Politechnika Warszawska

WYDZIAŁ MECHANICZNY
ENERGETYKI I LOTNICTWA

Instytut Techniki Lotniczej i Mechaniki Stosowanej

Dobór algorytmu do systemu planowania trajektorii obiektu na podstawie dwuwymiarowej mapy otoczenia

Michał Modzelewski

Numer albumu: 292768

Warszawa, 2022

Spis treści

1 Wstęp

W ostatnich latach zauważalny jest znaczący wzrost wykorzystywania autonomicznych robotów mobilnych do automatyzacji różnych procesów. Przykładami takich robotów mogą być: autonomiczne taksówki, roboty sortujące przesyłki, wielowirnikowce przeprowadzające windykacje w magazynach czy mobilne roboty sprzątające. Wraz ze zwiększającą się ingerencją autonomicznych maszyn mobilnych w życie ludzkie, zwraca się coraz większą uwagę na ich bezpieczeństwo poruszania, na które największy wpływ mam dobór bezkolizyjnej trajektorii. Ze względu na fakt, że każdy z robotów może być wykorzystywany do różnych prac, ważnym elementem jego działania jest wyznaczanie trajektorii i jej optymalizacja pod względem cech ważnych z punktu wiedzenia wykonywanego zadania przez robota. Najczęstszym parametrem optymalizacyjnym jest czas, który robot potrzebuje na przebycie danej trajektorii. W większości przypadków głównym czynnikiem wpływającym na czas ruchu robota jest długość trajektorii. Innym ważnym elementem doboru trajektorii jest złożoność obliczeniowa algorytmu, co ma w szczególności znaczenie, gdy robot znajduje się w dynamicznie zmieniającym się środowisku.

Celem tej pracy jest dobór algorytmu dla systemu bezkolizyjnego planowania trajektorii robota, poruszającego się w zamkniętym pomieszczeniu z przeszkodami.

2 Model obiektu

Obiekt dla którego będzie wyznaczana trajektoria to model samochodu ROSobt 2.0 [4], który jest widoczny na poniższej grafice.

Rysunek 1: Robot ROSobt 2.0

Robot nie posiada osi skrętnej, jednak układ sterowania pozwala obrót kołami w przeciwnych kierunkach oraz z różną prędkością, co sprawia, że oprócz poruszania się do przodu i tyłu, robot może również zmieniać kierunek przemieszczania. Obroty mogą być wykonywane zarówno podczas przemieszczania, jak i również, gdy robot się nie przemieszcza.

Wymiary ROSobt 2.0 to 235 mm długości (wzdłuż osi przemieszczania) 200 mm szerokości (prostopadle do osi przemieszczania) i 106 mm wysokości. Rozmiar robota w porównaniu do wymiarów pomieszczenia, w którym się znajduje, jest znikomy, w związku z tym postanowiono, że zostanie on zaniedbany podczas wyznaczania trajektorii.

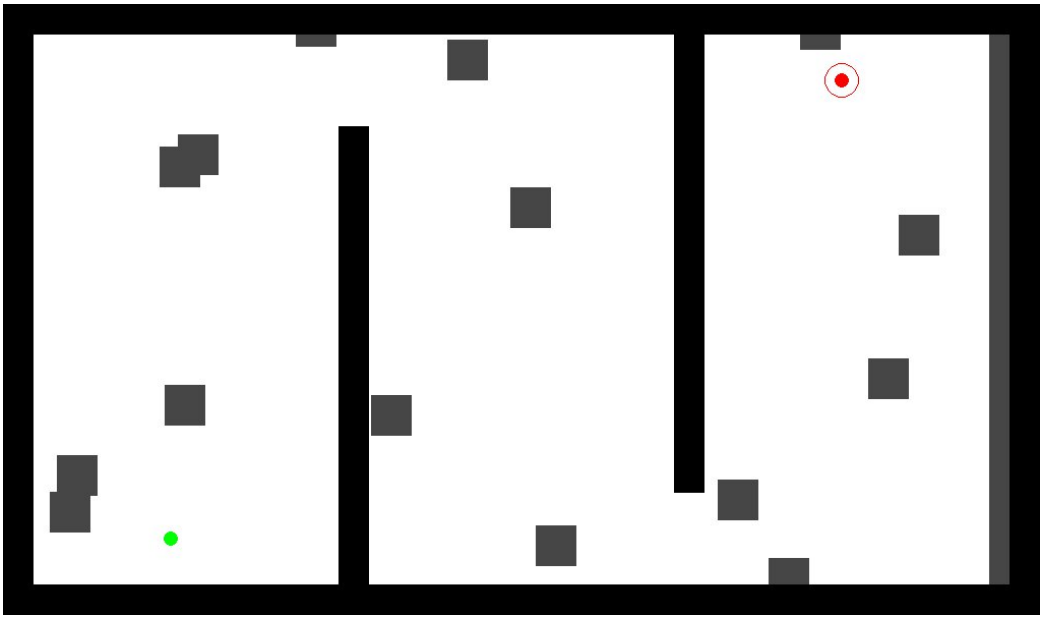
Model samochodu wyposażony jest między innymi w komputer pokładowy, który służy głównie do sterowania silnikami, przetwarzania danych z czujników oraz nawigacji robotem. W projekcie postanowiono, że trajektoria będzie wyznaczana przez osobny komputer (nieznajdujący się na robocie), a następnie przesyłana bezprzewodowo do robota. Celem takiego podejścia jest zmniejszenie obciążenia obliczeniowego komputera znajdującego się na robocie.

3 Model otoczenia

Obiekt, dla którego planowana jest trajektoria, znajduje się w zamkniętym pomieszczeniu z przeszkodami o wymiarach 20x34 m. Ze względu na fakt, że robot porusza się po powierzchni, model otoczenia uproszczono do dwuwymiarowej mapy widocznej na Rysunku 2. Mapa otoczenia została za modelowana w języku Python 3.6 z wykorzystaniem modułu PyGame [1].

Kontury zaznaczone czarnym kolorem oznaczają ściany pomieszczenia, w którym znajduje się robot, w związku z czym ten rodzaj przeszkody nie zmienia swojego położenia. Na rysunku widoczne są również szare kwadraty, które symbolizują przeszkody zmieniające swoje położenie np. ludzie znajdujący się w pomieszczeniu lub inne roboty. Dobór położenia tych przeszkód jest losowy. Przyjęto założenie, że podczas wyznaczania trajektorii system posiada informację na temat położenia przeszkód oraz robota.

W tej pracy rozważane są przypadki wyznaczania trajektorii dla stałego punktu startowego i końcowego trajektorii, które są zaznaczone na Rysunku 2. jako zielony i czerwony punkt. W celu porównania różnych algorytmów postanowiono o przetestowaniu algorytmów dla różnych rozkładów przeszkód - Rozdziale 6. Na tym etapie projektu nowy rozkład przeszkód (szare kwadraty) jest uzyskiwany poprzez generowanie nowej mapy o innym losowym ich rozmieszczeniu. Następnie tworzona jest nowa trajektoria, dzięki czemu algorytmy można przetestować dla większej ilości przypadków.



Rysunek 2: Dwuwymiarowy model otoczenia.

4 Wymagania systemu

Podczas poszukiwań odpowiedniego algorytmu dla systemu wyznaczania trajektorii postanowiono o przetestowaniu kilku algorytmów. W celu ich doboru oraz późniejszym wybraniu najlepszego z nich postanowiono zdefiniować wymagania stawiane systemowi:

- System powinien pozwalać na bezkolizyjne planowanie trajektorii od punktu startowego do końcowego.
- Użyty algorytm powinien mieć wysoka skuteczność znajdywania trajektorii.
- Długość wyznaczonej trajektorii powinna być względnie jak najmniejsza.
- Algorytm powinien posiadać względnie niską złożoność obliczeniową.

Po określeniu tych wymagań rozpoczęto pracę na zapoznawaniem się z algorytmami wyznaczania trajektorii.

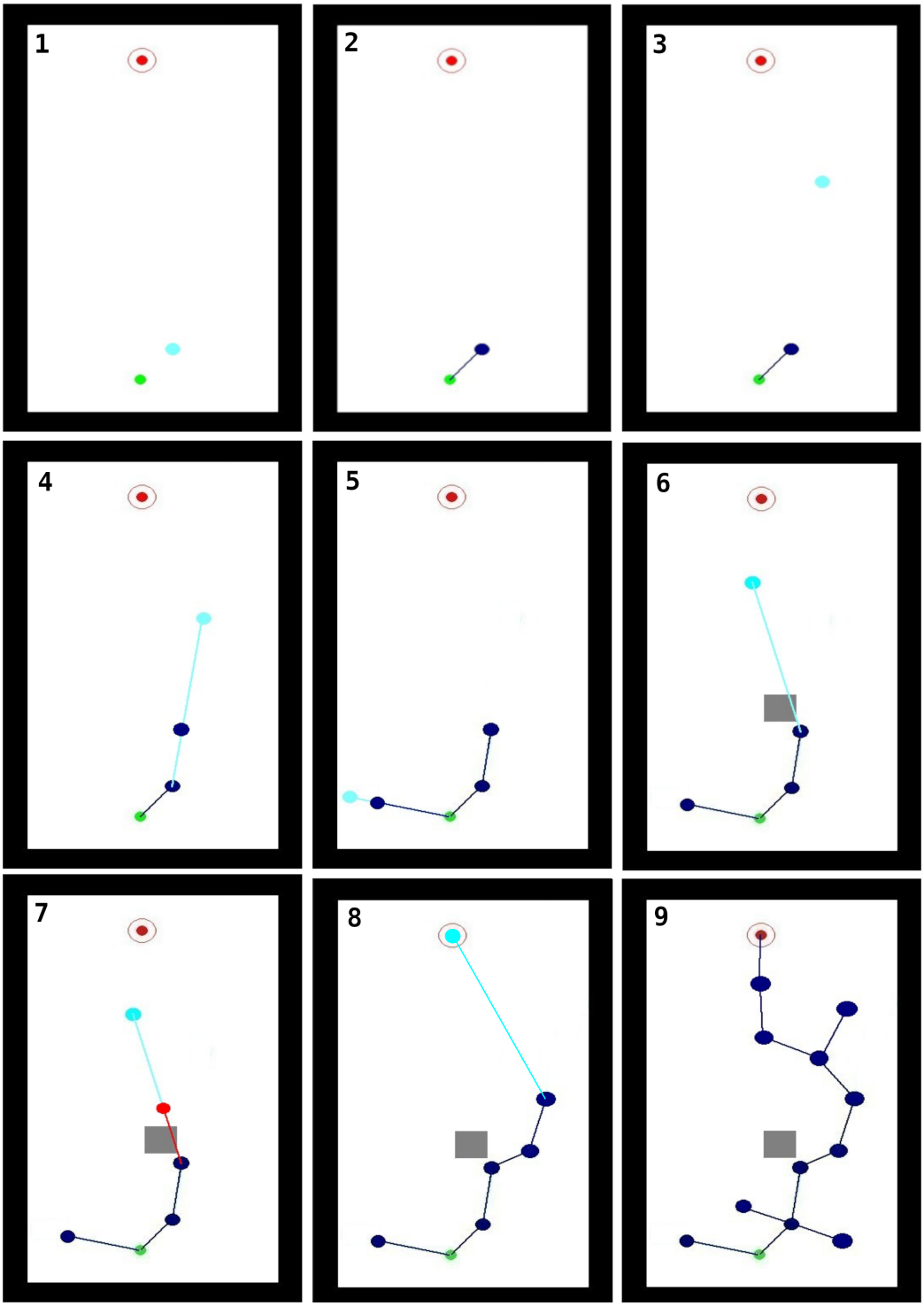
5 Opis algorytmów do wyznaczania trajektorii

