
<b>Usprawnienia i alternatywne rozwiązania</b>	<b>10</b>
<b>Podsumowanie</b>	<b>10</b>
<b>Źródła / bibliografia</b>	<b>10</b>

# 1. Wstęp

Poniższy dokument stanowi opis oraz dokumentację projektu, którego celem było opracowanie implementacji algorytmu rojowego (PSO) wykorzystywanego do rozpoznawania wskazanych wzorców na obrazach.

Inspiracją i pomocą w wykonaniu projektu miał być artykuł naukowy, który zgłębiał wspomniany temat [1].

## 1.1 Opis zawartości dokumentacji

Drugi rozdział zawiera opis problemu, któremu projekt stawia czoła.

W trzecim rozdziale dokonano opisu algorytmu rojowego, czyli podstawy do wykonania aplikacji wynikowej wraz z jego możliwymi ulepszeniami.

W czwartym rozdziale opisano zastosowaną implementację.

Piąty rozdział traktuje o testowaniu aplikacji wraz z jego wynikami.

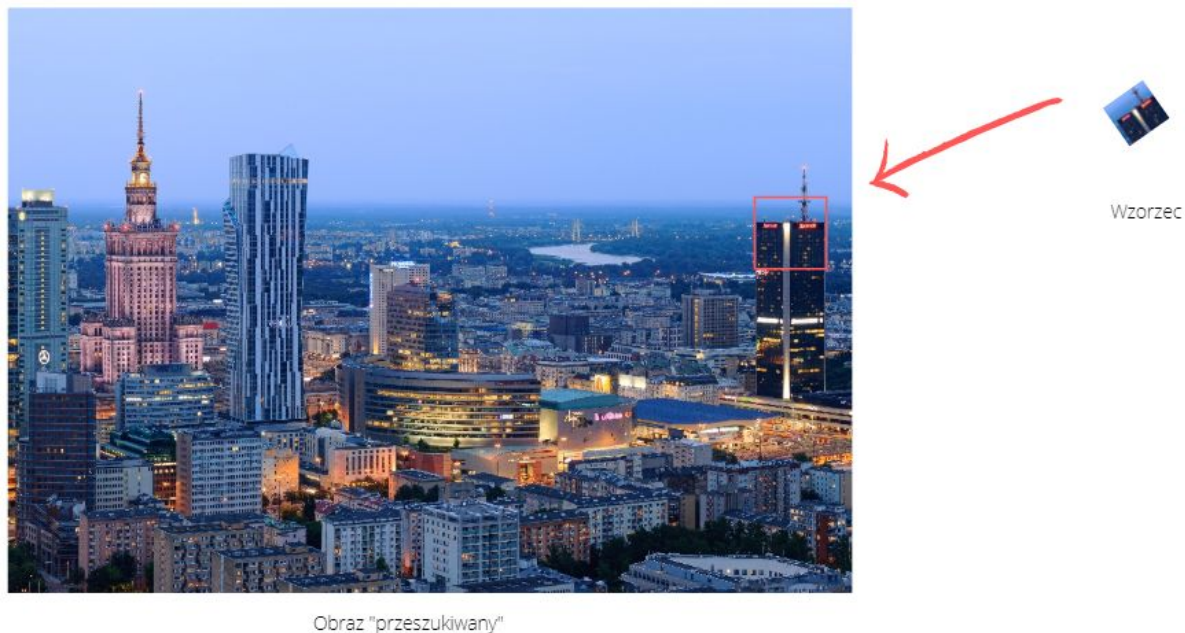
W szóstym rozdziale krótko wspomniano o możliwych rozwiązaniach alternatywnych dla problemu oraz ulepszeniach opisanego.

W siódmym rozdziale zamieszczono źródła oraz bibliografię.

# 2. Opis problemu

Wykrywanie wzorca na obrazie to problem szeroko znany w przetwarzaniu obrazów. W najbardziej podstawowej formie chodzi o znalezienie miejsca występowania (współrzędnych kartezjańskich) wyciętego fragmentu obrazu na tym samym obrazie.

