Fides, Ratio et Patria. Studia Toruńskie 19/2023
ISSN 2392-1463; e-ISSN 2956-3267
s. 340-355

https://doi.org/10.56583/frp.2564

Jan Edward Baumgart*

Uniwersytet Kazimierza Wielkiego w Bydgoszczy
https://orcid.org/0000-0003-1114-626X

Algorytmy Rojowe z wykorzystaniem MicroPython i Ulab dla mikrokontrolerów

Streszczenie

Głównym celem artykułu jest prezentacja możliwości implementacji algorytmu optymalizacji rojem cząsteczek (PSO) na platformach mikrokontrolerowych, w szczególności na płytce PyBoard. Poprzez rygorystyczne testowanie i analizę, badano zachowanie algorytmu w środowiskach o ograniczonych zasobach, dążąc do zrozumienia jego adaptacyjności, efektywności i stabilności. Przeprowadzono testy kodu PSO na trzech identycznych płytach PyBoard, wykonując algorytm wielokrotnie dla różnych funkcji testowych. Ten kompleksowy proces testowania pozwolił na uchwycenie i ocenę spójności wyników w różnych jednostkach mikrokontrolerów. Poprzez powtarzanie testów i zbieranie danych dotyczących wartości dopasowania, czasów wykonania i optymalnych rozwiązań, uzyskano cenne wglądy w zachowanie algorytmu. Wyniki analizy ujawniły kilka istotnych wniosków. Badanie potwierdziło, że mikrokontrolery, takie jak PyBoard, mogą być skutecznymi narzędziami do analizy i porównywania możliwości algorytmów optymalizacji, pomimo swoich ograniczonych zasobów. Podsumowując, wyniki badań dostarczyły cennych informacji na temat wydajności algorytmu PSO na platformach mikrokontrolerowych, otwierając nowe perspektywy dla projektowania systemów wbudowanych.

Słowa kluczowe: *optymalizacja rojem cząstek, platformy mikrokontrolerowe, pyboard, ulab, środowiska o ograniczonych zasobach, elastyczność, wydajność, stabilność, funkcje testowe, wartości przystosowania, czasy wykonania, optymalne rozwiązania, topografie optymalizacji, systemy wbudowane, analiza wydajności*

* Jan Edward Baumgart – magister inżynier, Katedra Systemów Inteligentnych Wydziału Informatyki Uniwersytetu Kazimierza Wielkiego w Bydgoszcz,
e-mail: Jan.Baumgart@ukw.edu.pl.

SWARM ALGORITHMS USING MICROPYTHON AND ULAB FOR MICROCONTROLLERS

Abstract

This study focuses on analyzing the possibilities of implementing the Particle Swarm Optimization (PSO) algorithm on microcontroller platforms, particularly on the PyBoard. Through rigorous testing and analysis, the behavior of the algorithm in resource-constrained environments was examined, aiming to understand its adaptability, efficiency, and stability. PSO code tests were conducted on three identical PyBoards, running the algorithm multiple times for various test functions. This comprehensive testing process allowed for capturing and evaluating the consistency of results across different microcontroller units. By repeating tests and gathering data on fitness values, execution times, and optimal solutions, valuable insights into the algorithm's behavior were obtained. The analysis results revealed several significant findings. For functions with simpler optimization landscapes, all devices achieved high effectiveness in finding the global optimum. In the case of more complex functions, the results obtained on microcontrollers were close to optimal values, suggesting good adaptability of the PSO algorithm. The study confirmed that microcontrollers like the PyBoard can be effective tools for analyzing and comparing the capabilities of optimization algorithms, despite their limited resources. In conclusion, the research results provided valuable information on the performance of the PSO algorithm on microcontroller platforms, opening new perspectives for embedded system design.

Keywords: *particle swarm optimization, microcontroller platforms, pyboard, ulab, resource--constrained environments, ,adaptability, efficiency, stability, test functions, fitness values, execution times, optimal solutions, optimization landscapes, embedded systems, performance analysis*

~ • ~

Wstęp

W dzisiejszym świecie technologia stale ewoluuje a mikrokontrolery stają się coraz bardziej powszechne i wszechstronne. Z ich pomocą można realizować różnorodne projekty, od prostych systemów wbudowanych[1] po zaawansowane urządzenia Internet of Things[2]. Jednakże, aby wykorzystać pełnię potencjału mikrokontrolerów, konieczne jest zapewnienie im efektywnych algorytmów. W tym kontekście, algorytmy rojowe stają się coraz bardziej popularnym narzę-

[1] M. A. Azam, S. Abdullah-Al-Nahid, M.M. Alam & B.A.,Plabon, *Microcontroller based high precision PSO algorithm for maximum solar power tracking,* "2012 International Conference on Informatics, Electronics & Vision (ICIEV)" 2012, pp. 292-297.

[2] A. Kurniawan, *Internet of Things Projects with ESP32: Build exciting and powerful IoT projects using the all-new Espressif ESP32,* 2019.

dziem do optymalizacji procesów biznesowych i nie tylko[3]. W niniejszym artykule skupimy się na analizie rozwiązań dla algorytmów rojowych w połączeniu z językiem programowania MicroPython oraz biblioteką uLab[4]. MicroPython, będący minimalistyczną implementacją języka Python stworzoną specjalnie z myślą o mikrokontrolerach, umożliwia ich programowanie w sposób intuicyjny i efektywny. Z kolei uLab, będący biblioteką numeryczną dla MicroPythona, zapewnia możliwość wykonywania szybkich operacji na danych numerycznych, co jest niezbędne przy implementacji algorytmów rojowych na mikrokontrolerach o ograniczonych zasobach. Poprzez połączenie tych trzech elementów – algorytmów rojowych, MicroPythona i uLab - otwieramy nowe perspektywy w zakresie projektowania systemów wbudowanych.