Algorytmy wyznaczania trajektorii ze względu na metodę eksploracji otoczenia dzielą się głównie na dwie grupy. Do pierwszej grupy zaliczają się algorytmy oparte na przeszukiwaniu otoczenia (ang. search-based method), które zazwyczaj operują na zdyskretyzowanej mapie otoczenia. Druga grupa algorytmów oparta jest na metodzie próbkowania otoczenia (ang. sampling-based method), aby znaleźć punkty, które po połączeniu utworzą pożądaną trajektorię. W celu znalezienia optymalnego algorytmu postanowiono sprawdzić algorytmy z obu grup, a następnie na podstawie przeprowadzonej analizy wybrać najlepszy ze względu na stawiane wymagania. W pierwszej kolejności skupiono się na algorytmie opartym na próbkowaniu - RRT (ang. Rapidly-exploring Random Tree), a następnie postanowiono zoptymalizować otrzymaną trajektorię za pomocą dwóch różnych metod. Kolejnym krokiem była implementacja algorytmów opartych na poszukiwaniu. W tym celu postanowiono zdyskretyzować mapę otoczenia i wykorzystać algorytm Dijkstra do znalezienia pożądanej trajektorii.

5.1 Algorytm RRT

RRT (ang. Rapidly-exploring Random Tree) [2, 3, 6] jest algorytmem służącym do efektywnego znajdywania trajektorii poprzez losowe próbkowanie otoczenia. Po każdym losowym wybraniu punktu szukane jest najbliższe połączenie do wcześniej uzyskanej próbki. W ten sposób powstaje graf, który rozrasta się w każdym kroku iteracyjnym do momentu znalezienia połączenia od punktu startowego do punktu końcowego trajektorii. Bardziej szczegółowy schemat działania algorytmu przedstawia Rysunek 3, który został opisany poniżej.

Schemat algorytmy RRT przedstawia 9 przykładowych etapów wyznaczania trajektorii pomiędzy zielonym punktem (start) a czerwonym (cel). Na początku działania algorytmu wybierany jest losowy punkt (oznaczony jasnoniebieski kolorem). Następnie szukany jest najbliższy sąsiad (w tym przypadku jest to punkt startowy) i sprawdzane są warunki czy połączenie pomiędzy punktami jest możliwe, tzn. czy dystans pomiędzy punktami nie jest większy niż dopuszczalny, określony jako parametr wejściowy do algorytmu oraz czy nie występuje przeszkoda pomiędzy tymi punktami. Jeżeli wszystkie z wymienionych warunków są spełnione, to powstaje połączenie pomiędzy punktami, co jest widoczne w kroku 2. Następnie losowany jest kolejny punkt, co przedstawiono w etapie 3. Przyjęty punk znajduje się w odległości znacznie większej niż dopuszczalna, dlatego na odcinku łączącym obrany punkt z najbliższym punktem powstającego drzewa, wyznaczany jest



Rysunek 3: Schemat objaśniający działanie algorytmu RRT

punkt w maksymalnej dopuszczalnej odległości od najbliższego węzła drzewa. Etap ten przedstawiono w kroku 4. Kolejny wybrany punkt również znajduje się w zbyt dużej na najbliższego węzła (punkt startowy), dlatego również obierany bliższy punkt znajdujący się na odcinku łączącym wylosowany punkt i punkt startowy. W etapie 6 i 7 przedstawiono sytuację, gdy wylosowany punkt znajduje się daleko od aktualnego drzewa oraz pomiędzy punktem a najbliższym węzłem drzewa znajduje się przeszkoda (szary kwadrat). W tym przypadku połączenie pomiędzy węzłem drzewa a punktem znajdującym się w dopuszczalnej odległości od tego węzła (czerwony punkt) jest niemożliwe ze względu na kolizję. Zaimplementowana wersja algorytmu RRT różni się od klasycznej wersji ponieważ, co kilka iteracji zamiast losowania punktu wybierany jest punkt końcowy, dzięki czemu graf jest rozrasta się szybciej w kierunku celu. Omaiwana sytuacja jest widoczna w etapie 8. W kolejnych krokach iteracyjnych wybierane są kolejne punkty, dzięki czemu drzewo się rozrasta, aż do osiągnięcia połączenia pomiędzy punktem startowym a docelowym, co jest widoczne na ostatniej ilustracji.

Schemat działania algorytmu RRT przedstawia również pseudokod znajdujący się następnej stronie.

Opisany algorytm zaimplementowano do oprogramowania i przetestowano na mapie otoczenia opisanej w Rozdziale 3. Pozostała część argumentów wejściowych została przedstawiona w poniższej tabeli.

Tablica 1: Parametry algorytmu RRT

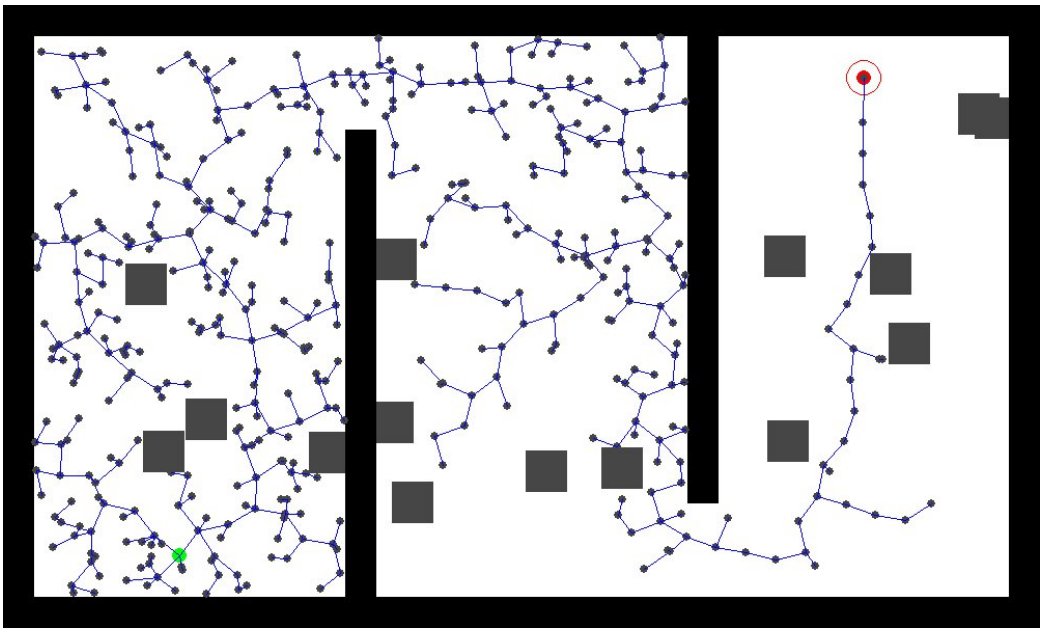
Symbol	d_{max}	d_{cel}	$iter_{max}$	$f_{l/c}$
Wartość	1 m	0,5 m	5000	10

Algorithm 1 RRT

Wejście: map - mapa z przeszkodami, q_{start} - punkt startowy, $q_{koncowy}$ - punkt końcowy, d_{max} - maksymalna dopuszczalna odległość pomiędzy węzłami, d_{cel} - maksymalna odległość węzła grafu od węzła końcowego, dla której uznaje się, że trajektoria została wyznaczona, $iter_{max}$ - maksymalna dopuszczalna ilość iteracji, $f_{l/c}$ - współczynniki określający ilość losowych próbek do ilości używania jako próbkę punkt końcowego
Wyjście: $traj$ - trajektoria (lista punktów tworzących trajektorię)
Inicjalizacja: $G.init(q_{start})$ - inicjalizacja grafu G,
$iter = 0$ - zmienna przechowująca ilość itreracji pętli
$q_{nowy} \leftarrow q_{start}$

while $(d_{cel} < DYSTANS(q_{nowy}, q_{koncowy}))$ || $(iter < iter_{max})$ **do**
  if $(iter \ \% \ f_{l/c} == 0)$ **then**
    $q_{nowy} \leftarrow q_{koncowy}$
  end
  else
    $q_{nowy} = LOSUJ\_PUNKT()$
  end
  q_{nw} = G.ZNAJDŹ_NAJBLIŻSZY_WĘZEŁ(q_{nowy})
  if $(d_{max} < DYSTANS(q_{nowy}, q_{nw}))$ **then**
    q_{nowy} = ZNAJDŹ_BLIŻSZY_PUNKT(q_{nowy}, q_{nw}, d_{max})
  end
  if $(KOLIZJA(q_{nowy}, q_{nw}, map))$ **then**
    break
  end
  else
    $G.DODAJ\_POŁĄCZENIE(q_{nw}, q_{nowy})$
  end
end
traj $\leftarrow G.TRAJEKTORIA()$
return $traj$

Na poniższym rysunku widać przykładowy wygląd siatki połączeń uzyskany z algorytmu RRT.