Fragment obrazu nie musi dokładnie odwzorowywać oryginału. Algorytm PSO w opisanej formie dopuszcza również sytuacje, w których obraz jest obrocony lub przeskalowany. Optymalizacja wszystkich tych parametrów składa się na miarę podobieństwa, która powinna być maksymalizowana. Utrudnieniami w rozpoznawaniu wzorca mogą być także zmieniona jasność obrazu, przesłonięcie obiektu lub naniesienie szumów (np. filtru Gaussa, białego szumu).



Rys 1. Poszukiwanie wzorca na obrazie, źródło [2]

## 3. Algorytm PSO

### 3.1 Opis algorytmu

PSO (ang. Particle Swarm Optimization), czyli po polsku optymalizacja za pomocą roju cząstek lub optymalizacja stadna cząstek to algorytm, który został zaprojektowany w oparciu o zachowania stadne zwierząt (ptaków, ryb, owadów). Jego działanie opiera się na fakcie, że cząsteczka (osobnik) szukając optymalnej pozycji zdobywa doświadczenie oddziaływując na siebie wzajemnie z pozostałymi cząsteczkami. Skutkuje to stopniowym przenoszeniem się całej populacji w coraz lepsze obszary przestrzeni rozwiązań.

Algorytm PSO zaadaptowany do szukania wzorca na obrazie sprowadza się do znalezienia takich współrzędnych kartezjańskich oraz kąta obrotu fragmentu obrazu, aby zmaksymalizować miarę podobieństwa pomiędzy tym fragmentem a szukanym wzorcem.

Elementy rozwiązania można przedstawić jako poniższą krotkę:

$$(x, y, \theta),$$

gdzie:

- $x, y$  - współrzędne kartezjańskie;

- $\theta$  - kąt obrotu;

## 3.2 Schemat (kroki) algorytmu (+ wzory)

Schemat postępowania w algorytmie PSO wygląda następująco:

- 1) Inicjalizacja roju cząstek losowymi wartościami dla każdej cząstki.
- 2) Ocena jakości reprezentowanego rozwiązania pomocą wybranej miary prawdopodobieństwa.
- 3) Aktualizacja rozwiązania, jeśli obecne rozwiązanie (*gbest*) jest lepsze od rozwiązania obiektu (*pbest*) i odwrotnie.
- 4) Obliczenie nowej prędkości cząstek za pomocą wzoru:

$$Vel_i^{t+1} = w \cdot Vel_i^t + c_1 \cdot r_1 \cdot (pbest_i^t - x_i^t) + c_2 \cdot r_2 \cdot (gbest_i^t - x_i^t)$$

gdzie:

$w$  - moment inercji

$c_1, c_2$  - stałe przyspieszenia. Mają one bezpośredni wpływ na wielkość kroku poszukiwań. Za wysoka wartość  $c_1$  prowadzi do dzielenia się roju na małe izolowane grupy, zawierające nawet jedną cząstkę. Natomiast przy zbyt wysokiej wartości  $c_2$  rój ma tendencję do skupiania się w lokalnym maksimum, zatrzymując wyszukiwanie.

$r_1, r_2$  - równomiernie rozmieszczone liczby losowe.

