POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Systemy informatyczne w automatyce (ASI)

PRACA DYPLOMOWA MAGISTERSKA

Filtry cząsteczkowe i ich zastosowania
w problemach wyznaczania lokalizacji

Particle filters for selected problems in positioning

AUTOR:
Wojciech Sopot

PROWADZĄCY PRACĘ:
dr hab. inż. Paweł Wachel, prof. ucz., W04/K8

WROCŁAW 2021

Spis treści

Rozdział 1

Wprowadzenie

W rozdziale przedstawiono cel i zakres pracy oraz podstawowe zadania i problemy wyznaczania lokalizacji. Ponadto opisano ogólną ideę stojącą za filtrami cząsteczkowymi - filtrację Bayesowską. Praca jest inspirowana artykułem [9].

1.1 Zadania i problemy wyznaczania lokalizacji

Mogłoby się wydawać, iż w dzisiejszych czasach, gdy mamy możliwość korzystania z systemu GPS, nie ma potrzeby zajmować się innymi sposobami wyznaczania lokalizacji. Jednak o ile jest to prawdą w dużej skali, jak np. gdy chce się wyznaczyć adres w mieście pod którym się znajdujemy, to gdy chcemy wyznaczyć lokalizację bardziej dokładnie, np. tak jak to robią niektóre odkurzacze mobilne, trzeba wykorzystać do tego dane z innych sensorów, np. lidarów. Czasami nie ma możliwości korzystania z systemu GPS, na przykład pod wodą. Innym problemem może być poprawa już znanego przybliżonego położenia. W pracy zostaną przeanalizowane dwa problemy związane z wyznaczaniem lokalizacji:

- Określanie położenia samolotu, na podstawie znanej mapy wysokościowej terenu, odczytów z wysokościomierza barometrycznego, oraz, ewentualnie, dodatkowych danych (np. odczytu z kompasu).

- Określanie lokalizacji robota mobilnego, umieszczonego w ograniczonej przestrzeni z przeszkodami (ale na płaskiej powierzchni - mapa jest w dwóch wymiarach).

Mimo tego, iż są to jedynie dwa przypadki, w praktyce można je uogólnić na wiele sytuacji. Dla przykładu w [16] zajęto się problemem lokalizacji pod wodą, który w praktyce można rozwiązać w ten sam sposób co problem samolotu - mapa wysokości zostaje jedynie zastąpiona przez mapę głębokości.

1.2 Cel pracy

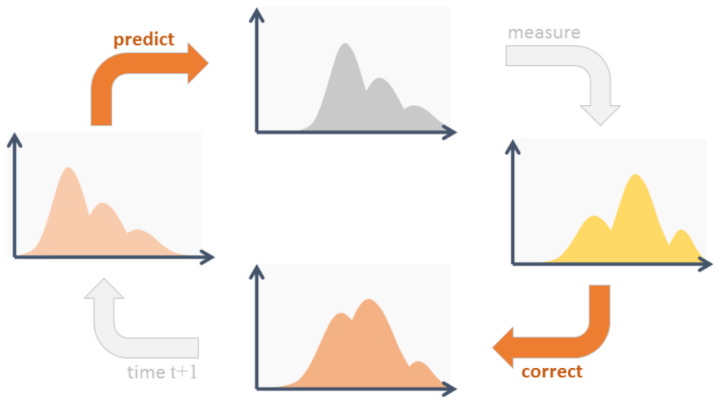
Celem pracy jest rozpoznanie możliwości zastosowania filtrów cząsteczkowych do rozwiązywania problemu lokalizacji oraz samodzielne zaimplementowanie i przebadanie kilku technik, uznanych arbitralnie przez autora za najciekawsze.

1.3 Zakres pracy

W zakres pracy wchodzi przegląd literatury na temat filtrów cząsteczkowych pod kątem problemów wyznaczania lokalizacji, następnie wybranie kilku rozwiązań oraz ich przebadanie. Aby było to możliwe konieczny jest także przegląd znanych narzędzi programistycznych, które mogą być zastosowane do zaimplementowania algorytmów.

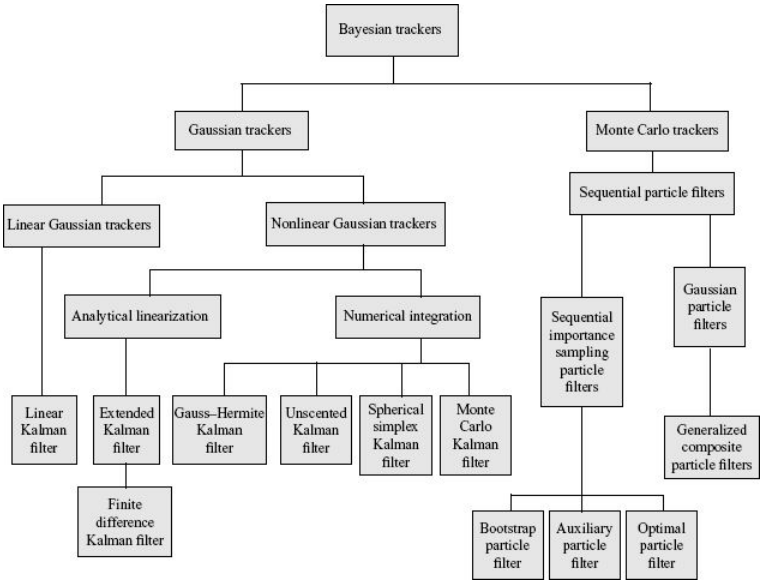
1.4 Idea filtracji Bayesowskiej

Na chwilę obecną, algorytmy oparte o filtrację Bayesowską są szeroko wykorzystywane w automatyce i robotyce. W skrócie, podejście to polega na przeprowadzaniu cykli predykcji i poprawek, na podstawie rozkładu a priori oraz zbieranych w kolejnych iteracjach pomiarów, w celu wyznaczenia rozkładu a posteriori stanu systemu. Ideę tego podejścia widać na rysunku (1.1).



Rysunek 1.1: Idea filtracji Bayesowskiej [4]

Na rysunku (1.2) przedstawiono hierarchię rozwiązań opartych o to podejście. Jak widać jest to bardzo duża rodzina rozwiązań, jednak w pracy zostaną poruszone jedynie metody oparte o filtry cząsteczkowe.



Rysunek 1.2: Hierarchia filtrów opartych o filtrację Bayesowską [11]

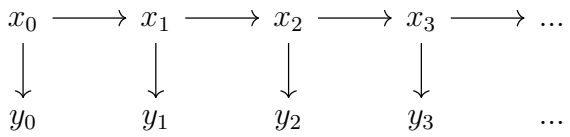
Rozdział 2

Filtry cząsteczkowe

W tym rozdziale został opisany podstawowy algorytm filtra cząsteczkowego, wraz z jego adaptacją do wyznaczania lokalizacji. Ponadto opisano tutaj kilka wybranych usprawnień i modyfikacji podstawowej wersji.

2.1 Opis problemu

Celem filtru cząsteczkowego jest wyznaczenie rozkładu wektora stanu x_k, na podstawie szczątkowych pomiarów y_k za pomocą filtracji Bayesowskiej.



Rysunek 2.1: Graficzna reprezentacja ewolucji stanu x oraz odpowiadających mu pomiarów y

System przedstawiony na rysunku (2.1) można opisać w następujący sposób:

$$\begin{aligned} x_k &= g(x_{k-1}) + w_{k-1} \\ y_k &= h(x_k) + v_k \end{aligned}$$

gdzie g i h to znane funkcje, a w_{k-1} i v_k to szumy. Jeśli oba szumy byłyby z rozkładu normalnego, to dla liniowych funkcji g i h problem można rozwiązać za pomocą filtru Kalmana. Aby zyskać na ogólności, system można opisać w następujący sposób:

$$\begin{aligned} x_k &= g(x_{k-1}, w_{k-1}) \\ y_k &= h(x_k, v_k) \end{aligned} \tag{2.1}$$

Taki problem, gdy funkcje g i h mogą być nieliniowe względem x_k, y_k oraz szumów w_{k-1} i v_k, zazwyczaj jest zbyt złożony, aby można było zastosować podejścia oparte o filtr Kalmana. W takich przypadkach jedną z możliwości jest zastosowanie filtrów cząsteczkowych. Warto zauważyć, iż w funkcji g może być zaszyta informacja o sterowaniu.

2.2 Matematyczny opis filtracji Bayesowskiej

Ogólne wzory na filtrowanie Bayesowskie można znaleźć na przykład w wikipedi [23]. Celem pojedynczego kroku filtracji jest wyznaczenie rozkładu a posteriori $p(x_k|y_{0\ldots k})$ opisującego prawdopodobieństwo, że system jest w stanie x_k, przy założeniu historii pomiarów $y_{0\ldots k}$, na podstawie rozkładu a priori $p(x_{k-1}|y_{0\ldots k-1})$ uzyskanego w poprzednim kroku oraz nowego pomiaru y_k. Etap predykcji można opisać poniższym równaniem:

$$p(x_k|y_{0\ldots k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{0\ldots k-1})dx_{k-1} \tag{2.2}$$

gdzie $p(x_k|x_{k-1})$ opisuje ewolucję systemu między krokami $k-1$ i k. Etap korekcji realizuje się w oparciu o wzór Bayesa, po otrzymaniu kolejnego pomiaru y_k:

$$p(x_k|y_{0\ldots k}) = \frac{p(y_k|x_k)p(x_k|y_{0\ldots k-1})}{p(y_k|y_{0\ldots k-1})} \tag{2.3}$$

gdzie $p(y_k|x_k)$ opisuje rozkład prawdopodobieństwa uzyskania pomiaru y_k w stanie x_k. Warto zauważyć, iż $p(y_k|y_{0\ldots k})$ można opisać wzorem:

$$p(y_k|y_{0\ldots k}) = \int p(y_k|x_k)p(x_k|y_{0\ldots k-1})dx_k \tag{2.4}$$

więc mianownik równania (2.3) jest stały względem x_k. Mając to na uwadze, oraz podstawiając równanie (2.2) do (2.3), można uzyskać ogólną formułę jednego kroku filtracji:

$$p(x_k|y_{0\ldots k}) = \mu p(y_k|x_k) \int p(x_k|x_{k-1})p(x_{t-1}|y_{0\ldots t-1})dx_{t-1} \tag{2.5}$$

gdzie μ jest stałą normalizacyjną, pozostałą po przyjęciu że $p(y_k|y_{0\ldots k})$ ma stałą wartość.