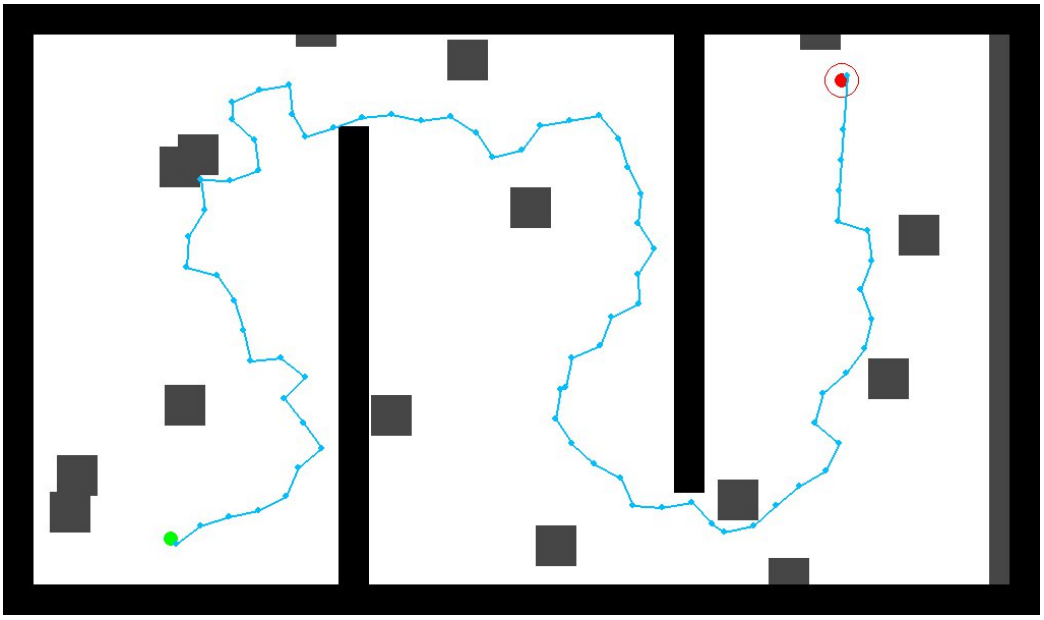


Rysunek 4: Całkowity graf algorytmu RRT

Analizując powyższy rysunek, można zauważyć duże zagęszczenie punktów i lewej części mapy, gdzie znajduje się punkt startowy a o wiele mniejsze w prawej części mapy, gdzie znajduje się punkt docelowy. Jest to spowodowane tym, że do próbkowania mapy użyto rozkładu równomiernego oraz tym, że drzewo rozpoczęło się rozrastać przy punkcie startowym, dzięki czemu mogło utworzyć wiele więcej połączeń w lewej części mapy.

Po przetestowaniu poprawności działania algorytmu postanowiono obrać jedną mapę ze stałymi położeniami przeszkód, aby móc porównać wyniki otrzymane przez algorytm RRT z algorytmami opisanymi w dalszej części pracy. Otrzymana trajektoria przez algorytm RRT dla mapy referencyjnej jest przedstawiona dla Rysunku 5

Uzyskana trajektoria łączy punkt startowy (zielony) z punktem docelowym (czerwony), jednak jest daleka od trajektorii optymalnej, ponieważ jej długość mogłaby być znacznie zredukowana np. poprzez wyeliminowanie niektórych węzłów i stworzenie nowych połączeń. W kolejnych podrozdziałach przedstawiono dwie metody optymalizacyjne tej trajektorii.



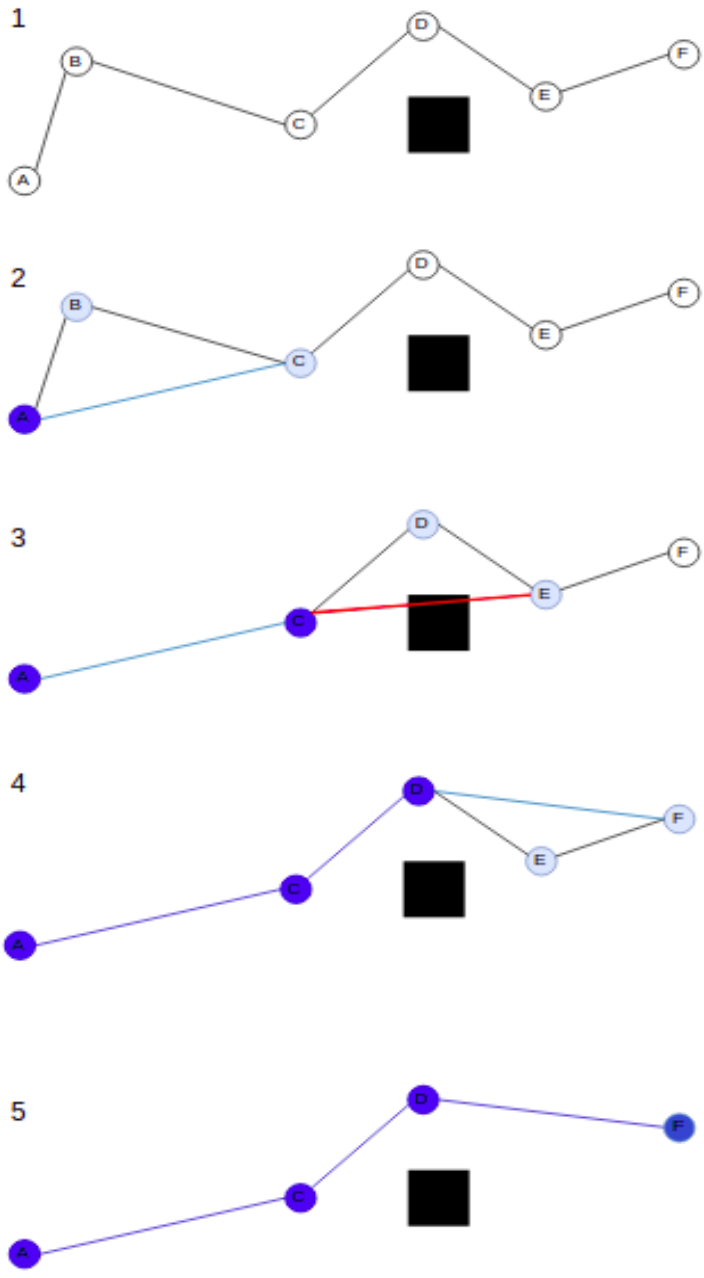
Rysunek 5: Trajektoria uzyskana z algorytmu RRT.

5.2 Algorytm RRTO3P

Jedną z prostych metod optymalizacyjnych trajektorii jest iteracja po wszystkich węzłach trajektorii i sprawdzanie warunku, czy jest możliwe połączanie pomiędzy węzłem n i $n+2$, gdzie n to numer węzła w trajektorii. Jeżeli warunek jest spełniony to węzeł $n+1$ jest usuwany oraz tworzone jest połączenie pomiędzy węzłem n i $n+2$. Nowo powstała trajektoria jest krótsza niż poprzednia, ponieważ w trójkącie każdy bok ma mniejszą długość niż suma długości pozostałych boków (warunek istnienia trójkąta). W pracy przyjęto oznaczenie dla tego typu algorytmu optymalizującego - RRTO3P (ang. Rapidly-exploring Random Tree - Optimized - 3 Points). Na Rysunku 6. przedstawiono schemat opisanego algorytmu.

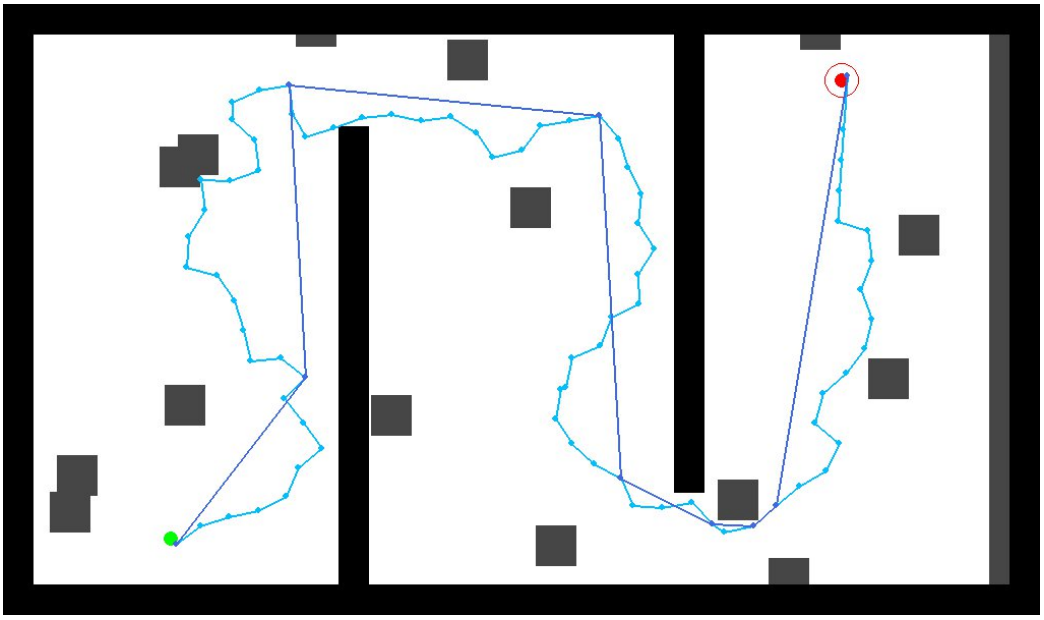
Trajektoria widoczna na Rysunku 6 utworzona jest przez połączenia pomiędzy węzłami A, B, C, D, E, F. Na grafice widoczna jest również przeszkoda, oznaczona jako czary kwadrat. Optymalizacja zaczyna się od węzła A gdzie sprawdzane jest czy możliwe jest stworzenie połączenia pomiędzy punktem A i C. W tym przypadku kolizja nie występuje wobec czego można usnąć węzeł B i stworzyć połączenie pomiędzy węzłami A i C. W kroku 3. widoczne jest, że pomiędzy węzłem C i E występuje przeszkoda dlatego nie możliwe jest stworzenie nowego połączenia - algorytm przechodzi do następnego węzła. Na odcinku DF nie występuje kolizja, dlatego zoptymalizowana trajektoria, uzyskana w pierwszym kroku iteracyjnym składa się z punktów

A, C, D, F. Warto zwrócić uwagę, że algorytm można zastosować ponownie do nowo powstałej trajektorii, wobec czego można uzyskać krótszą trajektorię A-D-F.