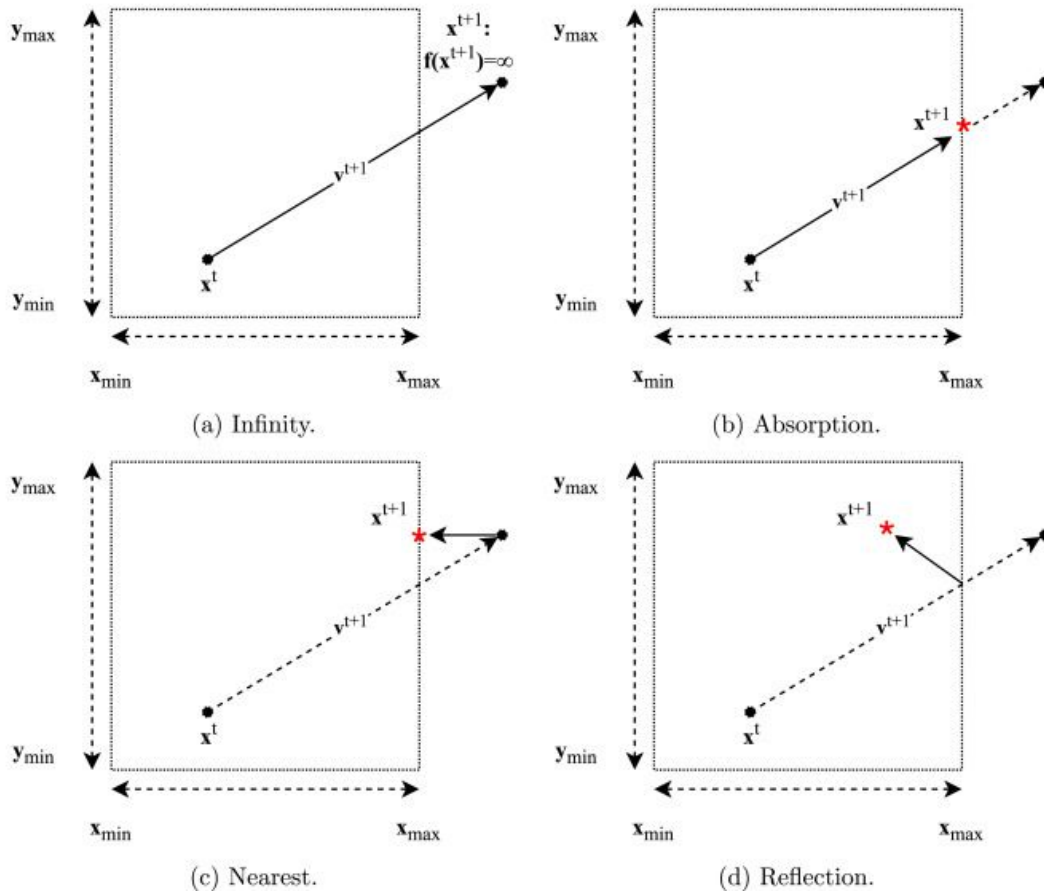
$x_i$  - aktualne rozwiązanie.

- 5) Obliczenie nowej pozycji cząstek w przestrzeni za pomocą wzoru:
- $$x_i^{t+1} = x_i^t + Vel_i^{t+1}$$
- 6) Stop, jeżeli spełnione jest kryterium zatrzymania, którym może być określona liczba iteracji bądź uzyskana dokładność wyniku.

## 3.3 Usprawnienia algorytmu

Dodatkowym krokiem, który można wykonać w celu poprawy wydajności PSO jest tzw. **eksplozja**. Jeśli *gbest* nie zmienia się przez kilka iteracji, najłatwiejszym sposobem, żeby rozwiązania PSO się poprawiały jest zatrzymanie roju i zrestartowanie PSO z niezmiennym *gbest*.

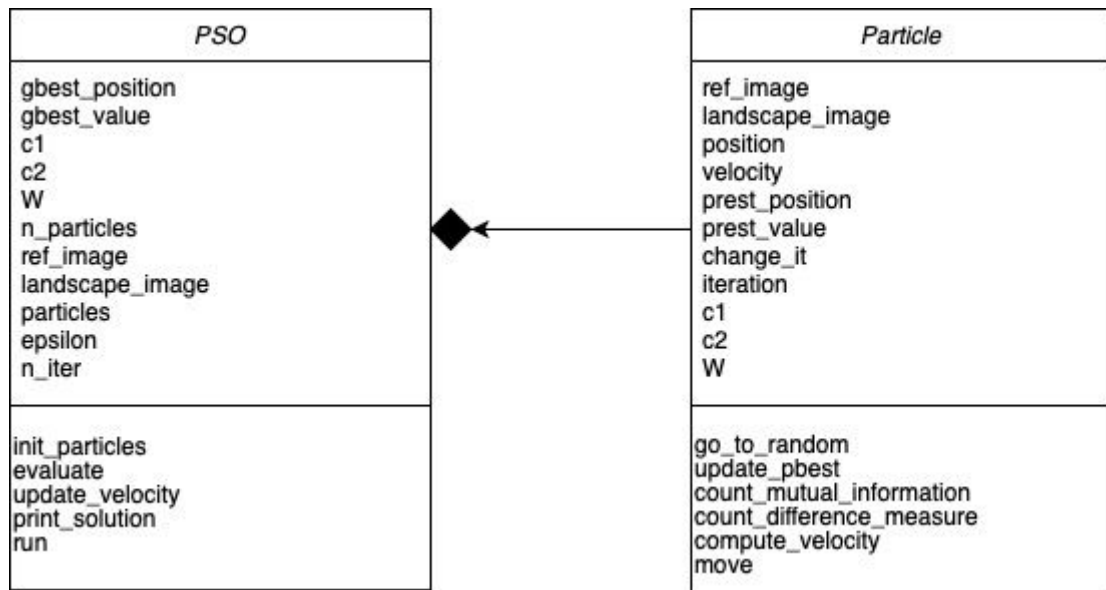
Kolejnym problemem, który można napotkać, jest wychodzenie cząstek roju za zakres. Rozwiązaniem może być losowe rozlokowanie cząstek, które miałyby wyjść poza obszar poszukiwań lub ich odbicie. Rysunek 3 prezentuje taką sytuację wraz z proponowanymi rozwiązaniami.



