



ALGORYTMY OPTYMALIZACJI		<i>Prowadzący:</i> Prof. zw. dr hab inż. Czesław Smutnicki		<i>Grupa zajęciowa:</i> K05-00e Środa, 13 ¹⁵ -15 ⁵⁵ TN
PROJEKT				
<i>Skład Grupy</i>				
<i>Imię i nazwisko:</i>	Łukasz Kołtun	Adrianna Szaruga	Olivier Kroll	Jakub Mazur
<i>Nr indeksu:</i>	247361	247294	247354	247379

1. Sformułowanie problemu

Problem przedstawionym do rozwiązania jest stworzenie aplikacji umożliwiającej wyznaczenie optymalnej trasy robota mobilnego. Aplikacja ma umożliwić na stworzenie ścieżki przejazdu robota z zadanego punktu początkowego do zadanego punktu docelowego uwzględniając brak możliwości wystąpienia kolizji z przeszkodami. Od wyznaczonej trasy wymaga się, aby była optymalna (minimalna) ze względu na długość trasy. Roboty mobilne w nawigacji wykorzystują mapę terenu zapisaną w postaci cyfrowej w pamięci urządzenia, niestety najczęściej jest ona niekompletna i wymaga ciągłej aktualizacji. Niekompletność jest spowodowana możliwością wystąpienia przeszkód nieuwzględnionych wcześniej na mapie.

2. Założenia projektowe

Celem projektu jest wykonanie aplikacji pozwalającej na wyszukanie optymalnej (pod względem drogi) ścieżki dla robota uwzględniając występowanie przeszkód. W tym celu wymagane będzie utworzenie odpowiedniej mapy odwzorowującej pole ruchu robota, implementacja oraz obsługa algorytmów służących do wyznaczania optymalnej ścieżki, wizualizacja procesu wyszukiwania ścieżki oraz przygotowanie interfejsu służącego do płynnego i wygodnego użytkowania programu.

3. Podział ról

Rozpatrywany projekt został rozdzielony na mniejsze części i zadania pomiędzy członków grypy projektowej w celu jak najbardziej optymalnej współpracy. Podział ról przedstawiał się następująco:

- Łukasz Kołtun: Przygotowanie oraz zaprogramowanie działania algorytmu D*Lite,
- Jakub Mazur: Przygotowanie oraz zaprogramowanie działania algorytmu LPA*,
- Oliver Kroll: Przygotowanie oraz zaprogramowanie "kreatora map" - modułu projektu pozwalającego na tworzenie, zapisywanie i usuwanie map, po których porusza się robot,
- Adrianna Szaruga: Przygotowanie Interfejsu i obsługi programu głównego.

4. Wykorzystane narzędzia

Do stworzenia aplikacji wykorzystano język Python 3.10 wraz z wieloma paczkami umożliwiającymi łatwiejsze zaimplementowanie poszczególnych możliwości. Paczka Tkinter posłużyła do tworzenia interfejsu graficznego aplikacji oraz do tworzenia mapy dla robota. Wykorzystano również paczkę Click, która umożliwiała na pozycjonowanie poszczególnych elementów mapy za pomocą myszki. Paczką wykorzystaną do zapisu i odczytu mapy z komputera jest paczka Pillow. Do wizualizacji map oraz wyświetlania ścieżki robota została wykorzystana biblioteka Pygame.