Rysunek 6: Schemat przedstawiający działanie algorytmu optymalizującego ścieżkę uzyskaną z algorytmu RRT - RRTO3P

Omawiany algorytm zaimplementowano w oprogramowaniu i przetestowano na trajektorii otrzymanej przez algorytm RRT. Wyniki zostały przedstawione na poniższym rysunku.

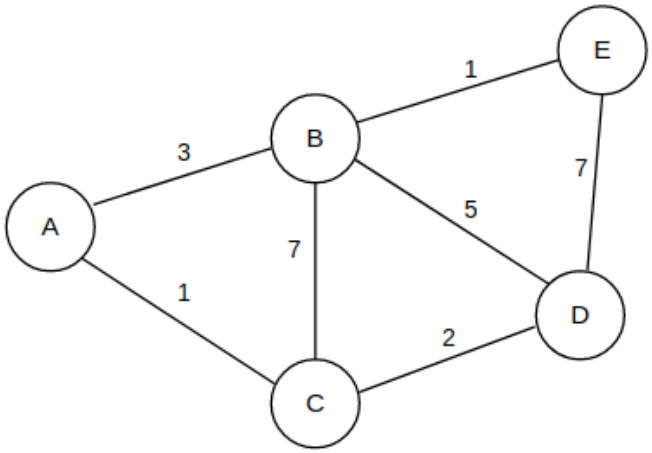


Rysunek 7: Trajektorie uzyskane przez algorytmy: RRT (jasnoniebieski kolor), RRTO3P (niebieski kolor).

Analizując powyższy rysunek widać, że algorytm znacznie uprościł trajektorię, dzięki czemu nowo otrzymana trajektoria jest krótsza o koło 29% długości trajektorii RRT. Jednak można również zauważyć, że wybór węzłów w tej metodzie nie został dokonany w optymalny sposób, dlatego kolejny sposób optymalizacji trajektorii zakłada wykorzystanie algorytmu minimalizującego długość trajektorii.

5.3 Algorytm RRTOD

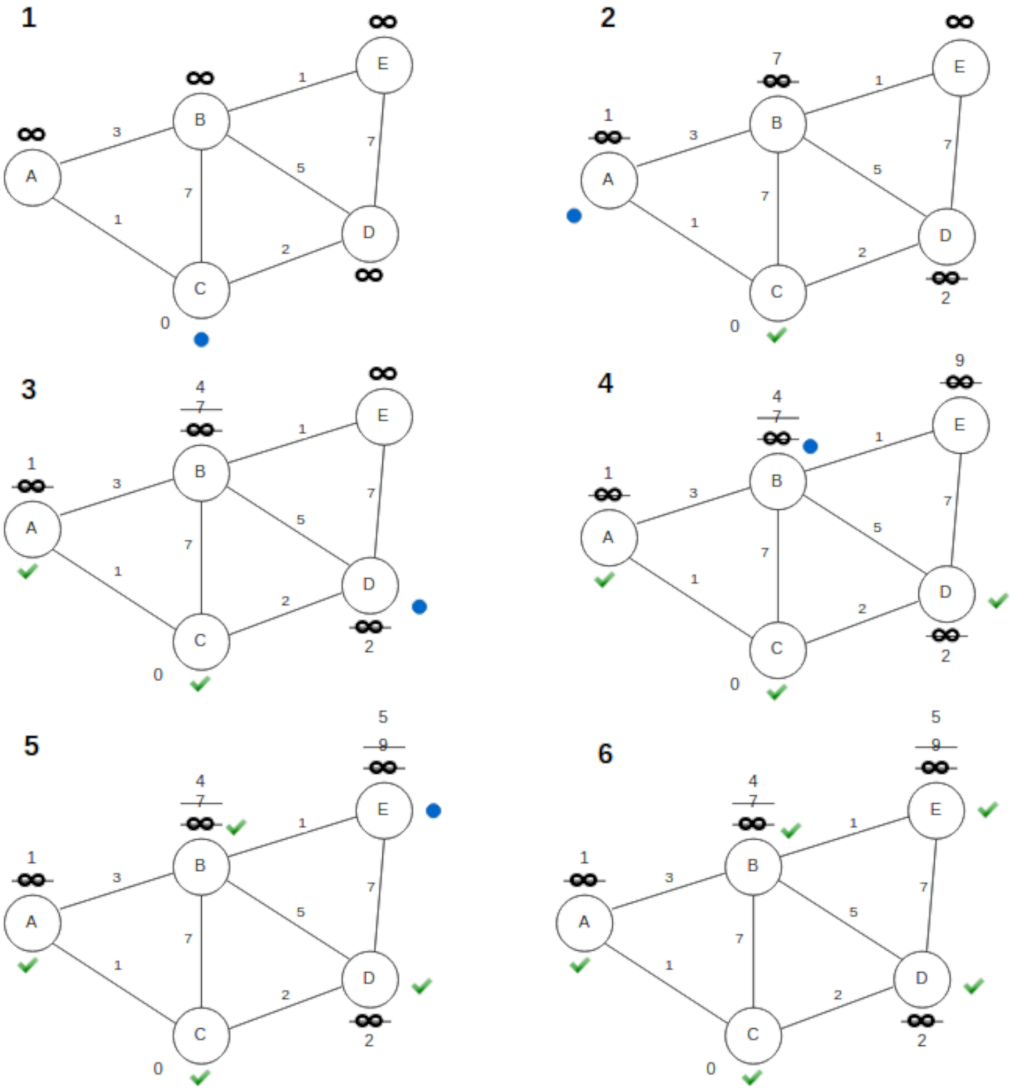
Drugą metodą optymalizacji trajektorii uzyskanej z algorytmu RRT jest wykorzystanie algorytmu Dijkstra [5, 7], który często jest wykorzystywany w grafach do wyznaczania ścieżki o minimalnym koszcie (sumie wag pomiędzy węzłami). Opisywany algorytm otrzymywania trajektorii został skrótowo nazwany RRTOD (ang. Rapidly-exploring Random Tree - Optimized - Dijkstra). Schemat działania algorytmu Dijkstra został przedstawiony na podstawie grafu znajdującego się na Rysunku 8.



Rysunek 8: Graf użyty do wyjaśnienia algorytmu Dijkstra

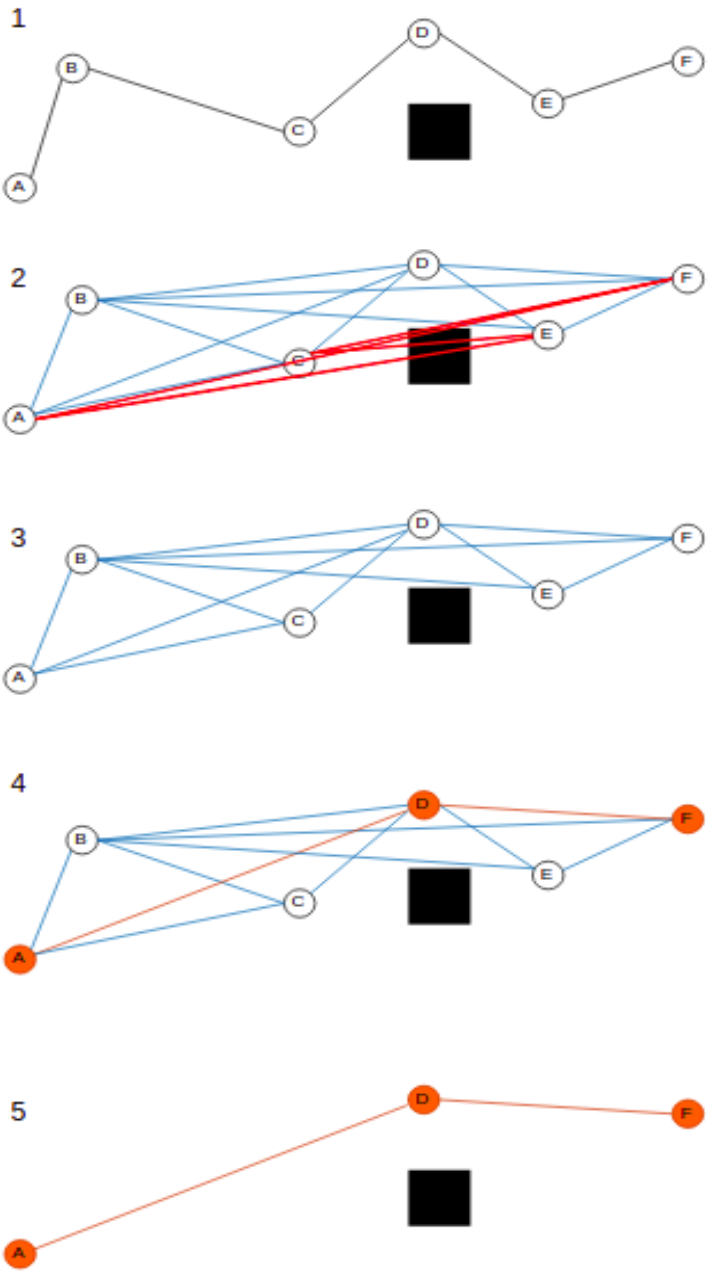
Powyższy graf składa się z 5 węzłów połączonych ze sobą. Każde połączenie ma swoją wagę, która oznacza koszt przemieszczenia się z jednego węzła do drugiego. Celem stawianym algorytmowi Dijkstra jest znalezienie ścieżki pomiędzy węzłem C i E o najmniejszym koszcie. Na Rysunku 9 przedstawiony schemat objaśniający działania algorytmu a poniżej opis schematu.