Rys. 2. Ograniczenia przestrzeni rozwiązań, źródło [3]

## 4. Implementacja rozwiązania

Algorytm zaimplementowano w języku Python z wykorzystaniem biblioteki OpenCV. Rozwiązanie składa się z dwóch klas: klasy Particle odpowiadającej pojedynczej cząsteczce oraz klasy PSO implementującej przestrzeń rozwiązań oraz algorytm zachowania poszczególnych cząstek. Strukturę rozwiązanie przedstawiono na Rys. 3.



Rys. 3. Struktura algorytmu PSO

Algorytm PSO polega na przeszukiwaniu przestrzeni rozwiązań do momentu znalezienia prawidłowego rozwiązania lub maksymalnej liczby iteracji. Przestrzenią rozwiązań jest obraz, na którym szukany jest zadany wzór.

Najważniejszym, twórczym elementem aplikacji jest wybór odpowiedniej funkcji podobieństwa. Podobieństwo między wzorem a aktualnie analizowanym wycinkiem przestrzeni rozwiązań liczone jest na dwa sposoby. W oryginalnym artykule funkcja podobieństwa była różnica obrazów. Jest to rozwiązanie słabo sprawdzające się w praktyce, ponieważ jest mało odporne, szczególnie na przesunięcia. Rozwiązanie z taką implementacją jest w stanie zwrócić prawidłowe rezultaty tylko dla bardzo wyróżniających się na tle wzorców.

Drugą zaimplementowaną funkcją jest miara wspólnej informacji. Miara ta opiera się o analizę histogramu 2D. Na podstawie histogramu obrazu wzorca i kandydata pozyskanego z tła oblicza się ich wzajemną informację. Miara ta mówi ile informacji o X można poznać, znając Y, czyli o ile poznanie jednej z tych zmiennych zmniejsza niepewność drugiej. Jeśli zmienne X i Y są niezależne, to ich wzajemna informacja jest zerowa, jeśli X i Y są identyczne, to każda zawiera pełną wiedzę o drugiej. Wzór pozwalający obliczyć tę miarę przedstawiony został poniżej:

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left( \frac{p(x, y)}{p(x) p(y)} \right)$$

gdzie:

- $p(x, y)$  - histogram 2D
- $p(x)$  - histogram jednowymiarowy wzorca
- $p(y)$  - histogram jednowymiarowy analizowanego fragmentu

Dodatkowo w aplikacji zaimplementowano rozszerzenia pozwalające na uzyskanie lepszych rezultatów. Jednym z nich była operacja wybuchu. Aby algorytm nie zatrzymywał się w lokalnych maksimach, gdy cząsteczka nie uzyskiwała lepszych rezultatów podczas poszukiwań swojej okolicy ( $p_{best}$  cząsteczki nie zmieniał się przez ustaloną liczbę iteracji) cząsteczkę przenoszono w inne, losowe miejsce na mapie przeszukiwań.

Dodatkowo jeśli cząsteczka próbowała wyjść poza obszar poszukiwań zostawała ona odpowiednio odbita lub wyrzucana w losowe miejsce na mapie.