Wybór mikrokontrolera

W poszukiwaniu niezawodnych narzędzi do eksperymentów, wybór odpowiedniego mikrokontrolera odgrywa kluczową rolę. Ten rozdział zagłębia się w nasz metodyczny proces wyboru, porównując kilka możliwych mikrokontrolerów.

Pierwszym kandydatem było Raspberry Pi Zero [5]które wyłoniło się jako silny kandydat. Płytka oferuje wsparcie dla pełnego języka Python, w połączeniu z architekturą 32-bitową i imponującą prędkością taktowania 1GHz. Jednakże, konieczność korzystania z pełnego systemu operacyjnego budziła obawy co do procesów działających w tle, które potencjalnie mogą zakłócać nasze testy, co pokrywało się z wyzwaniami obserwowanymi podczas testowania algorytmów na standardowych komputerach osobistych. Dodatkowo relatywnie wysoka cena płytki sprawiła że postanowiono szukać dalej.

Kolejna płytka która została wzięta pod uwagę było Arduino Uno[6]: Mimo swojej powszechnej popularności, Arduino Uno nie spełniło zadanych kryteriów. Brak wsparcia dla Pythona lub MicroPythona stanowił istotne ograniczenie, zwłaszcza biorąc pod uwagę powszechność algorytmów rojowych stowarzyszonych głównie z Pythonem co niestety nie pozwoliłoby na dalszy rozwój bez konieczności przepisywania algorytmów rojowych na nowe języki programowania.

[3] J. Baumgart, D. Mikołajewski, J.M. Czerniak, *Taking Flight for a Greener Planet: How Swarming Could Help Monitor Air Pollution Sources*, "Electronics" 2024, 13(3), 577, https://doi.org/10.3390/electronics13030577

[4] Welcome to the ulab book! – The ulab book 6.5.0 documentation. (n.d.). https://micropython-ulab.readthedocs.io/en/latest/

[5] N.S. Yamanoor, S. Yamanoor, *High quality, low cost education with the Raspberry Pi*, "2017 IEEE Global Humanitarian Technology Conference (GHTC)" 2017, pp. 5-1.

[6] B. Arduino, *Arduino Uno. Datasheet,* https://datasheet.octopart. com/A000066-Arduino-datasheet-38879526. pdf. (Downloaded: 13 Jun 2020).

Następną przetestowaną płytką była Micro:Bit[7]. Choć Micro:Bit oferowało wsparcie dla MicroPythona, stosunkowo skromne specyfikacje - zwłaszcza prędkość taktowania 16MHz i brak sprzętowego zegara czasu rzeczywistego - zmniejszały przydatność dla naszych wymagań. Już po pierwszych testach można dojść do wniosku że uruchomienie nawet 100 przebiegów na płytce z takim taktowaniem byłoby problematyczne a brak sprzętowego zegara czasu rzeczywistego sprawiało problemy podczas analizy czasu pracy algorytmów.

Kolejne bardzo obiecujące dwie płytki są to następująco HUZZAH32 (ESP32) oraz WiPy3. HUZZAH32 intryguje imponującymi specyfikacjami, w tym prędkością taktowania 240MHz, dużą pamięcią flash (16MB), oraz wbudowanymi funkcjami Bluetooth, WiFi i sprzętowym zegarem czasu rzeczywistego. Jednakże, wyższa cena i ograniczona dostępność stwarzały wyzwania logistyczne, co ostatecznie doprowadziło do jego wykluczenia z naszej bieżącej linii testowej. Niemniej jednak, pozostaje on obiecującą opcją na przyszłe przedsięwzięcia, szczególnie dla zaawansowanych algorytmów z łącznością między urządzeniową poprzez Bluetooth czy WIFI. Podobny scenariusz dotyczy płytkę WiPy 3, podobnie jak HUZZAH32, oferowała szereg pożądanych funkcji, w tym wsparcie dla MicroPythona, dużą pamięć i wbudowane połączenie bezprzewodowe. Jednakże, znacznie wyższa cena i ograniczona dostępność sprawiły że obie te płytki do czasu gdy osiągną niższy pułap cenowy będą trudne w wykorzystaniu na większą skalę, co przy założeniach „roju" jest bardzo pożądane.

Po dokładnym rozważeniu ostatniego kandydata, płytka PyBoard [8]v.1.1 wyłoniła się jako wstępny preferowany wybór dla naszych badań. Płytka ta została stworzona specjalnie dla MicroPythona z prędkością 168 MHz, z niewielką lecz obecną pamięcią wbudowaną (1024KB Flash) oraz z wbudowanym sprzętowym zegarem czasu rzeczywistego wraz z bardzo niską ceną wyróżnił tą płytkę na tle konkurencji.

Na kolejnym etapie postanowiono przeanalizować zachowanie poszczególnych płytek podczas operacji na liczbach całkowitych i zmiennoprzecinkowych, co ma ogromne znaczenie dla zastosowania w algorytmach rojowych, do tej części badania wybrano tylko 3 najlepiej rokujące płytki, PyBoard v1.1 , HUZZAH32 oraz WiPy3.

Testy były zainspirowane przez podobne analizy na podstawie wcześniejszych badań. Podczas pierwszego testu obliczeń na liczbach całkowitych, płytki zostały postawione przed zadaniem znalezienia liczb pierwszych, każdy test był wykonany 5 krotnie.

[7] M. Kalogiannakis, E. Tzagkaraki, S. Papadakis, *A systematic review of the use of BBC micro: bit in primary school,* in: *Proceedings of the 10th Virtual Edition of the International Conference New Perspectives in Science Education,* Florence 2021, pp. 379-384.

[8] J. Parab, *MicroPython PyBoard for IoT*, in: *Python Programming Recipes for IoT Applictions,* Singapore 2023, pp. 89-122.