2.3 Reprezentacja rozkładu prawdopodobieństwa

Aby móc w praktyce zrealizować filtr cząsteczkowy, konieczne jest znalezienie sposobu na cyfrową reprezentację ciągłego rozkładu prawdopodobieństwa związanego ze stanem systemu. Robi się to przybliżając go za pomocą skończonego zbioru cząstek, gdzie każda cząstka wygląda w następujący sposób:

$$p_{i,k} = \{x_{k,i}, w_{k,i}\}$$

gdzie $x_{k,i}$ to stan w iteracji k związany z i-tą cząstką, $w_{k,i}$ to jej waga. Korzystając z takiej reprezentacji, rozkład $p(x_k|y_{0...k})$ ze wzoru (2.2) można zapisać w następujący sposób:

$$p(x_k|y_{0...k}) \approx \sum_{i=0}^{N} w_{k,i}\delta(x_k - x_{k,i}) \tag{2.6}$$

gdzie N oznacza liczbę cząstek, a δ to delta Dirac'a. Dzięki takiej reprezentacji prawdopodobieństwo $p(x_k|y_{0...k-1})$ opisane we wzorze (2.2) zostaje zastąpione przez wagę $w_{k,i}$ dla każdej cząstki, co sprawia, że równanie (2.3) upraszcza się do:

$$w_{k,i} = w_{k-1,i}p(y_k|x_{k,i}) \tag{2.7}$$

2.4 Podstawowy algorytm - SIR

Sam skrót SIR oznacza stochastic importance resampling, i jest to standardowa realizacja filtra cząsteczkowego w praktyce. Opis algorytmu można znaleźć w [22]. Sam importance sampling polega na wykorzystaniu właściwości jednego rozkładu prawdopodobieństwa, do próbkowania z drugiego. W tym przypadku, próbkujemy z rozkładu $p(x_k|y_{0...k})$ korzystając z wag uzyskanych z $p(y_k|x_k)$. Algorytm SIR przebiega w następujący sposób:

(a) Zaczyna się od zainicjowania zbioru cząstek losowymi stanami i równymi dla wszystkich cząstek wagami. Ważne jest, aby uzyskana w ten sposób populacja była w stanie odpowiednio oddać właściwości niezdyskretyzowanego rozkładu.

(b) Symuluje się ewolucję cząstek $x_{k-1,i}$, próbkując $x_{k,i}$ z rozkładu opisującego ewolucję systemu, $p(x_k|x_{k-1})$, obecnego w równaniu (2.2), a dobranego na podstawie funkcji g z równania (2.1). W praktyce ważne jest, aby rozkład $p(x_k|x_{k-1})$ uwzględniał szumy obecne w systemie, co sprowadza się do tego, że do ewolucji wprowadza dodatkowe czynniki losowe.

(c) Pojawia się nowy pomiar y_k, na podstawie którego modyfikuje się wagi poszczególnych cząstek, korzystając z równania (2.7).

(d) Wagi są normalizowane.

$$w_{k,i} = \frac{w_{k,i}}{\sum_{j=0}^{N} w_{k,j}} \tag{2.8}$$

(e) Wyznacza się estymowany stan, licząc ważoną średnią stanów wszystkich cząstek. Robi się to korzystając ze wzoru:

$$\hat{x}_k = \sum_{i=0}^{N} w_{k,i} x_{k,i} \tag{2.9}$$

(f) Przeprowadza się ponowne próbkowanie, generując nową populację cząstek z poprzedniej, korzystając z wag. W praktyce zazwyczaj nie przeprowadza się próbkowania co iterację, aby przyspieszyć obliczenia. Robi się to dopiero gdy spełnione jest pewne kryterium, którym najczęściej jest spadek efektywnej liczby cząstek, opisanej równaniem (2.10), poniżej pewnego progu.

$$N_{eff_k} = \frac{1}{\sum w_{k,i}^2} \tag{2.10}$$

Inne metody na określenie momentu próbkowania, można znaleźć na przykład w [8].

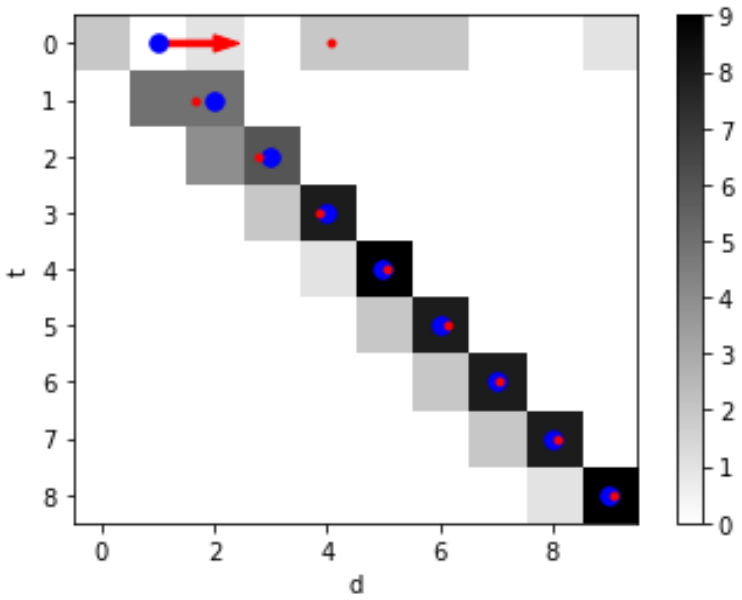
2.5 Adaptacja do wyznaczania lokalizacji

Rozwiązanie problemu wyznaczania lokalizacji za pomocą filtra cząsteczkowego często jest nazywane lokalizacją Monte Carlo. Celem jest wtedy wyznaczenie pozycji i orientacji pewnego obiektu, na przykład robota. W takim przypadku rozkład $p(x_k|x_{k-1})$ opisuje model ruchu obiektu, natomiast pomiary y_k są zebranymi danymi sensorycznymi. Prawdopodobieństwo $p(y_k|x_{k,i})$ wyznacza się na podstawie znanej mapy otoczenia.

2.6 Przykład

Najprostszym przykładem problemu lokalizacji, na którym można zademonstrować działanie filtra, jest ustalanie pozycji punktowego robota, przemieszczającego się ze stałą prędkością w kierunku ściany, mierząc swoją odległość od niej. Mapa w tym przypadku jest jednowymiarowa, a stan robota można opisać jedną

liczbą zmiennoprzecinkową, mówiącą o jego pozycji. Pomiar y_k jest odległością od robota do ściany na końcu drogi, natomiast jako rozkład $p(y_k|x_{k,i})$ przyjęto rozkład Gaussa o wartości oczekiwanej równej faktycznej odległości i wariancji 1. Na rysunku (2.2) przedstawiono efekt realizacji filtra cząsteczkowego. W tym przykładzie próbkowanie przeprowadzano na koniec każdej iteracji.



Rysunek 2.2: Przykładowa realizacja prostego filtra. Czerwona strzałka wskazuje kierunek ruchu robota, czerwone punkty pokazują estymowaną lokalizację robota, natomiast niebieskie punkty prawdziwe położenie. W kolejnych rzędach zestawiono histogramy położenia cząstek w poszczególnych iteracjach. Populacja składała się z 10 osobników.

2.7 Adaptacyjna liczba cząstek

Oczywistym jest, że największy wpływ na złożoność obliczeniową filtra cząsteczkowego ma ilość cząstek. W [7] opisano algorytm, który służy do dynamicznego zmieniania liczby cząstek. Polega on na zmodyfikowaniu kroku (f) podstawowego algorytmu, gdzie tuż przed próbkowaniem wyznacza się nową liczbę cząstek. Wykonuje się to, kolejno usuwając cząstki, i sprawdzając o ile pogorszyła się jakość estymacji. Miarę tego pogorszenia mierzy się za pomocą funkcji opisanej poniższym równaniem

$$\xi_t = |y_t - h(\hat{x_k})| \tag{2.11}$$

i wyznacza się ją za każdym razem, gdy usunie się cząstkę.

Algorytm 1: Algorytm dynamicznego doboru liczby cząstek.

Inicjalizacja $S_k = \{1, ..., N_k\}$ i $K_k = \emptyset$
Wyznaczenie $\xi(N_k)$
————————-Testowe usuwanie cząstek
for $d = 0$ **to** $dim(x_{k,i})$ **do**
  Sortowanie indeksów cząstek S względem wymiaru d
  for $i \in S$ **do**
    if $||x^d_{k,i} - x^d_{k,i+1}|| < \gamma_d$ **then**

$$K_t = \begin{cases} K_t \cup \{i+1\} \text{ jeśli } w_{k,i} > w_{k,i+1} \\ K_t \cup \{i\} \text{ w przeciwnym wypadku} \end{cases}$$

      $S_t = S_t / K_t, n_s = dim\{S_t\}$
      $\hat{x_k} = \sum_{j \in S} w_{k,j} x_{k,j}$
      Wyznaczenie $\xi(n_s)$ ze wzoru (2.11)
    end
  end
end
————————-Wyznaczanie nowego N_k
if $\xi_k(n) > \alpha, \forall n \in [n_s, N_k]$ **then**
  $N_{k+1} \sim U(N_t, N_{max})$
else
  $N_{k+1} = argmin_n\{\xi(n)\}$ taki że $\xi(n) < \alpha$
end
if $N_{k+1} < N_{min}$ **then**
  $N_{k+1} = N_{min}$
end

W algorytmie (1) wydzielono dwie sekcje, testowe usuwanie cząstek oraz wyznaczanie nowego N_k. W pierwszym etapie wyznacza się wartości $\xi(n_s)$ dla różnych rozmiarów populacji n_s, kolejno ujmując cząstki, które leżą zbyt blisko siebie. Minimalna odległość na danym wymiarze d, po przekroczeniu której usuwa się jedną z cząstek, jest określana przez parametr γ_d. W drugim etapie wyznacza się nowy rozmiar populacji N_{k+1} w zależności od ξ_k. Dopuszczalny błąd wprowadzany przez zmniejszenie populacji jest określany przez α.