W pierwszym kroku wszystkie węzły są oznaczane jako nieodwiedzone, wybierany jest węzeł startowy (C) oraz inicjalizowane są odległości węzłów od węzła startowego - wszystkie wartości równe nieskończoności oprócz wartości dla węzła startowego, która jest równa 0. W następnym kroku dla sąsiadów obecnie wybranego węzła nadpisywane są zmienne przechowujące odległość węzłów od węzła startowego. Węzeł C zostaje oznaczony jako odwiedzony i algorytm zmienia obecny węzeł z C na A. W trzecim kroku obliczane są odległości dla sąsiadów węzła A. Nadpisana zostaje wartość dla węzła B, ponieważ ścieżka C-A-B jest mniej kosztowna niż ścieżka C-B. Następnie węzeł A zostaje oznaczony jako odwiedzony i algorytm przechodzi do węzła D. W czwartym kroku zaktualizowana zostaje wartość odległości od węzła startowego dla węzłów C, B, E, co skutkuje zmianą wartości dla węzła E z nieskończoności na 9. Później węzeł D zostaje oznaczony jako odwiedzony i obecny węzeł zostaje zmieniony na B. Podczas aktualizacji wartości zmianie ulega wartość dla węzła E, która się zmienia z 9 na 5, co oznacza, że ścieżka C-A-B-E jest mniej kosztowna niż C-D-E. Następnie węzeł B zostaje oznaczony jako odwiedzony i algorytm przenosi się do węzła E. W ostatnim kroku żadna z wartości nie ulega zmianie, wobec czego najkrótszą ścieżką z punktu C do E jest ścieżka C-A-B-E.



Rysunek 9: Schemat objaśniający działanie algorytmu Dijkstra.

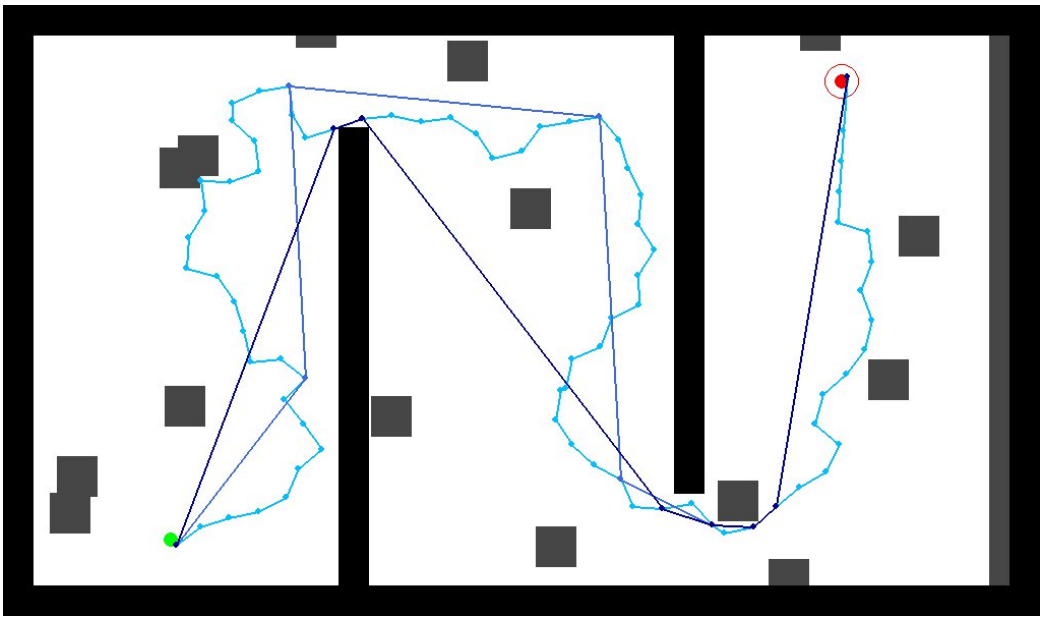
Opisany algorytm może zostać zastosowany do optymalizacji trajektorii, jednak w tym celu konieczne jest przekształcenie trajektorii w graf. Na Rysunku 10 przedstawiony został schemat optymalizacji trajektorii za pomocą algorytmu Dijkstra.



Rysunek 10: Schemat przedstawiający działanie algorytmu optymalizującego ścieżkę uzyskaną z algorytmu RRT - RRTOD

Trajektoria przedstawiona na powyższym rysunku jest taka sam jak w przykładzie omawianym na podstawie Rysunku 6. W celu stworzenia grafu konieczne jest wytworzenie połączeń pomiędzy wszystkimi węzłami. Wagami w tym przypadku są odległości pomiędzy węzłami. Ze względu na występowanie przeszkody, niektóre połączenia są kolizyjne (oznaczone na czerwono) i nie mogą być brane pod uwagę w dalszych rozważaniach. Wszystkie niekolizyjne połączenia są zaznaczone na niebiesko. W kroku czwartym użyty został algorytm Dijkstra, który wyznaczył ścieżkę (A-D-F) o minimalnym koszcie.

Po implementacji algorytmu Dijkstra w oprogramowaniu przetestowano go na trajektorii uzyskanej z algorytmu RRT. Wyniki przedstawia poniższy rysunek.

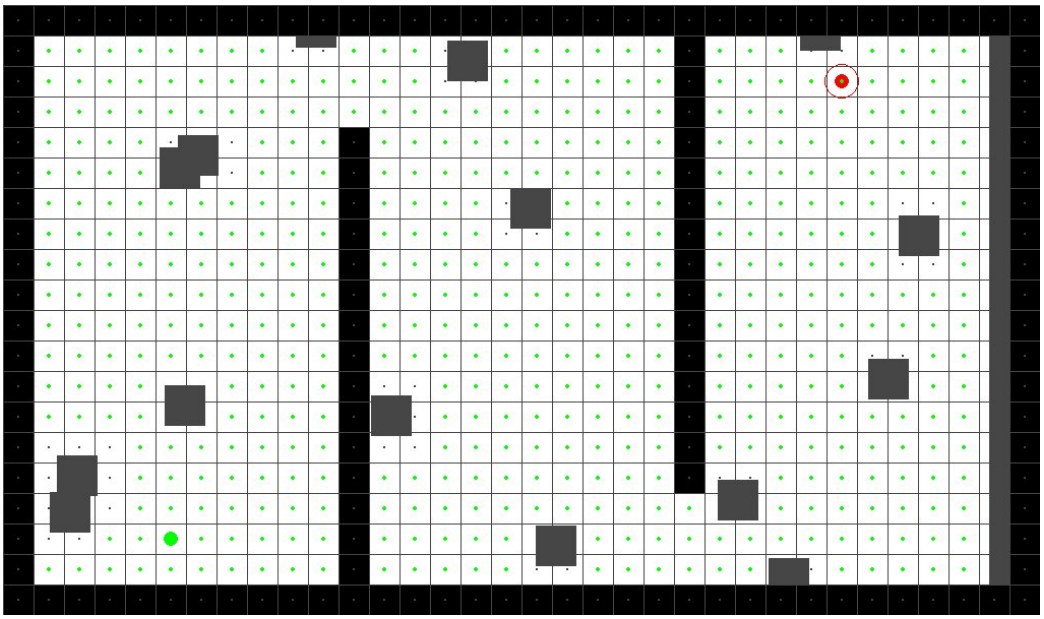


Rysunek 11: Trajektorie uzyskane przez algorytmy: RRT (jasnoniebieski kolor), RRTO3P (niebieski kolor), RRTOD (ciemnoniebieski kolor).

Nowo powstała trajektoria jest o krótsza od trajektorii bazowej o około 50% jej długości. Tak znaczące skrócenie długości zostało osiągnięte dzięki wybraniu węzłów, których połączenie daje najmniejszą odległość z punktu startowego do docelowego.

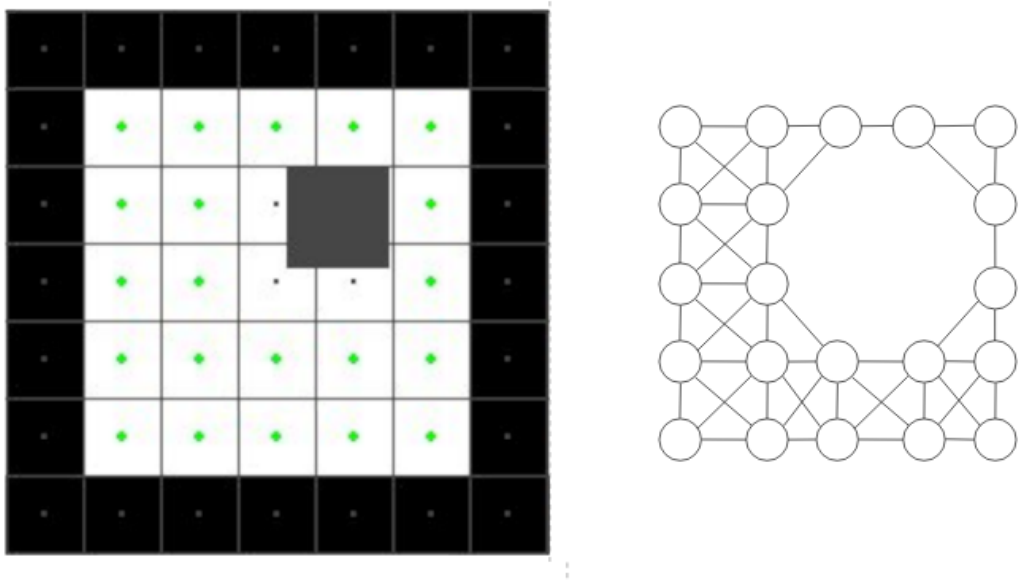
5.4 Algorytm GS

Kolejnym sposobem na wyznaczenie trajektorii było wykorzystanie algorytmu opartego na przeszukiwaniu otoczenia. W tym celu na początku zdyskretyzowano mapę otoczenia dzieląc go na 20x34 kwadratowych pól, których środki oznaczały możliwą pozycję węzła trajektorii. Na Rysunku 12. przedstawiona jest siatka pól, która ilustruje sposób dyskretyzacji mapy. W każdy z pól zaznaczony jest środek, który w zależności tego, czy na danym polu znajduje się przeszkoda - kolor czarny (zajęte pola) lub zielony (wolne pola). Dla opisywanego algorytmu przyjęto oznaczenie GS (ang. Grid Search).