W przypadku testu operacji na liczbach całkowitych najlepszą płytką okazała się płytka HUZZAH32, na drugim miejscu znalazła się płytka PyBoard a na ostatnim WiPy3. Jest to zaskakujący wynik ponieważ pod względem sprzętowym płytka PyBoard jest znacząco słabsza płytką. Źródłem tego wyniku okazała się optymalizacja oprogramowania płytki.

Test dla liczb zmiennoprzecinkowych polegał na obliczeniu liczby PI. Dla każdej z płytek odchylenie błędu wynosiło dokładnie taką samą wartość, to gdzie płytki się różniły to czas wykonania obliczeń. Płytka PyBoard wykonała obliczenia o ponad 60% szybciej od płytki Huzzah32 i o 38% szybciej od płytki WiPy3.

Oba testy pozwoliły dość do konkluzji ze pomimo słabszych parametrów, płytka zaprojektowana i sprzedawana przez twórców MicroPython jest najlepiej zoptymalizowana do wykonywania obliczeń, szczególnie zmiennoprzecinkowych.

Podsumowując, proces selekcji podkreślił znaczenie dopasowania specyfikacji technicznych do celów badawczych, przy jednoczesnym uwzględnieniu czynników takich jak dostępność i cena oraz wyników testów dla operacji na liczbach całkowitych i zmiennoprzecinkowych. Płytka PyBoard v.1.1 wyłoniła się jako optymalny wybór, spełniając nasze kryteria dzięki natywnym wsparciu dla MicroPythona, solidnej wydajności i istotnym funkcjom oraz bardzo dobrym wynikom testów. Jednocześnie cechując się wysoką dostępnością na rynku oraz bardzo niską ceną, średnio 60% poniżej ceny pozostałych płytek. Ta decyzja jak później się okazało stanowiła solidne fundamenty dla naszych kolejnych eksperymentów, zapewniając precyzję, efektywność i kompatybilność z naszymi badaniami.

Analiza kodu

Analiza ogólnodostępnych algorytmów rojowych pod względem wymagań wsparcia dla bibliotek w języku Python ujawnia dominującą rolę biblioteki NumPy. [9]W większości przypadków algorytmy rojowe, takie jak optymalizacja rojem cząsteczek (PSO[10]), algorytmy mrówkowe (ACO) oraz ewolucyjne algorytmy genetyczne, polegają na NumPy do efektywnej manipulacji danymi. Biblioteka NumPy, skrót od „Numerical Python", jest kluczową biblioteką do obliczeń numerycznych w Pythonie. Dzięki swojej efektywności i wszechstronności, stała się preferowanym wyborem dla wielu badaczy i praktyków zajmujących się algorytmami rojowymi.

W optymalizacji rojem cząsteczek, NumPy jest wykorzystywany do zarządzania wektorami pozycji i prędkości cząsteczek, obliczania funkcji celu oraz aktualizacji ich położenia. Podobnie, algorytmy mrówkowe wykorzystują NumPy do

[9] T.E. Oliphant, *Guide to numpy*, 2006, Vol. 1, p. 85.

[10] J. Kennedy, R. Eberhart, *Particle swarm optimization*, "In Proceedings of ICNN'95-international conference on neural networks" 1995, Vol. 4, pp. 1942-1948.

obliczania feromonów i ich aktualizacji w śladzie mrówek, oraz symulacji ruchu samej populacji mrówek. Natomiast, w przypadku ewolucyjnych algorytmów genetycznych, NumPy pełni rolę w reprezentacji osobników, obliczaniu funkcji przystosowania, selekcji, krzyżowaniu oraz mutacji.

Korzyści wynikające z wykorzystania NumPy w implementacji algorytmów rojowych są liczne. Przede wszystkim, biblioteka ta oferuje wydajność i prostotę, co przekłada się na szybsze obliczenia i efektywne wykorzystanie zasobów obliczeniowych a także prosty czytelny kod. Dodatkowo, kod oparty na NumPy jest łatwo przenośny między różnymi platformami i środowiskami, co zwiększa elastyczność i dostępność implementacji.

Numpy w wielu implementacjach było wykorzystywane głównie do akcji takich jak obliczanie wartości COS, SIN, zwracanie równomiernie rozmieszczonych wartości w danym przedziale liczbowym czy co najważniejsze w metodach stochastycznych generowanie wartości losowych.

Niestety dla nas biblioteka NumPy nie jest dostępna w pełnej wersji w języku MicroPython, tak jak ma to miejsce w języku Python[11] co eliminuje wykorzystanie jej natywnie dla mikrokontrolerów. Za sprawą jednak społeczności MicroPython znacząca część biblioteki jest na bieżąco przenoszona do biblioteki Ulab.

Ulab to podobna do numpy biblioteka dla mikropythona i CircuitPythona. Moduł ten napisany jest w języku C. Biblioteka jest standardowym modułem użytkownika mikropythona, co oznacza, że nie ma zależności sprzętowych i można ją skompilować na dowolną platformę. Obsługiwane są 8- i 16-bitowe typy całkowite a także zmiennoprzecinkowe. To wszystko powoduje że Ulab jest pozornie idealnym zamiennikiem dla biblioteki NumPy dla środowiska MicroPython.

Nie wszystkie funkcje biblioteki NumPy zostały przeniesione, to sprawia że część algorytmów pozostaje niedostępna bez odpowiedniej zmiany kodu. Główną taką niedostępną funkcją przez długi czas była funkcja random() wykorzystywana w algorytmach rojowych do generowania liczb losowych. Sytuacja ta zmieniła się w styczniu tego roku, gdy moduł random został dodany jako część biblioteki Ulab, to sprawiło że znacząca część zarówno algorytmów stochastycznych oraz funkcji testowych stała się bardzo prosta do uruchomienia bez znaczących zmian w kodzie.