2.8 Box Particle Filter (BPF)

W podejściu zaproponowanym w [6] i udoskonalonym w [17] zmieniono sposób reprezentacji cząstek. Zamiast punktów w przestrzeni stanów zastosowano

interwały, w celu poprawy jakości estymacji w sytuacjach, gdy pomiary są zbyt szybkozmienne. Na przykład, dla problemu opisanego w podrozdziale (2.6), zamiast liczby rzeczywistej opisującej położenie robota, mielibyśmy odcinek na którym może się znajdować. Ponieważ zmianie ulega reprezentacja cząstek, wszystkie kroki algorytmu opisanego w rozdziale (2.4) muszą zostać odpowiednio dostosowane.

- W kroku (a), cząsteczki interwałowe są inicjalizowane w taki sposób, aby pokrywać całą dopuszczalną przestrzeń stanów i nie nachodzić na siebie nawzajem.

- Ewolucja systemu opisana w kroku (b) zostaje zastąpiona przez przekształcenie interwałów. Wykorzystuje się do tego funkcję interwałową $[g]$ konstruowaną na podstawie funkcji g z równania (2.1). Ważne jest, aby, w przeciwieństwie do operacji z kroku (b), $[g]$ dawała w pełni deterministyczne wyniki, uwzględniające szumy obecne w systemie. Na przykład w problemie opisanym w rozdziale (2.6), jeśli cząstka byłaby interwałem [2,3] i miałaby być przesunięta o 1 ± 0.1, to korzystając z reguły 3σ [19] można skonstruować nowy interwał po przesunięciu [2.9,4.1]. Opisuje się to następującym wzorem:

$$[x_{k,i}] = [g]([x_{k-1,i}]) \tag{2.12}$$

- Zamiast pomiaru z kroku (c) wykorzystuje się interwał pomiarowy, uwzględniający jego niepewność (interwał ten można skonstruować według reguły 3σ), oraz konstruując funkcję $[h]$, na podstawie funkcji h z równania (2.1), generującą interwał pomiaru $[z]$ ze stanu danej cząstki. Nowe wagi ustala się na podstawie poniższych wzorów.

$$\begin{aligned} [z_{k,i}] &= [h]([x_{k-1,i}]) \\ [r_{k,i}] &= [z_{k,i}] \cap [y_k] \\ w_{k,i} &= \frac{|[r_{k,i}]|}{|[z_{k,i}]|} w_{k-1,i} \end{aligned} \tag{2.13}$$

  gdzie $[z_{k,i}]$ jest przewidywanym interwałem pomiaru dla cząstki i w kroku k, $[y_k]$ jest interwałem faktycznie zmierzonego pomiaru, a $|\bullet|$ jest objętością interwału (dla przykładu z rozdziału (2.6) byłaby to długość odcinka).

- Wagi są normalizowane tak samo jak w punkcie (d) podstawowego algorytmu.

- Przed wyliczeniem estymowanego stanu można przeprowadzić zawężenie interwałów. Polega ono na pomniejszeniu $[x_{k,i}]$ w oparciu o interwał $[r_{k,i}]$.

W przykładzie z rozdziału (2.6), mogło by to wyglądać w następujący sposób: jeśli uzyskano pomiar [7,9], który można uzyskać tylko w interwale [4,6], to cząstka z interwałem [3,5] zostanie zawężona do [4,5].

- Estymacja stanu odbywa się z wykorzystaniem średniej ważonej centrów interwałów (dla przykładu z rozdziału (2.6) byłby to środek odcinka).

$$\hat{x_k} = \sum_{i=0}^{N} w_{k,i} C_{k,i} \tag{2.14}$$

gdzie $C_{k,i}$ jest centrum i-tej cząstki w k-tej iteracji. . Robi się to korzystając ze wzoru

$$\theta = \sqrt{\frac{\sum_{i=0}^{N} |[x_{k,i}]|}{N|P_{union}|}} \tag{2.15}$$

gdzie P_{union} jest najmniejszym interwałem, który zawiera wszystkie interwały $[x_{k,i}]$. Gdy θ jest większe niż pewien próg, populacja jest inicjalizowana na nowo równomiernie dzieląc P_{union}.

- Ponowne próbkowanie przebiega zupełnie inaczej niż w kroku (f) podstawowego algorytmu. Sam moment ponownego próbkowania można nadal dobierać według wzoru (2.10), a samo próbkowanie zostaje zastąpione podziałem wybranych cząstek. Dla przykładu, gdy dana cząstka po próbkowaniu powinna być wzięta pięć razy, zostaje ona pięć razy podzielona wzdłuż najdłuższego wymiaru (może też być dzielona wzdłuż losowo wybranego wymiaru).

- Jednym z najważniejszych usprawnień zaproponowanych w [17] jest reinicjalizacja cząstek, w przypadku gdy suma wag spadnie do zera. Jest to konieczne, ponieważ w takim przypadku niemożliwe staje się próbkowanie.

2.9 Wprowadzenie operatora mutacji

W [13] zaproponowano, aby wzbogacić proces próbkowania z kroku (f) podstawowego algorytmu o operatory krzyżowania i mutacji. W kontekście wyznaczania lokalizacji, krzyżowanie można zaimplementować jako liniową interpolację dwóch losowo wybranych cząstek, natomiast mutację jako niewielkie zaszumienie cząstek. Pozwala to na poprawę działania filtra przy bardzo powolnej ewolucji, bądź nawet jej braku.

Rozdział 3

Przygotowane oprogramowanie

W tym rozdziale przedstawiono zastosowane narzędzia programistyczne, a następnie opisano przygotowane oprogramowanie.

3.1 Wybrane narzędzia programistyczne

Najważniejszym kryterium branym pod uwagę przy wyborze języka programowania była wydajność. Jeśli chodzi o IDE(integrated development environment) wystarczyło jeśli dało się w nim pracować z wybranymi językami. Kierując się wydajnością, jako język do implementacji algorytmów wybrano C++, jednak ze względu na brak prostego sposobu graficznej prezentacji wyników, pomocniczo skorzystano z języka Python i jego bibliotek numpy [10] i matplotlib [12]. Aby połączyć oba języki skorzystano z biblioteki pybind11 [14], która umożliwia proste tworzenie bibliotek dla Pythona napisanych w C++. W konsekwencji pociągnęło to za sobą wybór IDE - edytora tekstu Sublime Text [5], ponieważ częsta instalacja biblioteki wymuszała ciągłe korzystanie z wiersza poleceń. Aby ułatwić zarządzanie środowiskami Pythona, oraz samą z nim pracę, wykorzystano pakiet Anaconda [1]. W celu zaimplementowania algorytmu Box Particle Filter skorzystano z biblioteki interval będącej częścią zbioru bibliotek Boost [2].

3.2 Opis przygotowanej biblioteki

Opracowana biblioteka nosi nazwę PFlib. Została ona zorganizowana jako zbiór funkcji umożliwiających zarządzanie tablicami stanów, gdzie stany były prostymi strukturami danych, oraz kilkoma klasami: klasą implementującą Box Particle Filter, oraz klasami realizującymi mapy potrzebne do badań. Poniżej ogólnie przedstawiono ich opis (dokładniejsze informacje można uzyskać analizując kod):

- robot_2d - struktura służąca za kontener na stan robota opisany w rozdziale (4.1.1).
- roulette_wheel_resample - funkcja realizująca próbkowanie ruletkowe opisane w [18] w oparciu o wagi.
- sus_resample - funkcja realizująca próbkowanie o niskiej wariancji opisane w [24] w oparciu o wagi.
- as_array - funkcja konwertująca tablicę stanów na macierz.
- get_uniform_weights - funkcja inicjalizująca tablicę równych sobie wag.
- get_est - funkcja wyznaczająca średnią ważoną stanów dla danej tablicy stanów i wag.
- update_weights - funkcja przeliczająca wagi z wykorzystaniem nowego pomiaru, korzystając ze wzoru (2.7).
- drift_state - funkcja przeprowadzająca ewolucję pojedynczego stanu.
- drift_pop - funkcja przeprowadzająca ewolucję tablicy stanów.
- get_random_pop - funkcja inicjalizująca losową populację.
- get_linear_pop - funkcja inicjalizująca populację równo rozmieszczoną wzdłuż jednaj osi.
- get_new_N - funkcja wyznaczająca nowy rozmiar populacji korzystając z algorytmu (1).
- regularize - funkcja implementująca podejście opisane w rozdziale (2.9).
- BoxParticleFilter - klasa implementująca podejście opisane w rozdziale (2.8), posiadająca następujące metody:
  - init_pop - metoda inicjalizująca populację interwałowych cząstek.
  - reinit_pop - metoda reinicjalizująca populację.
  - update_weights - metoda przeliczająca wagi z wykorzystaniem nowego pomiaru, korzystająca ze wzoru (2.7).
  - drift - metoda przeprwadzająca ewolucję interwałowych cząstek.
  - get_est - metoda zwracająca estymowany stan, według wzoru (2.14)
  - resample - metoda przeprowadzająca próbkowanie o niskiej wariancji [24].

  - get_coeff - metoda zwracająca współczynnik wyznaczony ze wzoru (2.15)
  - get_pop - funkcja zwracająca macierz zawierającą reprezentacje interwałowych cząstek.

- PrimitiveMap - klasa realizująca mapę potrzebną do problemu opisanego w rozdziale (4.1.1), posiadająca następujące metody:
  - get_grid - metoda zwracająca macierz będącą zero-jedynkową reprezentacją mapy (1 - komórka zajęta, 0 - komórka wolna).
  - add_line - metoda pozwalająca ograniczyć mapę prostą.
  - add_circle - metoda pozwalająca dodać przeszkodę w kształcie koła.
  - get_meas - metoda pozwalająca uzyskać pomiar na zadanych współrzędnych w danej pozycji.