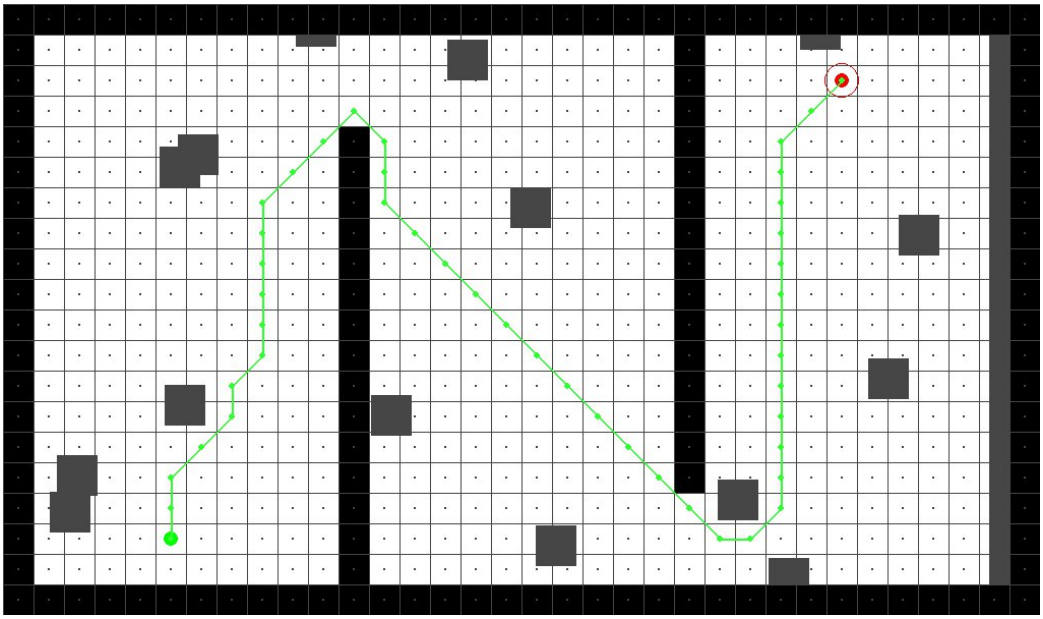
Rysunek 12: Zdyskretyzowana mapa otoczenia

Pod względem obliczeniowym tak zdyskretyzowaną mapę można traktować jako tablicę dwuwymiarową lub graf. W niniejszej pracy mapę potraktowano jako graf, którego węzły tworzą wolne pola (zielone) a połączenia pomiędzy węzłami mogą występować jedynie w najbliższym sąsiedztwie węzła. Interpretacja graficzna tego została przedstawiona poniżej.



Rysunek 13: Interpretacja zdyskretyzowanej mapy jako grafu.

Mając tak wyznaczony graf można wykorzystać algorytm Dijkstra omówiony w Podrozdziale 5.3. Na Rysunku 14. przedstawiono wyniki uzyskane po implementacji wyżej mówionego rozwiązania w oprogramowaniu.

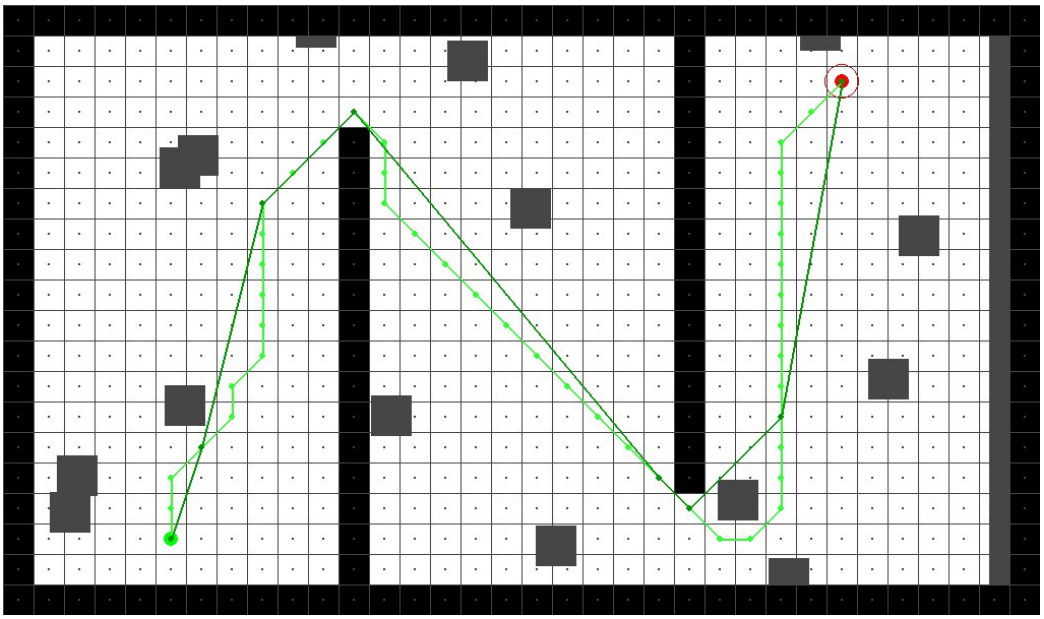


Rysunek 14: Trajektoria uzyskana przez algorytm GS

Trajektoria uzyskana w ten sposób jest najkrótszą dla rozważanego przypadku dyskretyzacji mapy oraz metody tworzenia grafu. Jednak ze względu na to, że połączenia pomiędzy węzłami występują tylko w najbliższym sąsiedztwie, uzyskana trajektoria nie jest najkrótszą możliwą dla wyznaczonych węzłów. Fakt ten sprawia, że uzyskana trajektoria może zostać zoptymalizowana pod względem długości przy ponownym wykorzystaniu algorytmu Dijkstra dla wyznaczonej trajektorii.

5.5 Algorytm GSOD

Winsoki uzyskane we wcześniejszym podrozdziale stanowiły podstawy do wyznaczenia nowego algorytmu wyznaczani trajektorii poprzez optymizację trajektorii uzyskanej przez algorytm GS metodą Dijkstra. Dla tego nowego algorytmu przyjęto oznaczenie GSOD (ang. Gird Search - Optimized - Dijkstra). Na Rysunku 15 został przedstawiony rezultat omawianej optymalizacji.



Rysunek 15: Trajektorie uzyskane przez algorytmy: GS (jasnozielony kolor), GSOD (ciemnozielony kolor).

Nowo uzyskana trajektoria jest krótsza o około 10% długości pierwotnej trajektorii, co potwierdza wnioski wyciągnięte podczas analizy algorytmu GS.

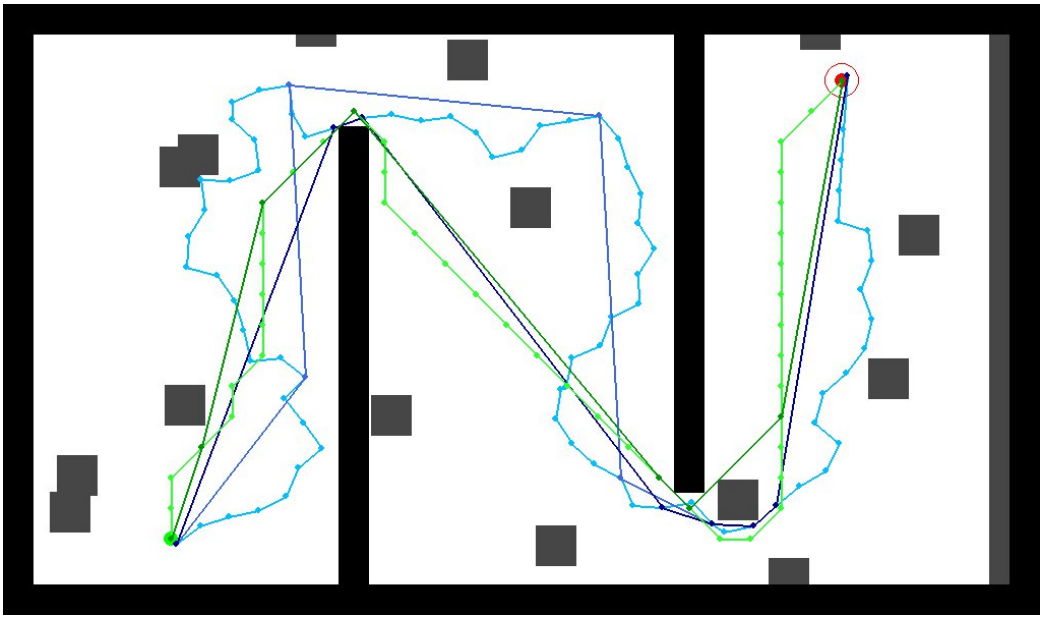
6 Porównanie algorytmów

Po implementacji wszystkich wyżej wymienionych algorytmów zdecydowano się je porównać w celu wybrania najlepszego pod względem wymagań stawianym systemowi. Głównymi kryteriami było:

- Skuteczność znajdywania trajektorii
- Długość trajektorii - minimalizacja długości
- Złożoność obliczeniowa algorytmu

6.1 Mapa referencyjna

W pierwszej kolejności porównano trajektorie otrzymane dla mapy referencyjnej, analizowane we wcześniejszych podrozdziałach. W tym celu zestawiono wszystkie trajektorie na jednym rysunku oraz wyznaczono ich długości.