Przygotowanie płytki

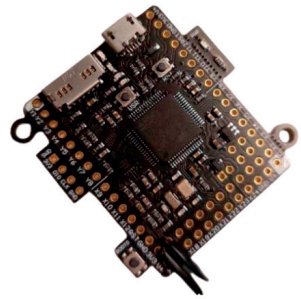
Do naszego zastosowania musimy uprzednio przygotować oprogramowanie poprzez dodanie biblioteki Ulab. Aby to zrobić skorzystano z procesu aktualizacji oprogramowania. Do procedury aktualizacji wykorzystano pliki udostępnione

[11] L.J. Miranda, *PySwarms: a research toolkit for Particle Swarm Optimization in Python*, "Journal of Open Source Software" 2018, 3(21), 433.

przez twórców biblioteki Ulab. Pliki te są dostępne do pobrania ze strony GitHub projektu.

Do aktualizacji płytek wykorzystano komputer z system MacOS (ale operacja bez problemu może być również wykonana na systemach Windows i Linux).

Na komputerze rozpoczęto od zainstalowania dfu-util za pomocą polecenia *brew install dfu-util.* Po zainstalowaniu oprogramowania, podłączono płytkę Pyboard i ustawiono ja w trybie DFU za pomocą zwarcia dwóch pinów na płytce, odpowiednio pin 3v3 oraz P1.



Rys. 1. Zdjęcie płytki PyBoard v1.1 przygotowanej do wejścia w tryb DFU

Następnie użyto dfu-util do przesłania pliku oprogramowania na mikrokontroler, wykonując odpowiednie polecenie z ścieżką do pobranego pliku '*sudo dfu-util --alt 0 -D firmware.dfu*' .

```
❯ dfu-util --alt 0 -D PYBV11-2.dfu
dfu-util 0.11

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2021 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Match vendor ID from file: 0483
Match product ID from file: df11
Multiple alternate interfaces for DfuSe file
Opening DFU capable USB device...
Device ID 314b:0106
Device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Interface #0 ...
Determining device status...
DFU state(2) = dfuIDLE, status(0) = No error condition is present
DFU mode device DFU version 011a
Device returned transfer size 2048
DfuSe interface name: "Internal Flash   "
File contains 1 DFU images
Parsing DFU image 1
Target name: ST...
Image for alternate setting 0, (2 elements, total size = 470296)
Setting Alternate Interface #0 ...
Parsing element 1, address = 0x08000000, size = 14864
Erase           [=========================] 100%        14864 bytes
Erase    done.
Download        [=========================] 100%        14864 bytes
Download done.
Parsing element 2, address = 0x08020000, size = 455416
Erase           [=========================] 100%       455416 bytes
Erase    done.
Download        [=========================] 100%       455416 bytes
Download done.
Done parsing DfuSe file
```

Rys. 2. Zrzut ekranu zakończonego procesu aktualizacji oprogramowania

Tak przygotowana płytka jest gotowa do uruchomienia naszych algorytmów z wykorzystaniem odpowiednich bibliotek.

Wybór i przygotowanie algorytmu

Algorytm PSO (Particle Swarm Optimization) jest jednym z najpopularniejszych algorytmów rojowych, stosowanych do optymalizacji różnorodnych problemów. Wybór algorytmu PSO nad innymi algorytmami rojowymi był podyktowany kilkoma czynnikami, które wyróżniają PSO i sprawiają, że jest on atrakcyjnym wyborem w kontekście uruchomienia na płytce PyBoard w MicroPython.

Algorytm PSO jest relatywnie prosty do zrozumienia i zaimplementowania. Jego koncepcja opiera się na zachowaniu roju cząsteczek, które poruszają się w przestrzeni poszukiwań w celu znalezienia optymalnego rozwiązania. Ta prostota implementacji czyni algorytm PSO idealnym wyborem dla projektów wymagających szybkiego wdrożenia. Mimo swojej prostoty, algorytm PSO jest skuteczny w rozwiązywaniu różnorodnych problemów optymalizacyjnych.[12] Działa on dobrze nawet w przypadku funkcji celu o wielu wymiarach i nieliniowych zależnościach. Jego zdolność do szybkiego zbiegania się do optymalnego rozwiązania czyni go atrakcyjnym wyborem w przypadku ograniczonych zasobów obliczeniowych.

Algorytm jest również elastyczny i łatwo może być dostosowany do różnych problemów optymalizacyjnych poprzez zmianę parametrów, takich jak liczba cząsteczek, współczynniki wagowe czy zakresy poszukiwań. Ta elastyczność czyni go uniwersalnym narzędziem, które można dostosować do specyfiki konkretnego problemu, z jakim się spotykamy. Dodatkowym czynnikiem, który przemawiał za wyborem algorytmu PSO nad innymi, był fakt, że został on już przetestowany na wielu platformach oraz porównany z innymi algorytmami rojowymi. Jego sprawdzona skuteczność oraz dostępność sprawiły, że PSO jest dobrze znany w dziedzinie optymalizacji, a jego kod źródłowy jest uznany za sprawdzony, dzięki swojej długiej historii i szerokiemu zastosowaniu, algorytm PSO stał się dobrze znany w dziedzinie optymalizacji.

PSO jest powszechnie używany w wielu dziedzinach nauki i przemysłu, co przyczyniło się do zgromadzenia bogatego zestawu narzędzi oraz wiedzy na temat jego działania i możliwości. Kod źródłowy algorytmu PSO został starannie przetestowany i jest uznany za niezawodny a dodatkowo zaimplementowany wielokrotnie. Dzięki temu, jego implementacja na płytce PyBoard w MicroPython może być wykonywana z większą pewnością co do poprawności działania oraz efektywności.