- HeightMap - klasa realizująca mapę potrzebną do problemu opisanego w rozdziale (4.1.2), posiadająca następujące metody:
  - get_grid - metoda zwracająca macierz będącą mapą wysokościową.
  - get_meas - metoda pozwalająca uzyskać pomiar na zadanych współrzędnych.
  - get_meas_prob - metoda zwracająca prawdopodobieństwo danego pomiaru.

Rozdział 4

Badania symulacyjne

W tym rozdziale opisano przeprowadzone badania symulacyjne. Zaczęto od opisania konkretnych problemów którymi się zajmowano, a następnie przeprowadzono badania mające potwierdzić czy algorytmy działają poprawnie. Na koniec przebadano zachowanie zaimplementowanych filtrów pod różnymi kątami.

4.1 Opis poszczególnych problemów

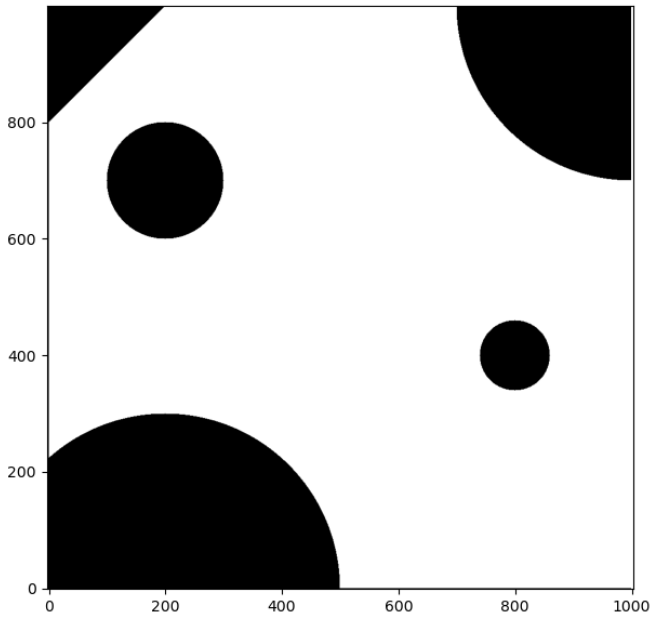
W badaniach zajęto się dwoma problemami wyznaczania lokalizacji. Pierwszy polega na ustaleniu pozycji robota na podstawie pomiaru odległości od ściany, drugi na ustaleniu pozycji samolotu na podstawie pomiaru wysokości. W obu przypadkach znano mapę otoczenia w którym znajdowały się obiekty. Faktyczna pozycja robota była zawsze oznaczana przez czerwoną kropkę, estymowana pozycja przez żółtą, natomiast cząstki oznaczano na niebiesko.

4.1.1 Robot w pomieszczeniu

Stan robota w pomieszczeniu opisano czterema liczbami:

$$x = \{p_x, p_y, \theta, v\}$$

gdzie p_x i p_y to pozycja robota, θ jest jego orientacją, natomiast v prędkością. Pomiar odległości od ściany był zawsze wykonywany w kierunku θ, i miał Gaussowski rozkład. Mapa jest kwadratowym pokojem o wymiarach 1000 na 1000 jednostek, wypełnionym kołami o różnych średnicach, oraz ograniczony prostymi. Na rysunku (4.1) przedstawiono przykładową mapę.
O ile nie będzie napisane inaczej, robot zaczynał na środku mapy, miał prędkość 10 jednostek na iterację, i skręcał w lewo o $0.1rad$ na krok.



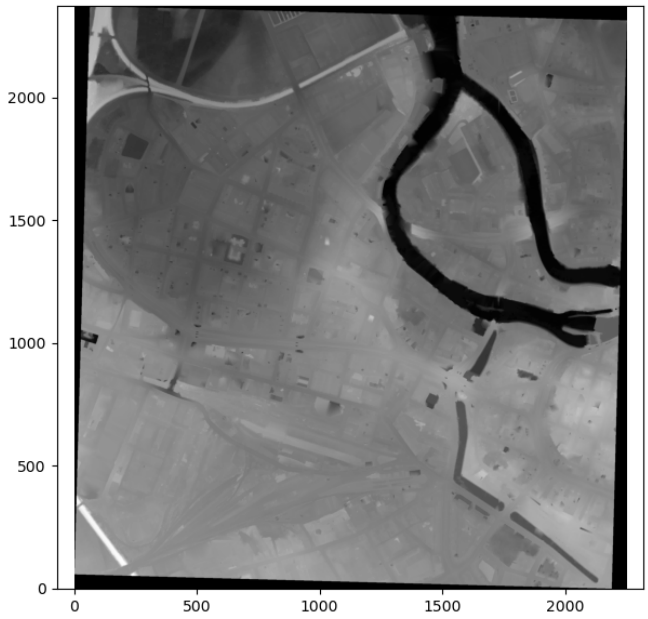
Rysunek 4.1: Przykładowa mapa pokoju

4.1.2 Samolot w locie

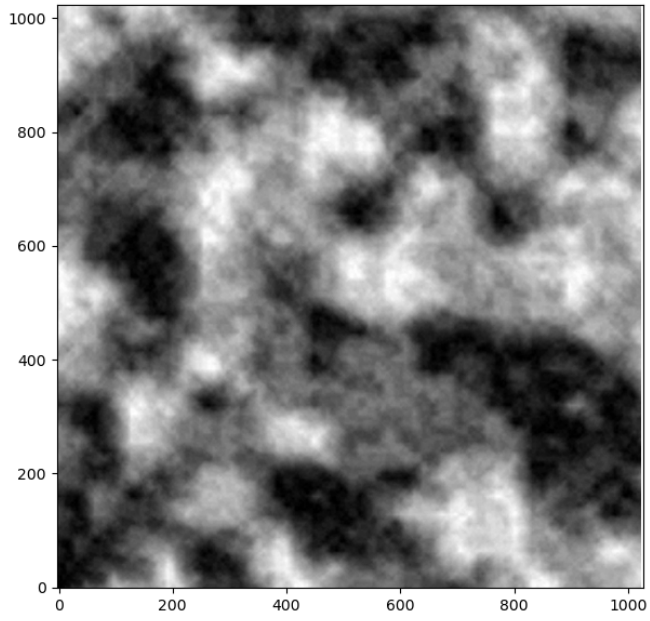
Stan samolotu opisano tak samo jak stan robota w rozdziale (4.1.1):

$$x = \{p_x, p_y, \theta, v\}$$

Mapa jest natomiast mapą wysokościową, w tym przypadku zdecydowano się na dwa warianty, pierwszy jest mapą fragmentu Wrocławia (przykład widoczny na rysunku (4.2)), natomiast drugi jest wygenerowanym szumem, którego zmienność można kontrolować (przykład widoczny na rysunku (4.3)).
O ile nie będzie napisane inaczej, samolot w lewym dolnym rogu mapy, miał prędkość 10 jednostek na iterację, i nie zmieniał orientacji $\theta = \frac{\pi}{4}$.



Rysunek 4.2: Mapa wysokościowa fragmentu Wrocławia [3]



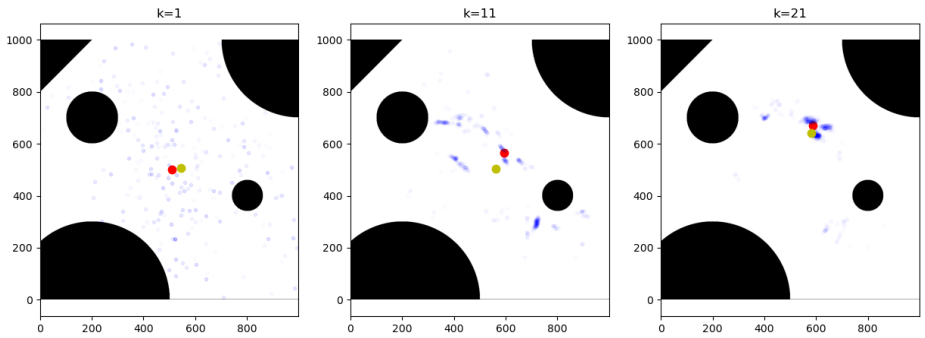
Rysunek 4.3: Przykładowa mapa wysokościowa uzyskana z szumu

4.2 Badania poprawności działania

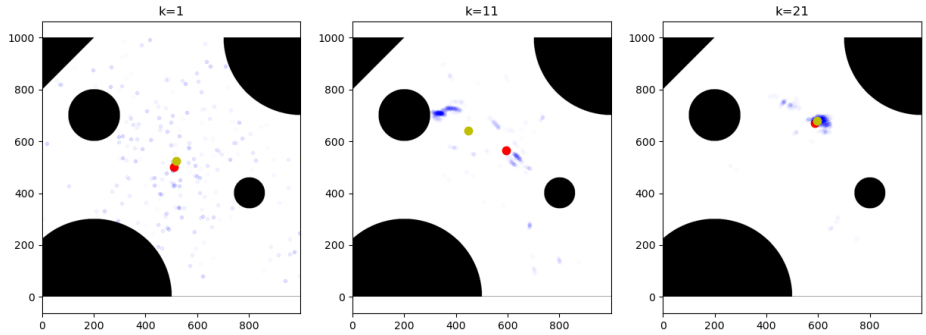
W tym rozdziale zajęto się badaniami mającymi potwierdzić poprawne działanie zaimplementowanego rozwiązania. Na początku sprawdzono, jak zmiana generatora liczb losowych wpłynęła na wyniki, następnie sprawdzono, jak poradzi sobie algorytm przy braku punktów odniesienia, potem co się dzieje przy braku ewolucji systemu ($v = 0$). Nie skupiano się na konkretnych wartościach liczbowych, jedynie wizualnie oceniano wyniki.