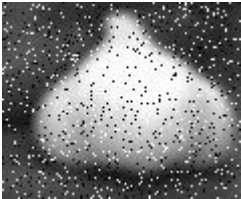
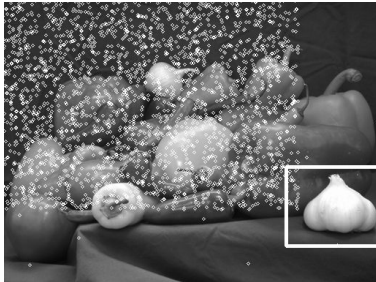
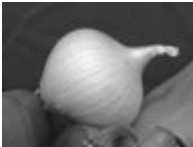
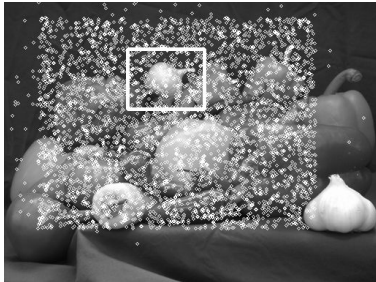

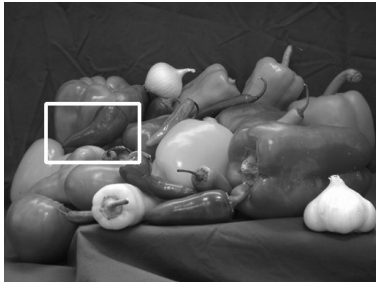


## 5. Wyniki oraz testowanie

Algorytm przetestowano w kilku różnych sytuacjach. Wyniki działania algorytmu przedstawiono w Tabeli 1. Algorytm przetestowano dla obrazów w odcieniach szarości w różnych rozmiarach, największy z nich był o wymiarach  $2800 \times 1760$  pikseli. Sprawdzono też jak algorytm radzi sobie z niewyraźnym wzorcem. Do obrazu wzorca dodano znaczne rozmycie filtrem Gaussa oraz szum typu salt&pepper. Aplikacja zwróciła poprawne wyniki także dla takiego przypadku.

Tab. 1. Wyniki działania aplikacji

Szukany wzór	Obraz wynikowy	Liczba iteracji i liczba cząsteczek
--------------	----------------	-------------------------------------



		Liczba iteracji: 1000 Liczba cząsteczek: 50
		Liczba iteracji: 1000 Liczba cząsteczek: 10
		Liczba iteracji: 100 Liczba cząsteczek: 10
		Liczba iteracji: 300 Liczba cząsteczek: 20
		Liczba iteracji: 100 Liczba cząsteczek: 20

## 6. Usprawnienia i alternatywne rozwiązania

Obecne rozwiązanie problemu sprowadza obraz oraz wzorec do skali szarości i dopiero wówczas stosowany jest algorytm. W artykule [1] autorzy wykonują obliczenia dla każdej ze składowych: R, G oraz B. Implementując takie rozwiązanie, algorytm byłby znacznie dokładniejszy podczas wyszukiwania spośród takich samych obiektów, takiego o wskazanym kolorze.

Sposobem powszechnie stosowanym w przetwarzaniu obrazów jest Uogólniona Transformata Hougha. Sprowadza się ona do porównywania obrysu krawędzi obiektu na obrazie z krawędzią referencyjną. Odbyna się to poprzez gromadzenie danych o linii należącej do konturu (kąt stycznej do punktu względem osi X, odległość od środka ciężkości kształtu). Jest ona dość dokładna, na dodatek nieczuła na obroty, a nawet niewielkie przesłonięcia obiektu. Jej zasadniczą wadą jest jednak wymagany duży nakład mocy obliczeniowej, co sprawia, że proces szukania trwa bardzo długo nawet na niewielkich obrazach.

## 7. Źródła / bibliografia

[1] - Perlin, H. A., Lopes, H. S., Centro, T. M. 2008. Particle Swarm Optimization for Object Recognition in Computer Vision. *IEA/AIE 2008*, LNAI 5072, s. 11-21

[2] -

<https://www.polityka.pl/tygodnikpolityka/mojemiasto/1794828,1,warszawa-swietuje-wolnosc.read>

[3] - <https://www.sciencedirect.com/science/article/abs/pii/S2210650218310198>

[4]

[https://matthew-brett.github.io/teaching/mutual\\_information.html?fbclid=IwAR0xDLqdNZCPaZx76hnbvJWEW5HO85XwoJN3sXJM-tB1pKcMO-bS1NGZSNk](https://matthew-brett.github.io/teaching/mutual_information.html?fbclid=IwAR0xDLqdNZCPaZx76hnbvJWEW5HO85XwoJN3sXJM-tB1pKcMO-bS1NGZSNk)