[12] J. Baumgart, B. Sangho, *A case study of the effectiveness of new methods of swarm optimization compared to known methods.* "Stud. Mater. Appl. Comput. Sci." 2021, 13(1), 47-50, ISSN: 1689-6300; C. Bell, *MicroPython for the Internet of Things*, Berlin, Heidelberg, Springer 2017.

Wdrożenie algorytmu na płytce PyBoard za pomocą języka MicroPython wymagało kilku dodatkowych kroków, szczególnie jeśli używane są zewnętrzne biblioteki lub narzędzia. Poniżej przedstawiamy sposób przygotowania oraz integracji algorytmu PSO z płytka PyBoard przy użyciu języka MicroPython.

Pierwszym krokiem jest zapewnienie, że kod będzie działał zarówno z biblioteką Numpy, jak i z biblioteką Ulab w przypadku, gdy Numpy nie jest dostępny. Dzieje się to za pomocą konstrukcji try-except, która sprawdza dostępność Ulab i importuje Numpy jako zastępcze rozwiązanie, jeśli jest to konieczne.

Oto przykładowy kod:

```
try:
  from ulab import numpy as np
except ImportError:
  import numpy as np
```

Rozwiązanie to sprawia, że w momencie gdy odpalamy kod na mikrokontrolerze będzie wykorzystywana biblioteka Ulab, a podczas pracy na komputerze nadal nasz kod będzie wykorzystywał bibliotekę numpy. To pozwala nam w prosty sposób porównywać wyniki dla obu bibliotek.

Następnym krokiem jest przygotowanie oprogramowania do uruchamiania algorytmów na płytce PyBoard. W poniższym fragmencie kodu definiujemy funkcję run_external_file, która wykonuje zewnętrzny skrypt na płytce PyBoard. Oto przykładowy kod:

```
import pyboard

def run_external_file(pyb, file_path):
  with open(file_path, 'r') as file:
    script_content = file.read()
  pyb.enter_raw_repl()
  ret = pyb.exec(script_content)
  pyb.exit_raw_repl()
  return ret

if __name__ == "__main__":
  file_path = "pso.py"
  devices = ["/dev/tty.usbmodem*"]
  for device in devices:
    pyb = pyboard.Pyboard(device, 115200)
    for i in range(1):
      result = run_external_file(pyb, file_path)
      decoded_data = result.decode("utf-8")
      print(decoded_data)
```

Kod ten wymaga modułu obsługi płytki PyBoard, moduł ten jest dostępny do pobrania z głównej strony projektu MicroPython.

Badania

W tej sekcji zagłębimy się w metodologię stosowaną do oceny algorytmu PSO na płytce Pyboard. Proces badań polegał na poddaniu kodu PSO testom na trzech identycznych płytkach oraz na komputerze osobistym. Aby zapewnić kompleksową analizę, kod PSO został wykonany 250 razy dla każdej funkcji testowej na każdym z urządzeń. Ten reżim testowania miał na celu uchwycenie szerokiego spektrum scenariuszy wydajności, oraz ocenić rozbieżność wyników między poszczególnymi urządzeniami. Powtarzanie testów było kluczowe w zmniejszeniu wpływu losowości charakterystycznej dla algorytmów PSO i dostarczeniu statystycznie istotnych wyników.

Co więcej, aby ustalić punkt odniesienia do porównania i zapewnienie poprawności wyników, kod PSO został również wykonany 250 razy na komputerze osobistym. Ten krok posłużył również jako punkt odniesienia do wydajności algorytmu na mikrokontrolerach w stosunku do konwencjonalnych platform obliczeniowych.

Podczas testów użyliśmy trzech różnych funkcji testowych[13][14], aby sprawdzić adaptacyjność algorytmu PSO w różnych warunkach. Pierwsza funkcja, DeVilliers-Glasser[15], była testem o wysokim wymiarze (dim=4) i dużym zakresie pola potencjalnych rozwiązań (-500 do 500). Druga funkcja, VenterSobiezcczanskiSobieski[16], miała większy wymiar (dim=10) ale krótszą liczbę iteracji (max_iter=40). Trzecia funkcja, CrossLegTable[17], była testem dwuwymiarowych i o krótkim czasie działania (max_iter=20).

Dla każdego testu zebraliśmy dane dotyczące wartości funkcji celu (fitness), czasu wykonania (runtime) oraz najlepszego znalezionego rozwiązania. Analiza tych danych pozwoliła nam na lepsze zrozumienie zachowania algorytmu PSO w różnych warunkach. Zebrane dane poddano analizie, aby wyciągnąć znaczące wnioski na temat zachowania algorytmu PSO na mikrokontrolerach.

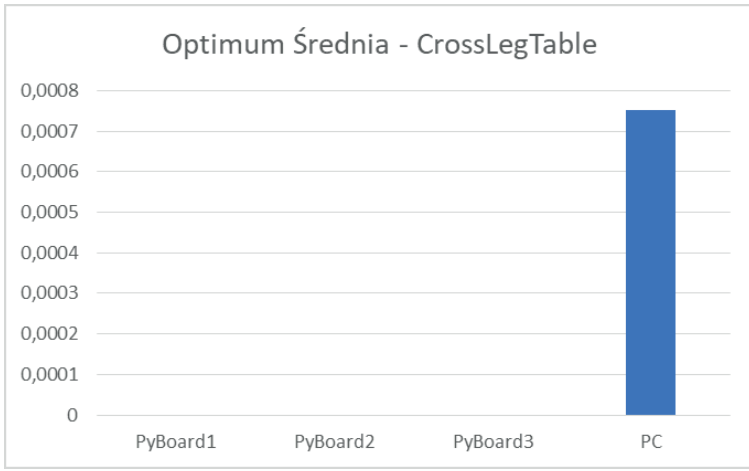
[13] M. Molga, C. Smutnicki, *Test functions for optimization needs*, 2005, 101, 48.