4.2.1 Wpływ generatora

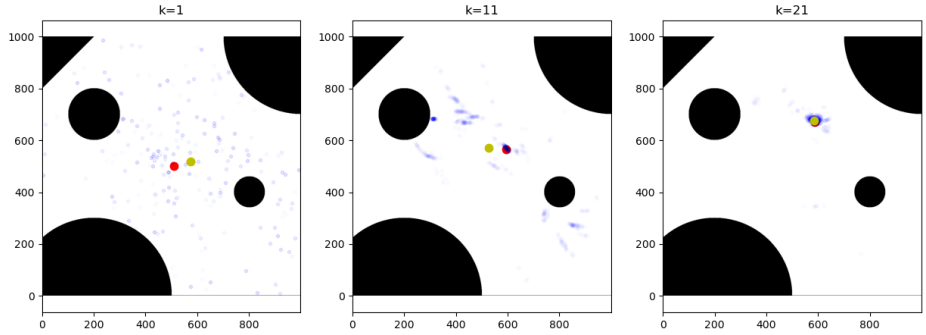
Przebadano trzy generatory wbudowane w język C++: domyślny linear congruential generator(LCG) [20], Mersenne Twister [21] oraz wbudowany niedeterministyczny generator. Populacja cząstek wynosiła $N = 1000$. Wyniki rysowano po 1, 11 i 21 iteracjach filtra(nr iteracji oznaczano jako k). Wyniki przedstawiono na rysunkach (4.4), (4.5), (4.6). Jak widać dla wszystkich generatorów wyniki są niemal takie same, i zbiegają do faktycznego położenia robota. Ponieważ dla każdego generatora uzyskano poprawne wyniki, w dalszych badaniach korzystano z generatora LCG ponieważ jest najprostszy, i, co za tym idzie, najszybszy.



Rysunek 4.4: Przykładowe wyniki dla generatora LCG



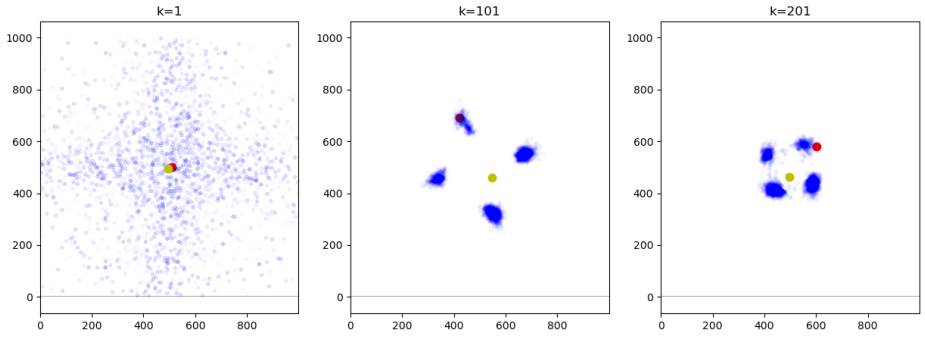
Rysunek 4.5: Przykładowe wyniki dla generatora Mersenne Twister



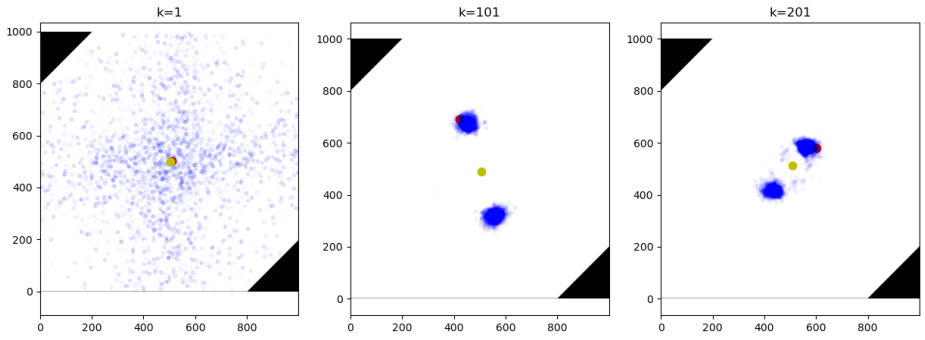
Rysunek 4.6: Przykładowe wyniki dla niedeterministycznego generatora

4.2.2 Niejednoznaczna mapa

W tym przypadku zbadano jak zachowa się filtr przy braku punktów odniesienia. W kwadratowym pokoju, powinno to spowodować pojawienie się czterech równie prawdopodobnych pozycji. Jak widać przewidywania potwierdziły się na rysunku (4.7). Na rysunku (4.8) przedstawiono sytuację, gdy mapa dopuszcza dwie możliwe pozycje. Badania przeprowadzono przy populacji $N = 10000$ cząstek.



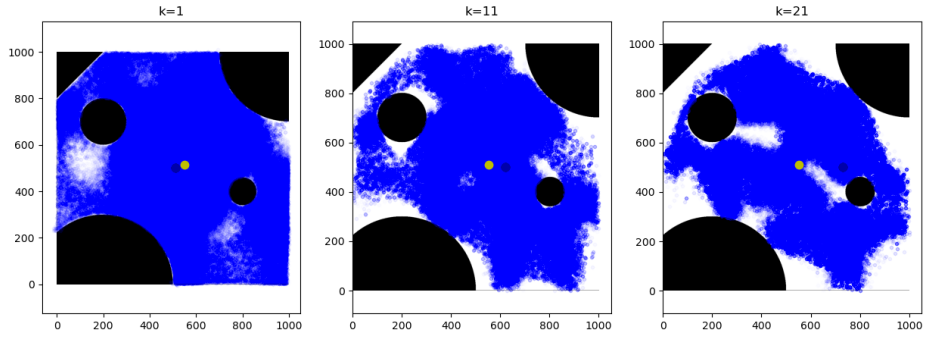
Rysunek 4.7: Przykładowe wyniki przy braku punktu odniesienia



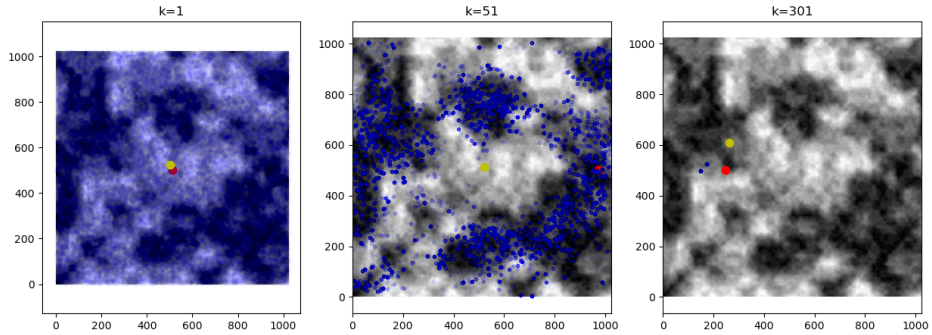
Rysunek 4.8: Przykładowe wyniki przy dwóch możliwych położeniach

4.2.3 Brak ewolucji systemu

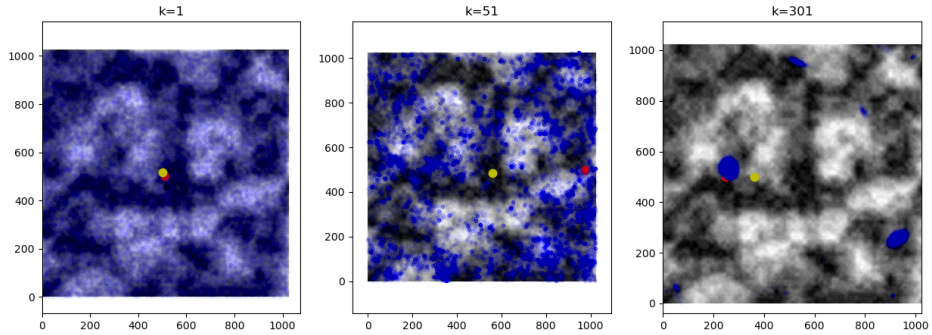
Zbadano, co się dzieje, gdy system nie ewoluuje. Spodziewano się, że pojawią się izolinie(w tym przypadku będą to krzywe/obszary dla których stała będzie wartość pomiaru), przynajmniej na początku, nim populacja nie stanie się zbyt zdegenerowana. Badania przeprowadzono dla populacji o rozmiarze $N = 100000$. Jak widać na rysunkach (4.9) i (4.10) przewidywania się spełniły, dodatkowo dla rysunku (4.10) dobrze widać problem pojawiającej się degeneracji. Problem ten można rozwiązać dzięki podejściu z rozdziału (2.9), które zademonstrowano na rysunku (4.11), na którym widać jak wychwycone zostały obszary na tej samej wysokości.



Rysunek 4.9: Przykładowe wyniki przy braku ewolucji systemu, dla robota w pokoju



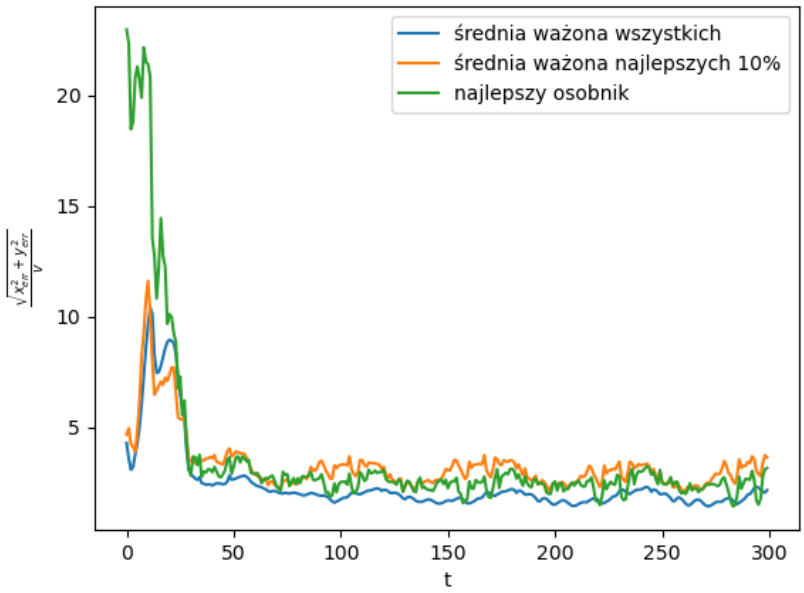
Rysunek 4.10: Przykładowe wyniki przy braku ewolucji systemu, dla samolotu



Rysunek 4.11: Przykładowe wyniki przy braku ewolucji systemu, dla samolotu, przy zmodyfikowanym algorytmie

4.3 Wpływ sposobu estymowania położenia

Przebadano trzy metody estymowania położenia: średnią ważoną cząstek, średnią ważoną 10% najlepszych cząstek, najlepszą cząstkę. Badania przeprowadzono na populacji 300 cząstek. Na wykresie (4.12) przedstawiono rezultaty z których wynika, że między średnią ważoną z najlepszych 10% i najlepszym osobnikiem nie ma większej różnicy, natomiast średnia ważona populacji jest nieco lepsza niż pozostałe dwa rozwiązania.