4. Model matematyczny algorytmów

Algorytm D* Lite

Pseudokod algorytmu D* Lite:

```
procedure CalculateKey( $s$ )
{01"} return [ $\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))$ ];

procedure Initialize()
{02"}  $U = \emptyset$ ;
{03"}  $k_m = 0$ ;
{04"} for all  $s \in S$   $rhs(s) = g(s) = \infty$ ;
{05"}  $rhs(s_{goal}) = 0$ ;
{06"}  $U.Insert(s_{goal}, [h(s_{start}, s_{goal}); 0])$ ;

procedure UpdateVertex( $u$ )
{07"} if ( $g(u) \neq rhs(u)$  AND  $u \in U$ )  $U.Update(u, CalculateKey(u))$ ;
{08"} else if ( $g(u) \neq rhs(u)$  AND  $u \notin U$ )  $U.Insert(u, CalculateKey(u))$ ;
{09"} else if ( $g(u) = rhs(u)$  AND  $u \in U$ )  $U.Remove(u)$ ;

procedure ComputeShortestPath()
{10"} while ( $U.TopKey() < CalculateKey(s_{start})$  OR  $rhs(s_{start}) > g(s_{start})$ )
{11"}    $u = U.Top()$ ;
{12"}    $k_{old} = U.TopKey()$ ;
{13"}    $k_{new} = CalculateKey(u)$ ;
{14"}   if ( $k_{old} < k_{new}$ )
{15"}      $U.Update(u, k_{new})$ ;
{16"}   else if ( $g(u) > rhs(u)$ )
{17"}      $g(u) = rhs(u)$ ;
{18"}      $U.Remove(u)$ ;
{19"}     for all  $s \in Pred(u)$ 
{20"}       if ( $s \neq s_{goal}$ )  $rhs(s) = \min(rhs(s), c(s, u) + g(u))$ ;
{21"}       UpdateVertex( $s$ );
{22"}   else
{23"}      $g_{old} = g(u)$ ;
{24"}      $g(u) = \infty$ ;
{25"}     for all  $s \in Pred(u) \cup \{u\}$ 
{26"}       if ( $rhs(s) = c(s, u) + g_{old}$ )
{27"}         if ( $s \neq s_{goal}$ )  $rhs(s) = \min_{s' \in Succ(s)} (c(s, s') + g(s'))$ ;
{28"}       UpdateVertex( $s$ );
```

```

procedure Main()
{29"}  $s_{last} = s_{start}$ ;
{30"} Initialize();
{31"} ComputeShortestPath();
{32"} while ( $s_{start} \neq s_{goal}$ )
{33"}   /* if ( $rhs(s_{start}) = \infty$ ) then there is no known path */
{34"}    $s_{start} = \arg \min_{s' \in Succ(s_{start})} (c(s_{start}, s') + g(s'))$ ;
{35"}   Move to  $s_{start}$ ;
{36"}   Scan graph for changed edge costs;
{37"}   if any edge costs changed
{38"}      $k_m = k_m + h(s_{last}, s_{start})$ ;
{39"}      $s_{last} = s_{start}$ ;
{40"}     for all directed edges  $(u, v)$  with changed edge costs
{41"}        $c_{old} = c(u, v)$ ;
{42"}       Update the edge cost  $c(u, v)$ ;
{43"}       if ( $c_{old} > c(u, v)$ )
{44"}         if ( $u \neq s_{goal}$ )  $rhs(u) = \min(rhs(u), c(u, v) + g(v))$ ;
{45"}         else if ( $rhs(u) = c_{old} + g(v)$ )
{46"}           if ( $u \neq s_{goal}$ )  $rhs(u) = \min_{s' \in Succ(u)} (c(u, s') + g(s'))$ ;
{47"}       UpdateVertex( $u$ );
{48"}       ComputeShortestPath();

```

Źródło: „D* Lite” Sven Koenig, Maxim Likhachev, 2002, American Association for Artificial Intelligence

Rola poszczególnych funkcji:

- CalculateKey() – wyznaczanie wartości klucza (kosztu pokonywanej drogi) dla każdego wężła
- UpdateVertex() – aktualizacja informacji o wężle,
- ComputeShortestPath() – wyznaczanie najkrótszej trasy (oblicza i zwraca klucz o najmniejszej wartości)
- Initialize() – inicjalizacja algorytmu oraz kolejki priorytetowej (na podstawie kopca)
- Main() – planowanie, modyfikacja grafu – aktualizacja mapy

s - aktualny wężel

s' - wężel poprzedzający aktualny

$h(s, s_{start})$ - heurystyka - szacuje od dołu odległość wężła s od s_{start}

$c^*(s, s')$ - funkcja szacująca najkrótszą drogę z wężła s do wężła s'

$g(s)$ - wartość określająca najkrótszą odległość wężła s od wężła docelowego

$$g(s) = g(s') + c(s, s')$$

$rhs(s)$ - wartość szacująca długość najkrótszej drogi do wężła s od wężła docelowego

$$rhs(s) = \min (g(s') + c(s, s'))$$

k_m - odległość, jaka robot pokonał od miejsca startu (kumulowana)

s_{start} - punkt początkowy

s_{goal} - punkt docelowy

Opis działania algorytmu:

D* Lite jest algorytmem służącym do wyznaczania ścieżki, który pozwala znaleźć optymalną drogę między punktem początkowym a końcowym w znanym lub częściowo znanym środowisku. Algorytm ten został zaprojektowany do działania na grafowej strukturze danych składającej się z węzłów połączonych liniami wyznaczającymi możliwy ruch między węzłami. Działanie algorytmu polega na porównywaniu dwóch wartości kluczy dla każdego węzła w grafie. Pierwszą uzyskaną wartością jest $g(s)$, które śledzi koszt dotarcia do danego punktu. Jest to najkrótsza ścieżka łącząca punkt początkowy z punktem aktualnym. W przypadku punktu początkowego wynik $g(s)$ wynosi zero, ponieważ punkt aktualny jest również punktem początkowym. Kolejną miarą jest $rhs(s)$, który jest używany w algorytmie do znalezienia kolejnej optymalnej pozycji, do której może „przejsć” ścieżka. Jeśli następna pozycja jest oznaczona jako przeszkoda, to koszt dotarcia do tego miejsca zostanie ustawiony na nieskończoność, co sprawi, że ścieżka będzie omijać ten punkt jako niedozwolony. Używając funkcji $g(s)$ oraz $rhs(s)$, algorytm wyznacza optymalną ścieżkę od punktu końcowego do punktu początkowego. Funkcję $g(s)$ oraz $rhs(s)$ stanowią podstawę planowania i ponownej rekalkulacji ścieżki. W przypadku, gdy D* Lite napotyka nieprzewidziane przeszkody i będzie zmuszony do ponownego zaplanowania ścieżki, wynik $rhs(s)$ jest ponownie przeliczany z nowym kosztem połączenia dla wszystkich węzłów, których to dotyczy.

Algorytm LPA*:

LPA* jest inkrementalną wersją algorytmu A*, przystosowaną do ograniczenia czasu obliczeń w przypadku zmian wartości wierzchołków grafu.

Algorytm ten służy do wyszukiwania optymalnej, pod względem długości, ścieżki pomiędzy dwoma punktami. Dzięki zastosowaniu heurystyki h (przewidzianej odległości węzła od końca), która pomaga ukierunkować poszukiwania, ograniczana jest ilość obliczanych przez algorytm w trakcie pracy węzłów, co skutecznie przyspiesza jego pracę.

Podstawą działania algorytmu jest obliczenie dla kolejnych węzłów wartości rhs i g . Odbywa się to dla węzłów pomiędzy początkowym i końcowym w kolejności opartej o klucz służący do wyznaczenia priorytetu, aż do obliczenia wartości dla węzła końcowego.

Ostateczne wyznaczenie ścieżki odbywa się z punktu końcowego poprzez poruszanie się z obecnego węzła (s) do jego poprzednika (s') minimalizującego wartość funkcji $g(s') + c(s',s)$ [gdzie c oznacza koszt przemieszczenia], aż do osiągnięcia punktu startu, co daje pewność, że uzyskana ścieżka będzie ścieżką optymalną.

Przykład pseudokodu dla algorytmu LPA*:

```

procedure CalculateKey(s)
{01} return [ $\min(g(s), rhs(s)) + h(s)$ ;  $\min(g(s), rhs(s))$ ];

procedure Initialize()
{02}  $U := \emptyset$ ;
{03} for all  $s \in S$   $rhs(s) = g(s) = \infty$ ;
{04}  $rhs(s_{start}) = 0$ ;
{05}  $U.Insert(s_{start}, [h(s_{start}); 0])$ ;

procedure UpdateVertex(u)
{06} if ( $u \neq s_{start}$ )  $rhs(u) = \min_{s' \in Pred(u)} (g(s') + c(s', u))$ ;
{07} if ( $u \in U$ )  $U.Remove(u)$ ;
{08} if ( $g(u) \neq rhs(u)$ )  $U.Insert(u, CalculateKey(u))$ ;

procedure ComputeShortestPath()
{09} while ( $U.TopKey() \neq CalculateKey(s_{goal})$  OR  $rhs(s_{goal}) \neq g(s_{goal})$ )
{10}    $u = U.Pop()$ ;
{11}   if ( $g(u) > rhs(u)$ )
{12}      $g(u) = rhs(u)$ ;
{13}     for all  $s \in Succ(u)$  UpdateVertex(s);
{14}   else
{15}      $g(u) = \infty$ ;
{16}     for all  $s \in Succ(u) \cup \{u\}$  UpdateVertex(s);

procedure Main()
{17} Initialize();
{18} forever
{19}   ComputeShortestPath();
{20}   Wait for changes in edge costs;
{21}   for all directed edges (u, v) with changed edge costs
{22}     Update the edge cost  $c(u, v)$ ;
{23}     UpdateVertex(v);