[14] M. Jamil, X.-S. Yang, *A Literature Survey of Benchmark Functions For Global Optimization, Problems*. "Int. Journal of Mathematical Modelling and Numerical Optimisation" 2013, 4, 150-194.

[15] S.K. Mishra, *Some new test functions for global optimization and performance of repulsive particle swarm method. Available at SSRN 926132*, 2006.

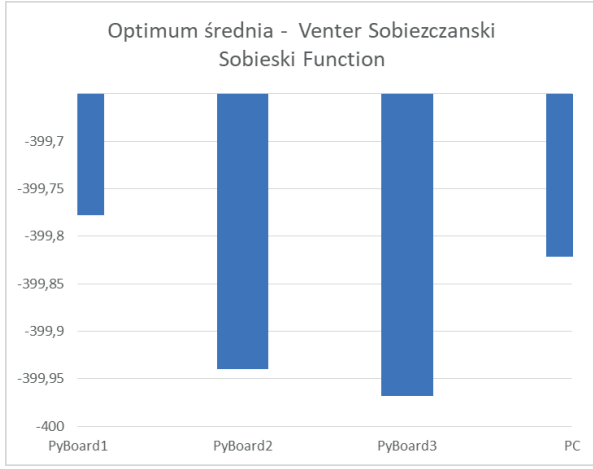
[16] *Problems* "Int. Journal of Mathematical Modelling and Numerical Optimisation" 2013, 4, 150-194; G. Venter, J. Sobieszczanski-Sobieski, *Particle Swarm Optimization*. "AIAA Journal" 2002, 41. 10.2514/2.2111.

[17] S.K. Mishra, *Some new test functions for global optimization and performance of repulsive particle swarm method,* 2006, Available at SSRN 926132.



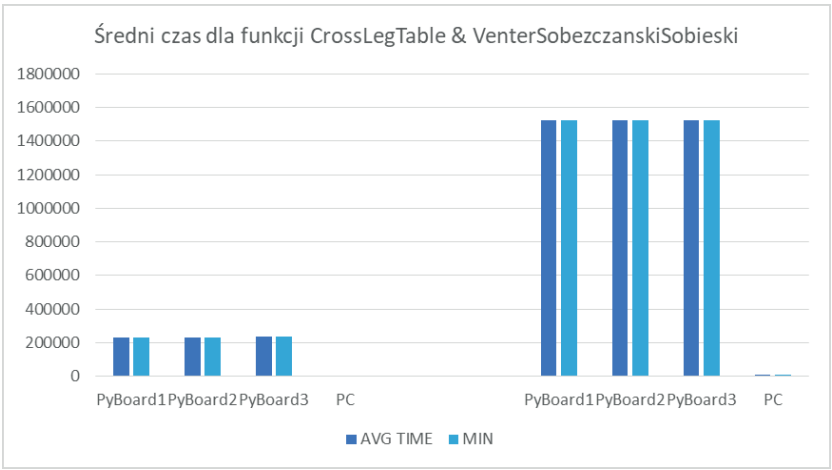
Wykres 1. Wykres przedstawiający średni wynik algorytmu dla funkcji CrossLegTable dla poszczególnych urządzeń

Dla funkcji CrossLegTable każda z funkcji uruchomionych na mikrokontrolerze osiągnęła 100% skuteczność w odnalezieniu globalnego optimum, w przypadku funkcji uruchomionych na komputerze funkcja nie osiągała idealnego wyniku.

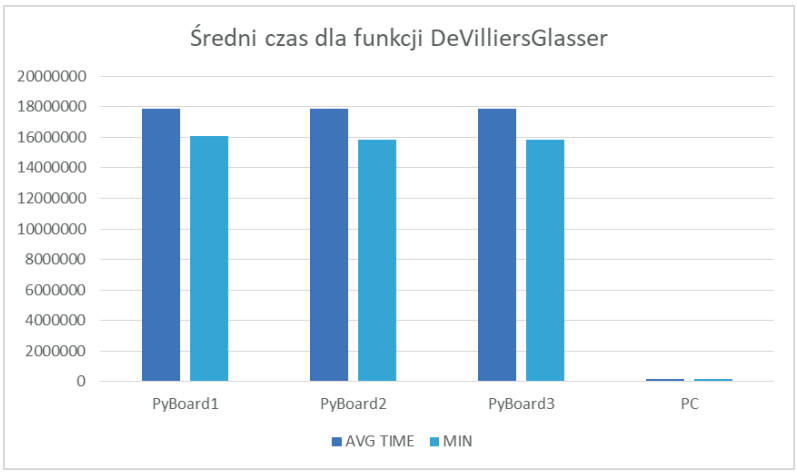


Wykres 2. Przedstawia średni wynik algorytmu dla funkcji VenterSobiezczanskiSobieski dla poszczególnych urządzeń

Dla funkcji VenterSobiezczanskiSobieski każda z funkcji uruchomionych na mikrokontrolerze osiągnęła wysoką skuteczność w odnalezieniu globalnego optimum, co interesujące wyniki wygenerowane na komputerze ponownie nie były wynikami najlepszymi. Wystąpiła również lekka rozbieżność w przypadku testów uruchomionych na mikrokontrolerach, jednak wyniki były bardzo zadowalające.