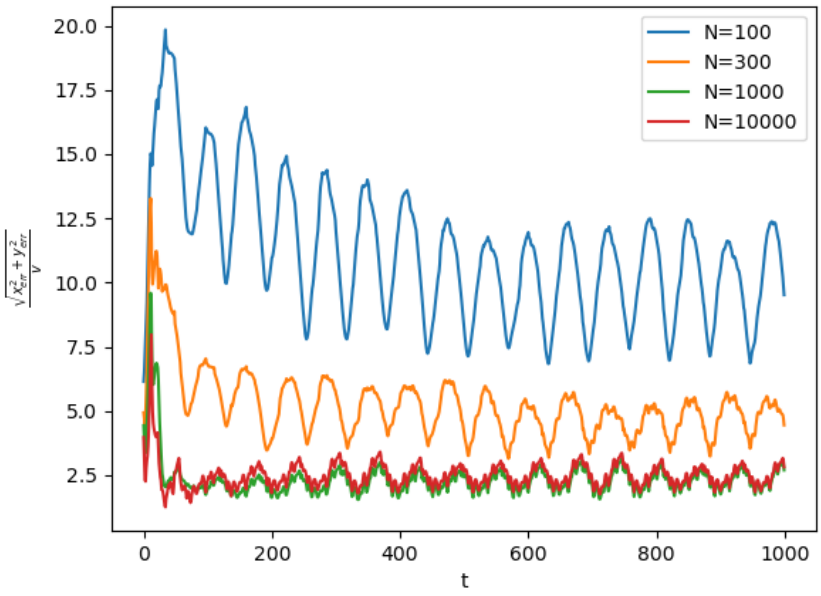


Rysunek 4.12: Przykładowe wyniki przy braku ewolucji systemu, dla samolotu

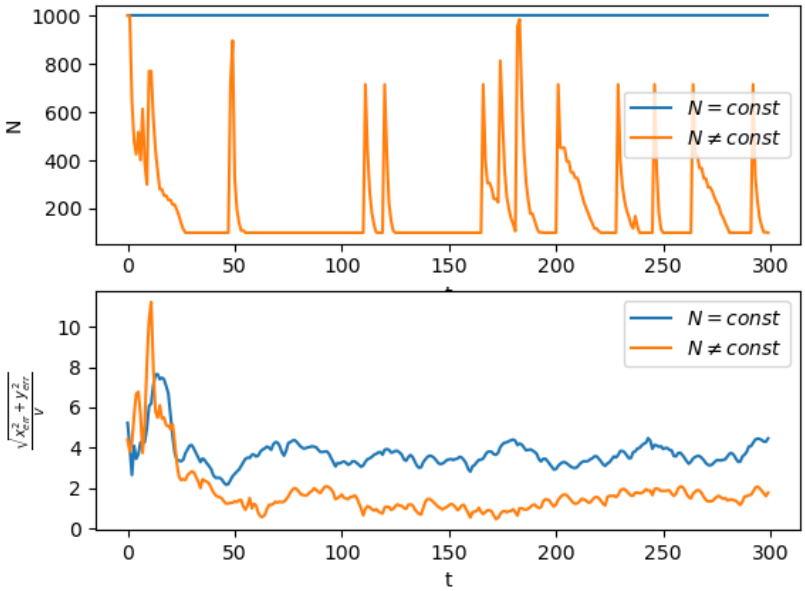
4.4 Wpływ liczby cząstek

Zbadano wpływ liczby cząstek na jakość estymacji. Badania przeprowadzono dla robota w pokoju, dla czterech różnych N: 100, 300, 1000 i 10000, wyniki uśredniając po 100 razy. Rezultaty zestawiono na rysunku (4.13). Jako błąd przyjęto pierwiastek sumy kwadratów błędów w kierunkach x i y, podzielony przez prędkość rzeczywistego robota. Jak widać, wraz ze wzrostem cząstek poprawia się jakość estymacji, jednak dzieje się tak do pewnego momentu, po którym zwiększenie liczb cząstek nie poprawia jakości estymacji (tutaj wyniosła ona około 1000 cząstek). Na wykresie (4.14) widać jak zmieniała się liczba

cząstek, gdy zastosowano podejście opisane w rozdziale (2.7). Wartości parametrów ustalono następująco: $N_{min} = 100, N_{max} = 1000, \alpha = 50, \gamma = 1$. Jak widać nie tracąc za bardzo na jakości estymacji można znacząco zredukować liczbę cząstek.



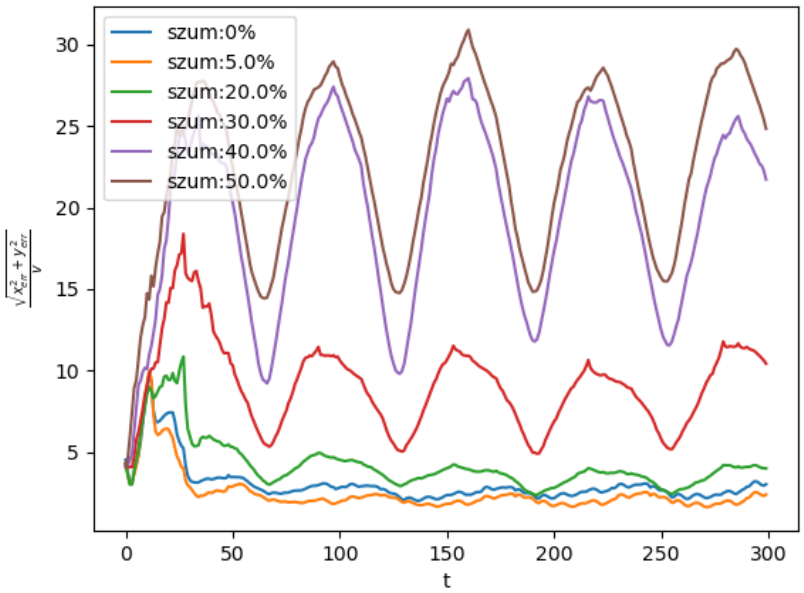
Rysunek 4.13: Wpływ liczby cząstek na jakość estymacji dla robota w pokoju



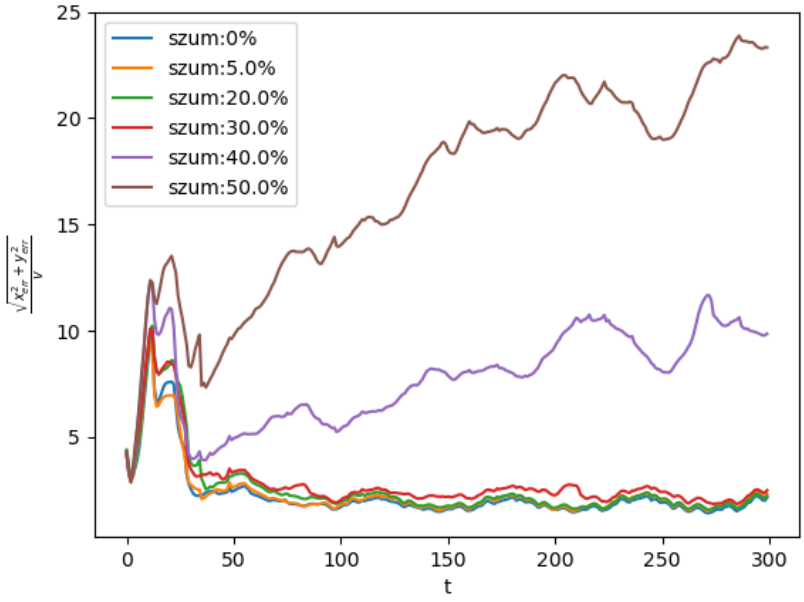
Rysunek 4.14: Wpływ dynamicznej zmiany liczby cząstek na jakość estymacji dla robota w pokoju

4.5 Wpływ szumu

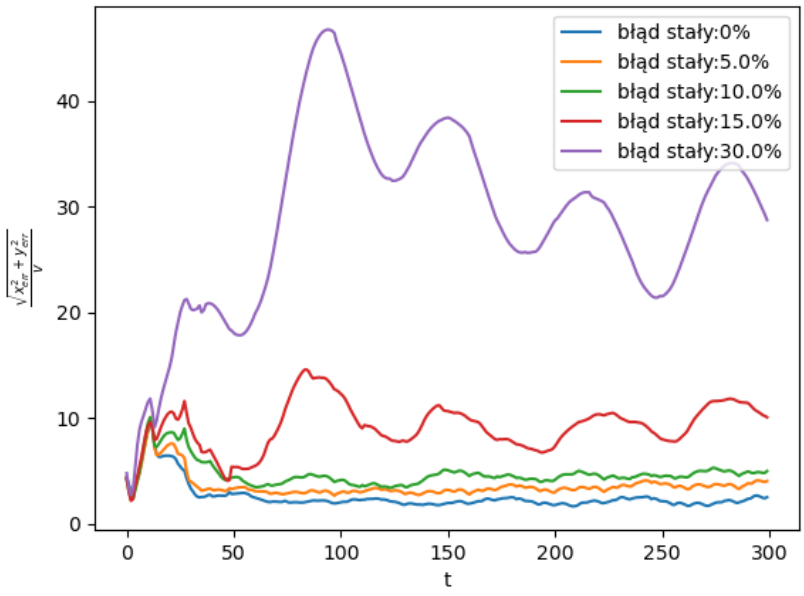
W tym rozdziale zbadano wpływ szumu na jakość estymacji. Przebadano trzy rodzaje zaszumienia: zaszumienie pobieranego pomiaru (rysunek (4.15)), zaszumienie sterowania (kąta o jaki robot skręca, rysunek (4.16)), oraz zbadano problem gdy pojawia się błąd systematyczny, stały dla każdej iteracji (rysunek (4.17)). Jak widać, zaszumienie pomiaru da się skompensować, przynajmniej dopóki nie przekroczy ono około 20% wartości pomiaru, zaś, jeśli chodzi o systematyczny błąd, próg ten wynosi około 10%. W przypadku zaszumienia sygnału sterującego kątem robota, filtr jest w stanie z powodzeniem kompensować szum na poziome 30%.