Rysunek 16:

Tablica 2: Długości trajektorii otrzymane poprzez poszczególne algorytmy.

	RRT	RRTO3P	RRTOD	GS	GSOD
Długość trajektorii [m]	56,62	43,93	37,52	39,52	36,08

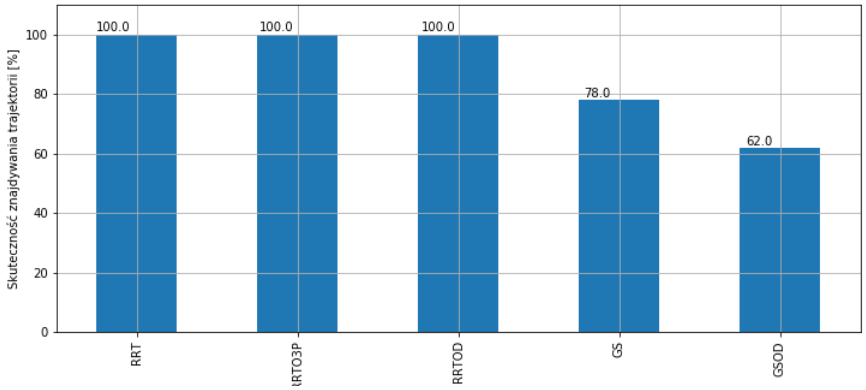
Z przedstawionych danych wynika, że trajektoria uzyskana z algorytmu RRT jest najdłuższa. Zastosowanie algorytmów optymalizujących ją pozwoliło na otrzymanie znacznie krótszych trajektorii. Najlepszy wyniki uzyskano przy zastosowaniu algorytmu GSOD.

6.2 Porównanie statystyczne

Ze względu na fakt, że w otoczeniu występują przeszkody, których pozycja jest losowa oraz algorytm RRT jest stochastyczny, postanowiono sprawdzić algorytmy na większej ilości map, a następnie przeprowadzić analizę statystyczną. W tym celu stworzono 50 map otoczenia, które różniły się rozmieszczeniem przeszkód (szarych kwadratów). Następnie dla każdej mapy zastosowano przygotowane algorytmy wyznaczania trajektorii.

6.2.1 Skuteczność algorytmów

Podczas obliczeń okazało się, że nie każdy z algorytmów jest w stanie znaleźć trajektorię dla każdej z map. Na poniższym rysunku przedstawiono procentową skuteczność znajdywania trajektorii w zależności od algorytmu.

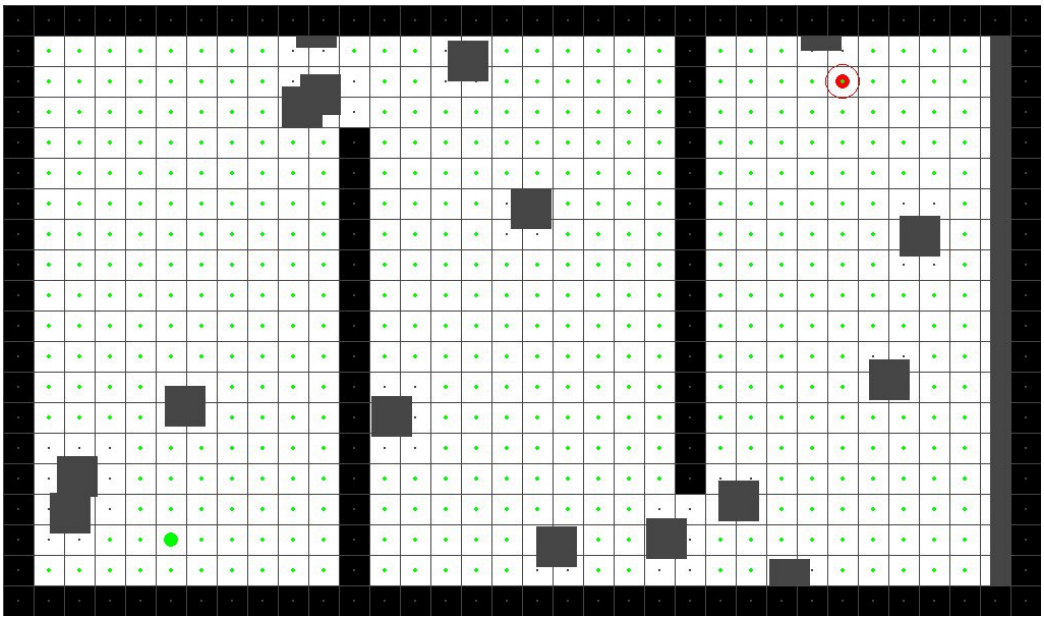


Rysunek 17: Skuteczność znajdywania trajektorii.

Analizując powyższy wykres można zauważyć, że algorytmy oparte na RRT wyznaczyły trajektorię dla wszystkich 50 map. Skuteczność algorytmów opartych na przeszukiwaniu zdeskretyzowanej mapy jest znacznie niższa.

Podczas poszukiwania przyczyn tej różnicy okazało się, że skuteczność metod opartyczych na dyskretyzaczji silnie zależy od rozmieszczenia przeszkód

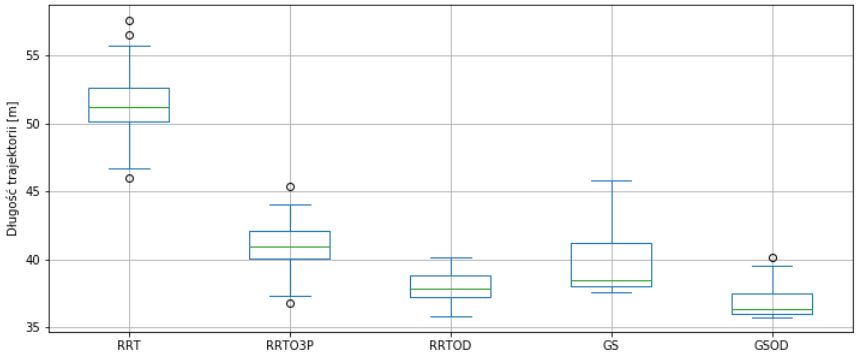
i kroku dyskretyzacji. Przy obecnych parametrach mapy istnieje duże prawdopodobieństwo, że przejścia pomiędzy ścianami zostaną zablokowane przez losowe przeszkody, co sprawia, że nie jest możliwe stworzenie połączeń pomiędzy węzłami znajdującymi się w lewej i środkowej części mapy lub środkowej i prawej części mapy. Przypadek ten został zobrazowany na Rysunku 18.



Rysunek 18: Zablokowanie połączenia pomiędzy częściami mapy przez przeszkody.

6.2.2 Długość trajektorii

Kolejnym kryterium porównawczym jest długość trajektorii. Podczas porównywanie algorytmów wykorzystano tylko te mapy dla których każdy z algorytmów znalazł rozwiązanie. W związku z tym statystyka długości trajektorii została wyznaczona na podstawie 31 trajektorii otrzymanych z każdego algorytmu. Na Rysunku 19 przedstawiono wykres pudełkowy przedstawiający rozkład długości otrzymanych trajektorii.



Rysunek 19: Statystyka długości trajektorii

Tablica 3: Parametry statystyczne opisujące rozkład długości trajektorii dla poszczególnych algorytmów.

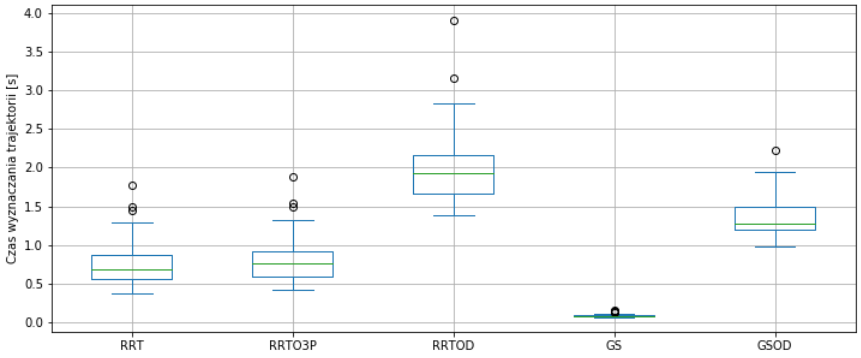
	RRT	RRTO3P	RRTOD	GS	GSOD
Średnia [m]	51.39	40.95	38.01	39.65	36.85
Odchylenie standardowe [m]	2.71	1.97	1.12	2.34	1.26
Wartość minimalna [m]	45.96	36.82	35.82	37.58	35.77
Pierwszy kwantyl [m]	50.09	40.07	37.23	38.02	35.96
Mediana [m]	51.17	40.99	37.83	38.46	36.37
Trzeci kwantyl [m]	52.61	42.06	38.84	41.24	37.48
Wartość maksymalna [m]	57.59	45.37	40.13	45.78	40.16

Z danych przedstawionych na Rysunku 19 oraz w Tabeli 3 wynika, że najmniejszą średnią długość trajektorii uzyskano przy użyciu algorytmu GSOD. Watro zwrócić uwagę, że wartość średniej długości trajektorii dla algorytmu RRTOD jest tylko nieco ponad 1 metr większa, ale za to algorytm jest bardziej powtarzany, ponieważ posiada mniejsze odchylenie standardowe.

6.2.3 Złożoność obliczeniowa

Ostatnim kryterium porównawczym jest złożoność obliczeniowa. Ze względu na fakt, że wykorzystano zarówno algorytmy deterministyczne (Dijkstra) jak i stochastyczne (RRT), trudno jest w sposób ścisły dokonać tego zestawienia. W związku z tym postanowiono o porównaniu parametrów statystycznych czasu potrzebnego do wyznaczenia trajektorii przez dane algorytmy. W tym celu wykorzystano te same mapy otoczenia co w Podrozdziale 6.2.2, dla

których każdy z algorytmów był wstanie wyznaczyć trajektorię. Obliczenia wykonywano na laptopie wyposażonym w procesor Intel(R) Core(TM) i7-7500U CPU 2.70GHz oraz 16 GB pamięci RAM. Wyniki przedstawiono na Rysunku 20. oraz Tabeli 4.