Wykres 3. Przedstawia średni oraz minimalny czas pracy algorytmu dla funkcji CrossLeg Table oraz Venter Sobiezczanski Sobieski dla poszczególnych urządzeń



Wykres 4. Przedstawia średni oraz minimalny czas pracy algorytmu dla funkcji DeVilliers-Glasser dla poszczególnych urządzeń

W przypadku funkcji Cross Leg Table, wszystkie urządzenia osiągają idealną wartość dopasowania (AVG) na poziomie 0 lub wyjątkowo blisko, co wskazuje na udaną konwergencję PSO do optymalnego rozwiązania. Niemniej jednak, istnieją niewielkie różnice w czasie obliczeń. PyBoard1, PyBoard2 i PyBoard3 wykazują podobne wartości w średnim czasie, oscylujące wokół 233 000 mikrosekund, podczas gdy PC wykazuje szybszą wydajność na poziomie 2045,81 mikrosekund. Minimalny czas (MIN) osiągnięte przez wszystkie płytki również jest bardzo zbliżony.

Funkcja VenterSobiezcczanskiSobieski prezentuje bardziej skomplikowany krajobraz optymalizacyjny. W tym przypadku, średnie wartości osiągniętego wyniku różnią się w zależności od typu urządzenia. Wyniki sugerują, że PSO napotyka trudności w osiągnięciu optymalnych rozwiązań, wyniki są jednak bardzo zbliżone do wartości optimum globalnego.

Podobnie również Funkcja DeVilliers-Glasser która charakteryzuje się negatywnymi wartościami dopasowania, co stwarza wyzwania dla optymalizacji. Wszystkie urządzenia osiągają taki sam wynik, może to wskazywać na to, że funkcja DeVilliers-Glasser jest mniej wrażliwa na zasoby obliczeniowe.

Co ciekawe, wszystkie mikrokontrolery osiągają porównywalne czasy obliczeń w tym przypadku, oznacza to że można wyjść z założenia że mikrokontrolery stanowią idealną podstawę jako ogólnodostępne narzędzie do analizy i porównywania możliwości nowo zaproponowanych algorytmów.

Podsumowanie

W niniejszym badaniu zagłębiono się w analizę możliwości implementacyjnych algorytmu optymalizacji rojem cząsteczek (PSO) na platformach opartych na mikrokontrolerach, takich jak Pyboards. Poprzez rygorystyczne testowanie i analizę, mieliśmy na celu zrozumienie zachowania się algorytmu w środowiskach o ograniczonych zasobach i dostarczenie wglądu w jego zdolność do adaptacji i efektywności oraz stabilności.

Badania objęły testowanie kodu PSO na trzech identycznych mikrokontrolerach wykonując algorytm wielokrotnie dla każdej funkcji testowej. Ten kompleksowy sposób testowania pozwolił nam uchwycić i ocenić spójność wyników w różnych jednostkach mikrokontrolerów. Poprzez powtarzanie testów i zbieranie danych dotyczących wartości dopasowania, czasów wykonania i optymalnych rozwiązań, zdobyliśmy cenne wglądy w zachowanie algorytmu. Wyniki zamieszczonej analizy ujawniły interesujące odkrycia. Dla funkcji o prostszych krajobrazach optymalizacyjnych, takich jak funkcja Cross Leg Table, wszystkie urządzenia wykazały udane zbieżności do optymalnych rozwiązań, chociaż z niewielkimi różnicami w czasach wykonania. Jednakże, dla bardziej skomplikowanych funkcji, takich jak VenterSobiezcczanskiSobieski, zarówno komputer osobisty jak i płytki z mikrokontrolerami miały trudności z osiągnięciem optymalnej zbieżności, co wskazuje na podobne zachowanie algorytmu niezależnie od platformy. Mikrokontrolery ze wsparciem MicroPython wydają się zatem być idealną platformą do rzetelnego odtwarzania badań, pomimo użycia 3 różnych urządzeń wyniki były bardzo zbliżone i stabilne ponadto poza różnicą. Pomimo dłuższego czasu pracy niż w wykonaniu komputera personalnego, co jest bezpośrednim wynikiem

ograniczonych zasobów. Algorytm bez problemu odnalazł globalne optima na płaszczyźnie przeszukiwań.

Patrząc w przyszłość, z naszej analizy wynika kilka ścieżek do dalszych badań. Po pierwsze, istnieje potrzeba dalszej implementacji algorytmów rojowych specjalnie dostosowanych do wdrażania na platformach mikrokontrolerów. Optymalizacja ta mogłaby obejmować dopracowanie algorytmu, eksplorację alternatywnych technik inteligencji rojowej[18] lub opracowanie specjalistycznych implementacji sprzętowych[19] w tym rozszerzenie możliwości biblioteki Ulab. Dodatkowo, nasze wyniki podkreślają znaczenie metodologii benchmarkingu i oceny wydajności do oceny efektywności algorytmów w różnych środowiskach obliczeniowych[20]. Przyszłe badania mogą rozwijać naszą metodologię, aby zbadać wydajność innych algorytmów optymalizacyjnych na mikrokontrolerach tak aby ułatwić odtwarzanie badań bez konieczności posiadania dostępu do sprzętu o dużej mocy obliczeniowej. Ponadto, wnioski płynące z naszych badań mogą posłużyć do zastosowań Internetu Rzeczy (IoT),[21] gdzie występują ograniczenia zasobów. Wykorzystując techniki inteligencji rojowej zoptymalizowane pod kątem architektury mikrokontrolerów, urządzenia IoT mogą autonomicznie optymalizować swoje działania, jednocześnie oszczędzając energię i zasoby obliczeniowe czy rozwiązując inne problemy natury optymalizacyjnej[22] w trybie „na żywo".

Podsumowując, niniejsze badanie dostarcza cennych wglądów w wydajność optymalizacji rojem cząsteczek na platformach mikrokontrolerów i kładzie podwaliny pod przyszłe badania w zakresie inteligencji rojowej i wbudowanej optymalizacji. Poprzez rozwiązanie wyzwań związanych z środowiskami obliczeniowymi na mikrokontrolerach, możemy odblokować nowe możliwości wdrażania inteligentnych algorytmów w aplikacjach rzeczywistych.