Rysunek 4.15: Wpływ zaszumienia pomiaru na jakość estymacji dla robota w pokoju



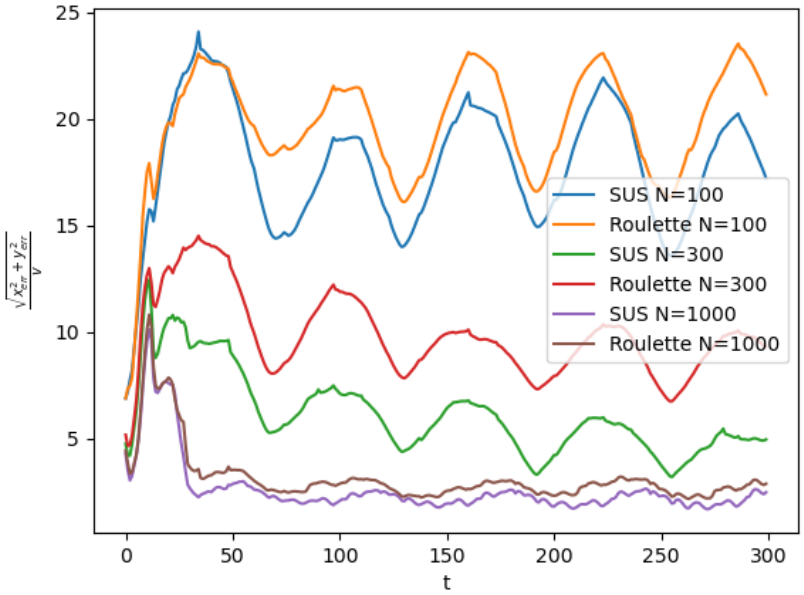
Rysunek 4.16: Wpływ zaszumienia odczytu zmiany orientacji na jakość estymacji dla robota w pokoju



Rysunek 4.17: Wpływ pojawienia się systematycznego błędu pomiaru na jakość estymacji dla robota w pokoju

4.6 Wpływ metody próbkowania

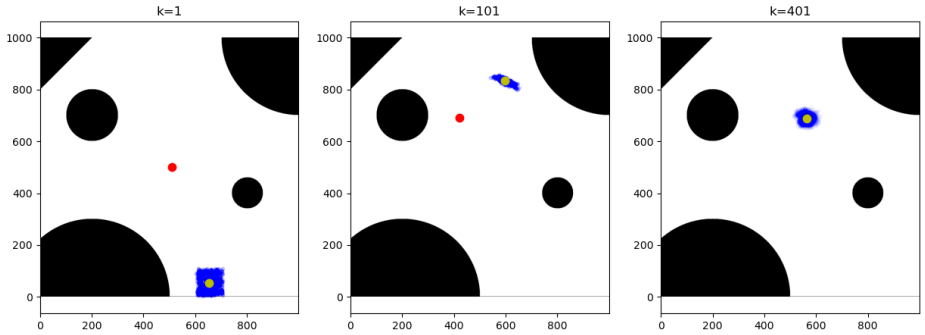
Przebadano dwie metody próbkowania: najpopularniejszy stochastic universal sampling [24] (SUS), z którego korzysta się ze względu na niską złożoność obliczeniową, oraz roulette sampling [18]. Badania przeprowadzono dla trzech różnych rozmiarów populacji: 100,300,1000, wyniki uśredniając po 100 razy. Jak widać, SUS daje zazwyczaj lepsze wyniki, zwłaszcza dla niewielkich populacji.



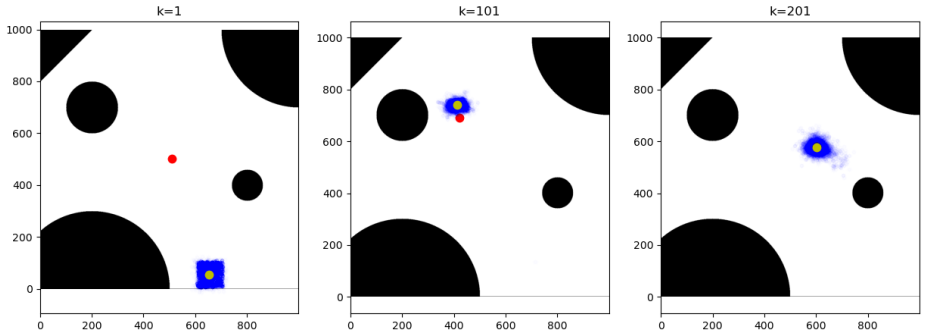
Rysunek 4.18: Wpływ metody próbkowania cząstek na jakość estymacji.

4.7 Skuteczność w poprawianiu błędnie określonego położenia

Sprawdzono zachowanie filtra w sytuacji, gdy w jakiś sposób nie poradzi sobie z określeniem położenia i zgubi się. Wyniki widać na rysunku (4.19). Można zaobserwować, iż algorytm jest w stanie odszukać faktyczny stan, jednak jest to bardziej kwestia szczęścia. Podejście opisane w rozdziale (2.9) jest w stanie delikatnie poprawić rozwiązanie, jednak nie jest ono tak skuteczne jak przeszukanie od nowa całej przestrzeni stanów jak to się robi w przypadku algorytmu opisanego w rozdziale (2.8).



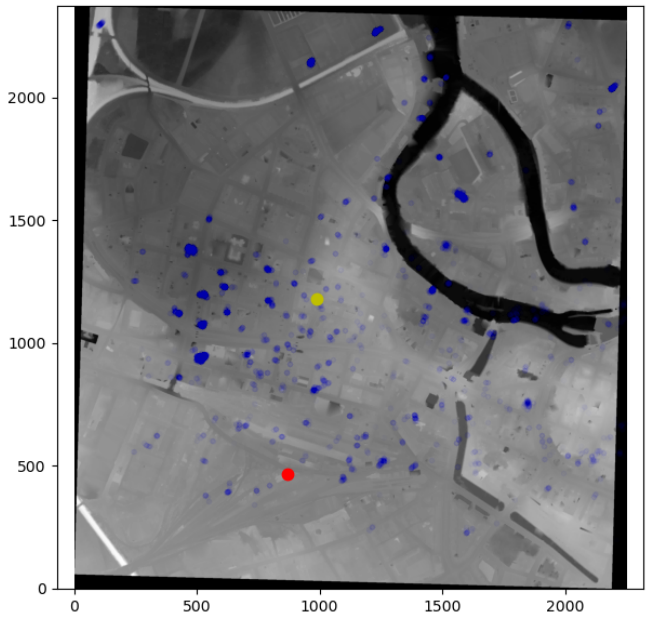
Rysunek 4.19: Sprawdzenie jak filtr radzi sobie w sytuacji gdy nie ma cząstek w pobliżu faktycznego stanu



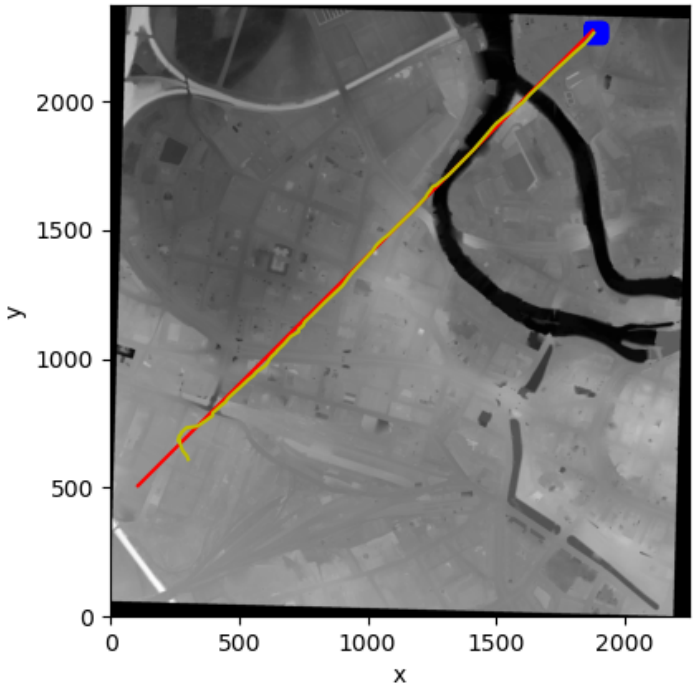
Rysunek 4.20: Sprawdzenie jak zmodyfikowany filtr radzi sobie w sytuacji gdy nie ma cząstek w pobliżu faktycznego stanu

4.8 Wpływ zmienności mapy

W problemie robota w pokoju, pomiary, które może zbierać robot, zazwyczaj nie zmieniają się znacząco, przy niewielkich zmianach stanu. Zupełnie inaczej jest w problemie samolotu w locie, gdzie przesunięcie o 10 jednostek daje w zasadzie losowe pomiary. Na rysunku (4.21) widać przykładową realizację podstawowego algorytmu dla tego problemu. Mimo iż liczba cząstek wyniosła $N = 10000$ algorytm nie jest w stanie dokładnie określić położenia. Na rysunku (4.22) zaprezentowano jak BPF opisany w rozdziale (2.8) radzi sobie z tym problemem, liczba cząstek wyniosła $N = 961$. Jak widać, wyniki są nieporównywalnie lepsze niż w przypadku podstawowego algorytmu.