```

Źródło: Koenig, Sven; Likhachev, Maxim (2001), Incremental A*

- CalculateKey() - obliczanie wartości klucza
- UpdateVertex() - aktualizacja informacji o wierzchołku oraz jego umieszczenie lub usunięcie z kolejki priorytetowej
- ComputeShortestPath() - obliczanie i aktualizuje wartości wierzchołków do momentu gdy nie zostanie obliczony wierzchołek s_{goal} i wartości $rhs(s_{goal})$ i $g(s_{goal})$ nie będą identyczne
- Initialize() - inicjalizacja danych używanych przez algorytm, tworzona jest tu pierwsza nierówność pomiędzy wartościami rhs i g (na początek $rhs(s_{start}) = 0$, $g(s_{start}) = \infty$)
- Main() - główna pętla programu w którym wykorzystywany jest ten algorytm

s - aktualny węzeł

s' - węzeł poprzedzający aktualny

$h(s, s_{goal})$ - heurystyka - szacuje od dołu odległość węzła s od s_{goal}

$c^*(s, s')$ - funkcja szacująca najkrótszą drogę z węzła s do węzła s'

$g(s)$ - wartość określająca najkrótszą odległość węzła s od węzła s_{start}

$$g(s) = g(s') + c(s, s')$$

$rhs(s)$ - wartość szacująca długość najkrótszej drogi do węzła s od węzła s_{start}

$$rhs(s) = \min(g(s') + c(s, s'))$$

s_{start} - punkt początkowy

s_{goal} - punkt docelowy

5. Realizacja projektu

Interfejs aplikacji

Interfejs aplikacji został stworzony z pomocą modułu Tkinter w Pythonie. Po uruchomieniu programu wyświetla się Menu Główne. Pierwszy przycisk "Menu Mapy" otwiera nowe okno z modułem "kreator map", który pozwala na stworzenie, przeglądanie oraz usuwanie map, po których może poruszać się robot.

Przycisk drugi "Menu Algorytmy" otwiera nowe okno pozwalające na działanie na wykorzystanych algorytmach. W prawym górnym rogu wyświetla się czas działania poszczególnych algorytmów oraz różnica pomiędzy nimi, która pozwala na porównanie szybkości ich działania. W menu znajdują się następujące przyciski: „Wybierz mapę”, „Algorytm 1”, „Algorytm 2”, „Wyczyść” oraz „Powrót”. „Wybierz mapę” powoduje zmianę okna na okno wyboru mapy, na podstawie której prowadzona będzie optymalizacja. Po prawej stronie znajduje się lista dostępnych map. Po wybraniu żądanej mapy możliwe jest naciśnięcie jednego z trzech przycisków na dole okna. Przycisk „Podgląd Mapy” wyświetla mapę w podglądzie w głównej części okna. Przycisk „Wybór mapy” pozwala na wybranie mapy i powrót do menu algorytmów gdzie w głównej części okna będzie wyświetlać się wybrana mapa. „Powrót do menu” wraca do menu głównego. Przyciski „Algorytm 1” i „Algorytm2” pozwalają na otworenie nowego okna i wyświetlanie działania algorytmu w czasie rzeczywistym. Przycisk „Wyczyść” czyści całe okno. Przycisk „Powrót” pozwala na zamknięcie okna i wrócenie do Menu Głównego.

Kreator map

Część projektowa „kreator map” składa się z okna reprezentującego „Menu Mapy” posiadającego 4 przycisków funkcyjnych: stwórz mapę, przegląd map, usuń mapę oraz powrót do menu głównego.

Pierwszy przycisk (Stwórz mapę) jest odpowiedzialny za otworzenie okna z generatorem mapy. Proces tworzenia mapy polega na podaniu w pierwszej kolejności wysokości i szerokości mapy (z racji na ilość obliczeń zalecany maksymalny rozmiar mapy wynosi 200x200). Zatwierdzenie parametrów mapy spowoduje stworzenie i wyświetlenie mapy na ekranie. Dalsze tworzenie obiektów będzie aktualizowało wyświetlaną mapę. Kolejnym krokiem jest stworzenie robota oraz punktu docelowego w postaci koła o kolorze zielonym dla robota oraz czerwonym dla punkt docelowego i umiejscowienie ich na mapie. Ostatnim krokiem tworzenia mapy będzie stworzenie i ewentualne usuwanie barier, których zadaniem jest ograniczenie ruchu robota. Tworzenie barier polega na uaktywnieniu odpowiedniej funkcji a następnie zaznaczeniu na mapie punktów, z których zostanie stworzony wielokąt. W programie zastosowano bariery widoczne (kolor szary) i niewidoczne (kolor brązowy). Bariery widoczne są barierami, które robot ma zapisane w pamięci i są one widoczne dla robota od razu przy rozpoczęciu ruchu. Bariery niewidoczne są barierami, które robot wykrywa dopiero po zbliżeniu się do nich, co powoduje zmianę optymalnej trasy. Usuwanie barier polega na identycznym procesie z wyjątkiem zmiany koloru tworzonego wielokąta na biały (taki sam jak tło mapy). Stworzona tak mapa jest przygotowana do zapisu pod podaną

przez użytkownika nazwą. Mapa jest zapisywana w rozszerzeniu .ppm i .png a informacje odnośnie powstałych elementów zapisywane są w postaci pliku .csv.