---

18 D. Ewald, J.M. Czerniak, H. Zarzycki, *OFNBee method used for solving a set of benchmarks*, in: *Advances in Fuzzy Logic and Technology 2017: Proceedings of: EUSFLAT-2017–The 10th Conference of the European Society for Fuzzy Logic and Technology, September 11-15, 2017, Warsaw, Poland IWIFSGN'2017–The Sixteenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets, September 13-15, 2017*, Warsaw 2018, Volume 2 (10), pp. 24-35.

19 S.Magnenat, P. Rétornaz, B. Noris, F. Mondada, *Scripting the swarm: event-based control of microcontroller-based robots,* in: *SIMPAR 2008 workshop proceedings*, 2008.

20 D. Ewald, J.M. Czerniak, M. Paprzycki, *A new OFNBee method as an example of fuzzy observance applied for ABC optimization. Theory and Applications of Ordered Fuzzy Numbers: A Tribute to Professor Witold Kosiński,* 2017, 223-237.

21 J. Czerniak, G. Śmigielski, D. Ewald, M. Paprzycki, W. Dobrosielski, *New proposed implementation of ABC method to optimization of water capsule flight,* in: *2015 Federated Conference on Computer Science and Information Systems (FedCSIS),* 2015, pp. 489-493.

22 J. Baumgart, D. Mikołajewski, J.M. Czerniak, *Taking Flight for a Greener Planet: How Swarming Could Help Monitor Air Pollution Sources*, "Electronics" 2024, 13 (3), 577, https://doi.org/10.3390/electronics13030577

References

Bibliografia

Azam M. A., Abdullah-Al-Nahid S., Alam M. M., Plabon B. A., *Microcontroller based high precision PSO algorithm for maximum solar power tracking,* in: *2012 International Conference on Informatics, Electronics & Vision (ICIEV),* 2012, pp. 292-297.

Baumgart J., Mikołajewski D., Czerniak J.M., *Taking Flight for a Greener Planet: How Swarming Could Help Monitor Air Pollution Sources*. "Electronics" 2024, 13(3), 577. https://doi.org/10.3390/electronics13030577

Bell C., *MicroPython for the Internet of Things*, Berlin, Heidelberg, Springer 2017.

Czerniak J., Śmigielski G., Ewald D., Paprzycki M., Dobrosielski W., *New proposed implementation of ABC method to optimization of water capsule flight,* in: *2015 Federated Conference on Computer Science and Information Systems (FedCSIS),* 2015, pp. 489-493.

Kalogiannakis M., Tzagkaraki E., Papadakis S., *A systematic review of the use of BBC micro: bit in primary school. In Proceedings of the 10th Virtual Edition of the International Conference New Perspectives in Science Education,* Florence 2021, pp. 379-384.

Ewald D., Czerniak J. M., Paprzycki M., *A new OFNBee method as an example of fuzzy observance applied for ABC optimization. Theory and Applications of Ordered Fuzzy Numbers: A Tribute to Professor Witold Kosiński*, 2017, 223-237.

Miranda L. J., *PySwarms: a research toolkit for Particle Swarm Optimization in Python,* "Journal of Open Source Software" 2018, 3(21), 433.

Baumgart J., Sangho B., *A case study of the effectiveness of new methods of swarm optimization compared to known methods,* "Stud. Mater. Appl. Comput. Sci." 2021, 50–47 ,(1)13, ISSN: 1689–6300

Ewald D., Czerniak J. M., Zarzycki H., *OFNBee method used for solving a set of benchmarks,* in: *Advances in Fuzzy Logic and Technology 2017: Proceedings of: EUSFLAT-2017–The 10th Conference of the European Society for Fuzzy Logic and Technology, September 11-15, 2017, Warsaw, Poland IWIFSGN'2017–The Sixteenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets, September 13-15, 2017*, Warsaw 2018, Volume 2 10, pp. 24-35.

Jamil M., Yang X.-S. *A Literature Survey of Benchmark Functions For Global Optimization Problems,* "Int. Journal of Mathematical Modelling and Numerical Optimisation" 2013, 4, 150-194.

Kennedy J., Eberhart R., *Particle swarm optimization*, in: *Proceedings of ICNN'95-international conference on neural networks* 1995, Vol. 4, pp. 1942-1948.

Kurniawan A., *Internet of Things Projects with ESP32: Build exciting and powerful IoT projects using the all-new Espressif ESP32*, 2019.

Magnenat S., Rétornaz P., Noris B., Mondada F., *Scripting the swarm: event-based control of microcontroller-based robots,* in: *SIMPAR 2008 workshop proceedings*, 2008.

Mishra S. K., *Some new test functions for global optimization and performance of repulsive particle swarm method,* 2006 Available at SSRN 926132.

Molga M., Smutnicki C., *Test functions for optimization needs*, 2005, 101, 48.

Oliphant T. E., *Guide to numpy*, 2006, Vol. 1, p. 85.

Parab J., *MicroPython PyBoard for IoT*, in: *Python Programming Recipes for IoT Applictions,* Singapore 2023, pp. 89-122.

Problems "Int. Journal of Mathematical Modelling and Numerical Optimisation" 2013, 4, 150-194.

Venter G., Sobieszczanski-Sobieski J., *Particle Swarm Optimization*. "AIAA Journal" 2002, 41. 10.2514/2.2111.

Yamanoor N. S., Yamanoor S., *High quality, low cost education with the Raspberry Pi.* In *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, 2017, pp. 1-5.

Netografia

Arduino B., *Arduino Uno. Datasheet*, 2015 (Downloaded: 13 Jun 2020). https://datasheet. octopart. com/A000066-Arduino-datasheet-38879526. pdf.

Welcome to the ulab book! — The ulab book 6.5.0 documentation. (n.d.). https://micropython-ulab. readthedocs.io/en/latest/