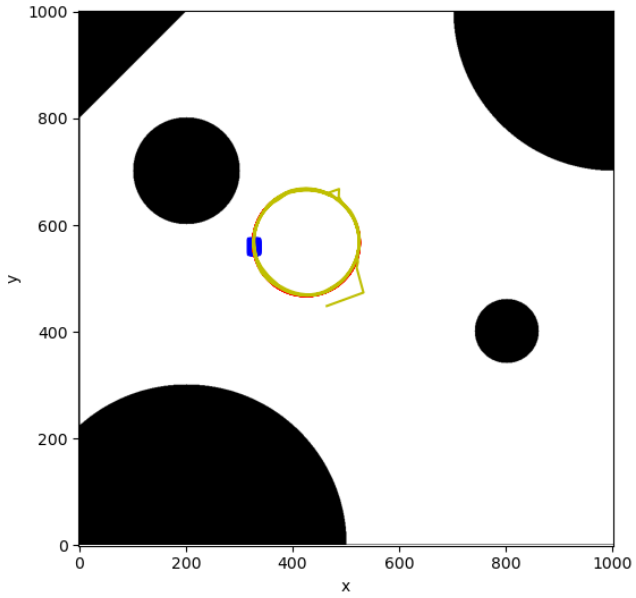
Rysunek 4.21: Przykład zastosowania podstawowego algorytmu do problemu samolotu w locie



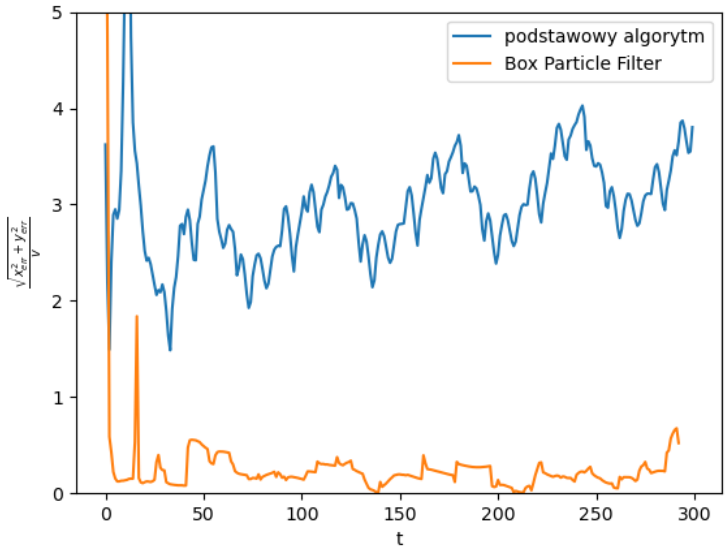
Rysunek 4.22: Przykład zastosowania BPF do problemu samolotu w locie

4.9 Zastosowanie BPF do problemu robota w pokoju

Na koniec podjęto próbę zastosowania BPF do problemu robota w pokoju. Przykład realizacji jest widoczny na rysunku (4.23). Jak widać, wyniki są bardzo obiecujące, zwłaszcza, gdy spojrzy się na rysunek (4.24), gdzie porównano BPF z podstawowym algorytmem (na pojedynczym uruchomieniu), a na którym widać, iż BPF daje o wiele lepsze wyniki. Należy zauważyć, że podczas uruchamiania BPF dawał zazwyczaj takie same wyniki dla wszystkich uruchomień, natomiast podstawowy algorytm znacząco zmieniał się za każdym razem.



Rysunek 4.23: Przykład zastosowania BPF do problemu robota w pokoju



Rysunek 4.24: Porównanie BPF i podstawowej wersji dla problemu robota w pokoju

Rozdział 5

Podsumowanie

W tym rozdziale przedstawiono wnioski oraz wskazano możliwe kierunki przyszłych badań.

5.1 Wnioski

W ramach pracy udało się zrealizować postawiony we wstępie cel. Poniżej zestawiono wnioski, które wysnuto po ukończeniu pracy.

- Filtry cząsteczkowe są szeroko wykorzystywanym i wciąż badanym narzędziem. Pozwalają one na radzenie sobie z problemami, gdy zarówno sam system jak i jego zależność od szumów jest wysoce nieliniowa. Ponadto, ponieważ są one w stanie, do pewnego stopnia, radzić sobie z systematycznymi błędami(rozdział (4.5)), to mogą korygować błędy popełnione przy projektowaniu modelu.

- Filtry cząsteczkowe doskonale nadają się do rozwiązywania problemów lokalizacji, przy znajomości mapy. Zawdzięczają to głównie swojej prostocie, ponieważ mając model systemu, w zasadzie jedynym parametrem jaki trzeba ustalić jest liczba cząstek, którą można łatwo wyznaczyć w oparciu o wydajność urządzenia, na którym uruchamia się algorytm, albo stosując podejście opisane w rozdziale (2.7).

- Dużym problemem są sytuacje, gdy lokalizacja z początku się nie powiedzie. O ile podstawowy algorytm nie jest w stanie szybko korygować takich błędów, to usprawnienia, takie jak to opisane w rozdziale (2.9), są w stanie wyeliminować ten błąd. Innym rozwiązaniem może być ponowna inicjalizacja algorytmu, jak to ma miejsce w algorytmie BPF opisanym w rozdziale (2.8).

- Całkowita zmiana podejścia do tego, jak się modeluje się cząstki, okazała się bardzo dobrym pomysłem, gdy otrzymujemy szybkozmienne pomiary, jak to ma miejsce w problemie samolotu opisanego w rozdziale (4.1.2), i zbadanego w rozdziale (4.8).

- Przygotowane, na potrzeby pracy, oprogramowanie doskonale spełniło swoje zadanie. Dzięki połączeniu C++ i Pythona udało się w pełni skorzystać z zalet obu języków, to jest wydajności C++ i prostoty w manipulacji danymi Pythona. Jedynym problemem był czas, który należało poświęcić na integrację obu języków, a który był zbyt długi, ze względu na problematyczne usuwanie błędów implementacyjnych.

5.2 Kierunki przyszłych badań

Jak wspominano wcześniej, filtry cząsteczkowe są nadal rozwijane. Poniżej przedstawiono kilka pomysłów, które można zbadać.

- Zrezygnowanie z implementacji C++ na rzecz całkowitego przejścia na Python, z wykorzystaniem na przykład pakietu Numba [15]. Poza przyspieszeniem porównywalnym z C++, pozwoliło by to, na przykład, na poszerzanie kodu o obliczenia na karcie graficznej. Dzięki temu, można by spróbować rozwiązać problemy opisane bardziej skomplikowanymi modelami.

- Zamiast zwykłych uśrednień, można zbadać inne metody estymacji stanu. Na przykład przeprowadzić klasteryzację cząstek, a następnie uśredniać dla poszczególnych klastrów i wybierać jeden, o największej sumie wag.

Dodatek A

Zawartość załączonej płyty CD

- Praca magisterska - dokument w formacie .pdf
- Przygotowane oprogramowanie - pliki w formacie .cpp oraz setup.py
- Przygotowane demonstracje - pliki w formacie .py poza setup.py
- requirements.txt - opis środowiska Pythonowego

Bibliografia

[1] Anaconda software distribution, 2020.

[2] Boost c++ libraries, 2021. [Online; dostęp 18-czerwiec-2021].

[3] Fragment 66574_759325_m-33-35-c-a-3-1, 2021. https://geoportal.pl/ [Online; dostęp 19-czerwiec-2021].

[4] Object tracking: Particle filter with ease, 2021. [Online; dostęp 19-czerwiec-2021].

[5] Sublime text, 2021. [Online; dostęp 18-czerwiec-2021].

[6] F. Abdallah, A. Gning, P. Bonnifait. Box particle filtering for nonlinear state estimation using interval analysis. *Automatica*, 44(3):807–815, 2008.

[7] P. Closas, C. Fernández-Prades. Particle filtering with adaptive number of particles. *IEEE Aerospace Conference Proceedings*, 03 2011.

[8] P. Del Moral, A. Doucet, A. Jasra. On adaptive resampling procedures for sequential monte carlo methods. *Bernoulli*, 01 2012.

[9] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, P.-j. Nordlund. Particle filters for positioning, navigation and tracking. *Signal Processing, IEEE Transactions on*, 50:425 – 437, 03 2002.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Wrze. 2020.

[11] A. Haug. Bayesian estimation and tracking: A practical guide. *Wiley*, 06 2012.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[13] M. Hussain. *Real-coded genetic algorithm particle filters for high-dimensional state spaces*. Praca doktorska, 04 2014.

[14] W. Jakob, J. Rhinelander, D. Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

[15] S. K. Lam, A. Pitrou, S. Seibert. Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[16] F. Maurelli, A. Mallios, D. Ribas, P. Ridao, Y. Petillot. Particle filter based auv localization using imaging sonar. *IFAC Proceedings Volumes*, 42(18):52–57, 2009. 8th IFAC Conference on Manoeuvring and Control of Marine Craft.

[17] N. Merlinge, K. Dahia, H. Piet-Lahanier. A box regularized particle filter for terrain navigation with highly non-linear measurements. *IFAC-PapersOnLine*, 49, 12 2016.

[18] Wikipedia contributors. Fitness proportionate selection — Wikipedia, the free encyclopedia, 2020. [Online; dostęp 14-czerwiec-2021].

[19] Wikipedia contributors. 68–95–99.7 rule — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 13-czerwiec-2021].

[20] Wikipedia contributors. Linear congruential generator — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 13-czerwiec-2021].

[21] Wikipedia contributors. Mersenne twister — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 13-czerwiec-2021].

[22] Wikipedia contributors. Particle filter — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 14-czerwiec-2021].

[23] Wikipedia contributors. Recursive bayesian estimation — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 11-czerwiec-2021].

[24] Wikipedia contributors. Stochastic universal sampling — Wikipedia, the free encyclopedia, 2021. [Online; dostęp 14-czerwiec-2021].

Spis rysunków