Rysunek 20: Statystyka czasu potrzebnego do wyznaczania trajektorii

Tablica 4: Parametry statystyczne opisujące rozkład długości trajektorii dla poszczególnych algorytmów.

	RRT	RRTO3P	RRTOD	GS	GSOD
Średnia [s]	0.808	0.859	2.052	0.088	1.383
Odchylenie standardowe [s]	0.339	0.344	0.550	0.023	0.288
Wartość minimalna [s]	0.380	0.426	1.380	0.070	0.976
Pierwszy kwantyl [s]	0.557	0.600	1.665	0.074	1.198
Mediana [s]	0.693	0.767	1.929	0.079	1.281
Trzeci kwantyl [s]	0.876	0.926	2.154	0.089	1.497
Wartość maksymalna [s]	1.777	1.879	3.906	0.152	2.227

Analizując wyżej przedstawione dane, można zauważyć, że najmniej czasu obliczeniowego zużywa algorytm GS. Również dla tego algorytmu odchylenie standardowe ma bardzo niską wartość. Jest to spowodowane tym, że algorytm ten wykorzystuje deterministyczny algorytm Disjkstra, w związku z tym czas potrzebny na obliczenia zależy głównie złożoności grafu. Grafy powstałe na podstawie użytych map są bardzo podobne pod względem ilości węzłów i połączeń pomiędzy nimi, związku z czym czas potrzebny na wyznaczenie trajektorii jest bardzo zbliżony.

Średni czas potrzebny na wyznaczenie trajektorii z wykorzystaniem algorytmu RRT jest niemal 10 razy większy. Tak duża różnica jest spowodowana

głównie faktem, że algorytm ten próbkuje otoczenie o wiele więcej razy niż liczba możliwych węzłów w zdyskretyzowanej mapie otoczenia. Za każdym razem gdy wybrana jest próbka otoczenia, algorytm RRT musi dodatkowo sprawdzić czy pomiędzy tą próbką, a najbliższym węzłem grafu nie występuje kolizja, co przy zastosowanych funkcjach sprawdzających jest kosztowne obliczeniowo. W GS takie sprawdzenie jest niepotrzebne ponieważ w tym algorytmie graf powstaje na początku i składa się jedynie z węzłów w bliskim otoczeniu których nie ma przeszkody.

Warto zwrócić uwagę, że dla algorytmu RRTO3P średni czas jest jedynie większy o 6% średniego czasu dla algorytmu RRT. Porównując ten wynik z danymi z Podrozdziału 6.2.2 wynika, że stosunkowo niewielkie zwiększenie złożoności obliczeniowej pozwoliło na wyznaczanie trajektorii o średnio 20% krótszej. Algorytmy RRTOD oraz GSOD znacznie różnią się pod względem zużycia czasu obliczeniowego od swoich pierwotnych wersji, czyli algorytmów RRT i GS. Różnica ta głównie wynika z konieczności sprawdzenia występowania kolizji na połączaniach pomiędzy węzłami, co jest jest kosztowne obliczeniowo.

7 Podsumowanie

W pracy przedstawiono kilka algorytmów do wyznaczania trajektorii oraz porównano jej pod względem wymagać stawianych systemowi. Algorytmy oparte na próbkowaniu (ang. sampling-based method) mają lepszą skuteczność znajdywania trajektorii niż algorytmy oparte na przeszukiwaniu (ang. search-based method). Wyniki przeprowadzonej analizy statystycznej długości trajektorii pokazały, że zastosowanie optymalizacji dla trajektorii otrzymanych z podstawowych algorytmów (RRT i GS) pozwala na znaczne zmniejszenie długości. Optymalizacja ta wiążę się jednak, że zwiększeniem złożoności obliczeniowej, co może nawet kilkunastokrotnie zwiększyć czas obliczeń.

Ze względu na fakt, że najważniejszym kryteriami systemu do wyznaczania trajektorii są: bezkolizyjność, wysoka skuteczność oraz minimalizacja długości, postawiono o wyborze algorytmu RRTOD dla projektowanego systemu. Algorytm ten ma bardzo wysoką skuteczność planowania oraz wyznaczane trajektorie są stosunkowo krótkie - jedynie algorytm GSOD pozwala na wyznaczenie krótszych trajektorii, jednak jego skuteczność jest dużo niższa. Wadą wyboru RRTOD jest największa złożoność obliczeniowa spośród przedstawionych algorytmów, jednak problem ten można rozwiązać wykorzystując komputer o większej mocy obliczeniowej.

Podczas analizy statystycznej otrzymanych wyników powstało kilka po-

mysłów pozwalających na optymalizację użytych algorytmów. Główny pomysł zakłada zwiększenie parametru d_{max} - maksymalnej dopuszczalnej odległości pomiędzy węzłami dla algorytmów opartych na próbkowaniu oraz zmniejszenie kroku dyskretyzacji dla algorytmów opartych na przeszukiwaniu. Obecny sposób dobory tych długości zakładał, że obydwie te wartości są sobie równe. Zwiększenie parametru d_{max} mogłoby przyspieszyć rozrastanie się grafu, dzięki zmniejszyłaby się liczba próbek potrzebnych do wyznaczania trajektorii. Zmniejszenie kroku dyskretyzacji zwiększyć skuteczność algorytmów, ponieważ powstałby więcej opcji wytyczenia tras, jednak trzeba również pamiętać, że taka zmiana zwiększyłaby czas potrzebny do wyznaczania trajektorii.

Przyszła praca nad systemem zakłada optymalizację parametrów opisanych powyżej oraz adaptację systemu do wyznaczania trajektorii, podczas gdy robot się porusza. W przypadku gdy robot i przeszkody zmieniają swoje położenie, konieczne jest uwzględnienie przypadku, w którym pierwotnie wyznaczona trajektoria staje się kolizyjna. Do wyznaczenia nowej trajektorii można zastosować dwa podejścia: wyznaczyć zupełnie nową trajektorię (do tego jest obecnie przystosowany system) lub wyznaczyć nową trajektorię na podstawie wcześniejszej trajektorii z uwzględnieniem nowych pozycji robota i przeszkód. Te drugie podejście będzie tematem dalszych prac nad systemem.

8 Literatura

[1] *Pygame documentation.* https://www.pygame.org/docs/.

[2] Tim Chin. *Robotic Path Planning: RRT and RRT*.* https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378, 2019.

[3] Dorian Skrobek Dawid Cekus. *Poszukiwanie optymalnej trajektorii manipulatora Scara z wykorzystaniem metod RRT i PSO.* 2017.

[4] Husarion. *ROSbot manual.* https://www.robotshop.com/media/files/content/r/rco/pdf/rosbot-20-pro-robotic-platform-datasheet.pdf.

[5] Muhammad Adeel Javaid. *Understanding Dijkstra's Algorithm.* 2013.

[6] James J. Kuffner Steven M. LaValle. *Rapidly-Exploring Random Trees: Progress and Prospects.* 2000.

[7] Neelam Tyagi. *Dijkstra's Algorithm: The Shortest Path Algorithm.* https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm.

9 Spis skrótów

GS ang. Grid Search
GSOD ang. Grid Search - Optimized - Dijkstra
RRT ang. Rapidly-exploring Random Tree
RRTO3P ang. Rapidly-exploring Random Tree - Optimized - 3 Points
RRTOD ang. Rapidly-exploring Random Tree - Optimized - Dijkstra

10 Spis symboli

map Mapa otoczenia w którym znajduje się robot
q_{start} Punkt startowy trajektorii
$q_{koncowy}$ Punkt końcowy trajektorii
d_{max} Maksymalna dopuszczalna odległość pomiędzy węzłami w algorytmie RRT
d_{cel} Maksymalna odległość węzła grafu od węzła końcowego, dla której uznaje się, że trajektoria została wyznaczona w algorytmie RRT
$iter_{max}$ Maksymalna dopuszczalna liczna iteracji w algorytmie RRT
$f_{l/c}$ Współczynniki określający ilość losowych próbek do ilości używania jako próbkę punkt końcowego w algorytmie RRT

11 Oprogramowanie

Oprogramowanie wytworzone podczas realizacji projektu zostało napisane w języku Python 3.6. z wykorzystaniem modułów: PyGame, collections, random, math oraz csv. Cały kod jest załączony do projektu w dwóch postaciach:

- plikach .pdf, które umożliwiają wydruk kodu i jego archiwizację w formie papierowej. Pliki znajdują się w katalogu "Pliki_z_kodem_do_druku"
- plikach .py oraz .ipynb, które umożliwiają uchronienie oprogramowania za pomocą interpretera Python. Pliki znajdują się w katalogu "Pliki_z_kodem"

Kod został podzielony na kilka plików, ze względu na jego obiektowy charakter. Poszczególne pliki zawierają następujący kod:

- **main.py** - główny plik, który służy uruchomienia wybranego algorytmu RRTOD w celu znalezienia trajektorii. Rezultatem wywołania tego pliku powinno być stworzenie pliku "RRTOD_trajektoria.csv", który zawiera współrzędne węzłów wyznaczonej trajektorii. W pliku "main.pyźnajdują się również parametry mapy i algorytmu wraz z opisami.
- **compare_algorithms.ipynb** - plik pozwalający na porównanie wszystkich algorytmów użytych w pracy
- **RRT.py** - implementacja algorytmu RRT
- **RRTO3P.py** - implementacja algorytmu RRTO3P
- **RRTOD.py** - implementacja algorytmu RRTOD
- **GS.py** - implementacja algorytmu GS
- **GSOD.py** - implementacja algorytmu GSOD
- **Map.py** - implementacja mapy otoczenia wraz z funkcjonalnościami do wizualizacji mapy
- **algorytms.py** - implementacja grafu oraz algorytmu Dijkstra