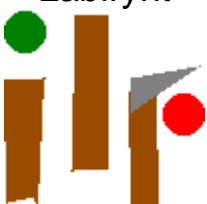

Drugi przycisk (Usuń mapę) tworzy okno, w którym widnieje lista stworzonych już map. W oknie znajduje się odpowiedzialny za wyświetlenie zaznaczonej mapy przycisk, który zgodnie z przeznaczeniem wyświetla w programie zaznaczoną mapę. Występuje również przycisk odpowiedzialny za usunięcie mapy, który zgodnie z przeznaczeniem usuwa mapę w wszystkich formatach (.ppm/.png/.csv). Trzeci przycisk (Przegląd map) tworzy okno identyczne co przycisk odpowiedzialny za usuwanie map z wyjątkiem braku przycisku odpowiedzialnego właśnie za usuwanie. Występuje w tym oknie możliwość jedynie podglądu istniejących map. Czwarty przycisk (Powrót do menu głównego) powoduje powrót do menu głównego, z którego można wybrać inne opcje programu.




Algorytm D*Lite oraz Algorytm LPA*

Oba algorytmy zostały zaimplementowane zgodnie z modelami i pseudokodami zawartymi w opracowaniu (D* Lite” Sven Koenig, Maxim Likhachev, 2002, American Association for Artificial Intelligence). Drogi generowane przez algorytmy wyświetlane są krok po kroku w trakcie symulacji ruchu robota w nieznanym lub częściowo znanym środowisku. W celu porównania wyznaczania optymalnej ścieżki został wprowadzony pomiar czasu obliczeń. Różnica w czasie przeznaczonym do obliczeń wyświetlana jest w oknie interfejsu.

6. Badanie działania algorytmów

W celu zbadania skuteczności funkcjonowania algorytmów dokonano porównania pod względem wymaganego czasu do wygenerowania rozwiązania. Wyniki eksperymentu dla map o różnym ułożeniu przeszkód zapisano w tabeli poniżej:

Mapa	Czas działania LPA*	Czas działania D*Lite	Różnica czasu
<p>Labirynt</p> 	51.574	7.4147	44.127
<p>X</p> 	25.009	17.475	7.534

Widoczne 	54.997	1.281	53.716
Niewidoczne 	125.027	68.120	56.907
Pusty 	2.813	0.675	2.138

Czas pracy LPA został wyznaczony w przypadku, gdy po każdym ruchu robota wykonywana jest rekalkulacja ścieżki. W przypadku, gdy ograniczona zostanie ilość ponownych obliczeń ścieżki poprzez wykonywanie rekalkulacji tylko wtedy, gdy ścieżka przecina niewidoczną przeszkodę uzyskane zostało zdecydowane ograniczenie czasu potrzebnego do wyznaczania optymalnej drogi (w przypadku mapy „widoczne” uzyskano w ten sposób redukcję czasu z 55 do 3,5 sekundy). Gdy za każdym razem wykonywana jest rekalkulacja widać znaczną przewagę działania D* Lite nad LPA*.

7. Wnioski

W ramach tego projektu porównujemy skuteczność D* Lite oraz LPA* jako algorytmów planowania. Zaobserwowaliśmy, że w przypadku ponownego planowania D* Lite jest szybszy od LPA*. Algorytm ten jest nie tylko wydajny w tworzeniu ścieżek, jest też szybszy od innych algorytmów, takich jak A*, D* w ponownym wyznaczaniu ścieżek i omijaniu przeszkód. Poprawna implementacja algorytmu optymalizacyjnego daje pewność, że uzyskane rozwiązanie będzie rozwiązaniem optymalnym. Jednakże nie gwarantuje to krótkiego czasu obliczeń, który zależy także od wybranego sposobu realizacji algorytmu.

D* Lite jest niezwykle przydatny w dziedzinach wymagających pracy w często nieznanym i zmiennym środowisku. Znalazł zastosowanie w robotyce, a konkretnie w autonomicznych robotach mobilnych, np. w przypadku samobieżnych, automatycznych odkurzaczy domowych. D* Lite jako następcę łączący cechy D* i LPA* (inkrementalnej wersji A*) w zadaniach, do których został zaprojektowany radzi sobie lepiej od swoich poprzedników.