POLITECHNIKA WARSZAWSKA

Wydział Mechatroniki

Praca przejściowa

Ireneusz Szulc

Planowanie bezkolizyjnych tras dla zespołu robotów mobilnych

Opiekun pracy:
prof. dr hab. Barbara Siemiątkowska

Warszawa, 2018

Spis treści

Rozdział 1

TODO

Karta tematu:
Temat pracy: Planowanie bezkolizyjnych tras dla zespołu robotów mobilnych
Temat pracy (w jęz. ang.): Path planning for a group of mobile robots

Zakres pracy:

- Projekt algorytmu wyznaczania trajektorii dla pojedynczego robota
- Algorytm detekcji i zapobiegania kolizjom między robotami
- Implementacja oprogramowania symulacyjnego
- Przeprowadzenie testów symulacyjnych

Podstawowe wymagania:

- Aplikacja powinna umożliwiać symulację ruchu robotów oraz definiowanie położenia przeszkód przez użytkownika.
- Planowanie tras dotyczy robotów holonomicznych.

Rozdział 2

Konspekt pracy

- Wstęp teoretyczny:
  - Cooperative Pathfinding
  - algorytm A* - szczegółowo
  - przegląd metod planowania tras dla wielu robotów
  - artykuł o Cooperative Pathfinding, time-space A*
  - artykuł o wyznaczaniu priorytetów i metodach planowania tras (prezentacja): Path Coordination, time-space A*
  - metoda ładunków - problem minimów lokalnych
  - replanowanie po wykruciu kolizcji (algorytm D*)
  - algorytmy WHCA* i IADPP
  - Reciprocal Collision Avoidance
  - metody przydziału priorytetów - zwiększanie i przeliczanie
  - metody zcentralizowane vs rozproszone (porównanie)
  - time-space A*, heurystyki, Reservation Table
- zastosowanie: ciasne korytarze, częsty problem kolizji, szpitale, transport dokumentów, paczek
- generowanie mapy - labiryntu do testów: własny algorytm, teoria grafów, własności mapy
- time-space A* - pseudokod, schemat blokowy, własne heurystyki, modyfikacje
- metoda przydziału / zmiany priorytetów

- ograniczenia - nałożone uproszczenia: ruch skośny, czas dyskretny, brak czasu na obrót
- Implementacja aplikacji - stack technologiczny: Java 8, Java FX, Spring, Spring Boot, testy jednostkowe jUnit, git, IntelliJ, Maven, Linux; schemat klas aplikacji
- obszerne testy, porównanie wyników metod przy tych samych warunkacj początkowych

Rozdział 3

Wstęp

3.1 Cel pracy

Przedmiotem niniejszej pracy jest przegląd metod rozwiązujacych zagadnienie planowania bezkolizyjnych tras dla wielu robotów. Stanowi to również wstęp teoretyczny do zaprojektowania algorytmu i implementacji oprogramowania pozwalającego na symulację działania systemu planowania tras.

Praca skupia się na przypadkach, w których mamy do czynienia ze środowiskiem z dużą liczbą przeszkód (np. zamknięty budynek z licznymi ciasnymi korytarzami), gdzie problem blokowania sią agentów prowadzi często do zakleszczenia. Należy wtedy zastosować nieco inne podejścia niż te, które sprawdzają się w przypadku otwartych środowisk, a które zostały opisane np. w pracach: [**?**], [**?**]

W niniejszej pracy zajmować będziemy się rozwiązaniem problemu, w którym mamy pełną informację o mapie otoczenia oraz o określonym położeniu początkowym i celu dla każdego z robotów. Zadaniem algorytmu będzie wyznaczenie możliwie najkrótszej bezkolizyjnej trasy dla wszystkich robotów. Należy jednak zaznaczyć, że priorytetem jest dotarcie każdego z robotów do celu bez kolizji z innymi robotami, dopiero później chcemy, aby wyznaczone drogi były możliwie najkrótsze.

3.2 Koordynacja ruchu robotów

Koordynacja ruchu robotów jest jednym z fundamentalnych problemów w systemach wielu robotów. [**?**]

Kooperacyjne znajdowanie tras (ang. Cooperative Pathfinding) jest problemem planowania w układzie wieloagentowym, w którym to agenci mają za zadanie znaleźć bezkolizyjne drogi do

swoich, osobnych celów. Planowanie to odbywa się w oparciu o pełną informację o środowisku oraz trasach pozostałych agentów. [?]

Problem kooperacyjnego znajdowania tras pojawia się często m.in. w grach komputerowych, gdzie należy wyznaczyć drogi dla wielu jednostek, które mogą blokować się wzajemnie. Algorytmy do wyznaczania bezkolizyjnych tras dla wielu agentów (robotów) mogą znaleźć również zastosowanie w szpitalach (np. roboty TUG i HOMER do dostarczania sprzętu na wyposażeniu szpitala [?]) oraz magazynach (np. roboty transportowe w magazynach firmy Amazon 3.1).

Rysunek 3.1: Roboty Kiva pracujące w magazynie firmy Amazon. Źródło: `https://www.roboticsbusinessreview.com/supply-chain/amazon_warehouse_demand_devours_robots_and_workers`

Przestrzeń konfiguracyjna

Przestrzeń konfiguracyjna to formalna, matematyczna przestrzeń będąca zbiorem możliwych stanów danego układu fizycznego. W zależności od rodzaju i liczby wyróżnionych parametrów stanu przestrzenie konfiguracyjne mogą mieć wiele wymiarów.

Metoda hill-climbing

Metoda hill-climbing jest rodzajem matematycznej optymalizacji, lokalną metodą przeszukiwania. Jest to iteracyjny algorytm, który zaczyna w wybranym rozwiązaniu problemu, następnie próbuje znaleźć lepsze rozwiązanie poprzez przyrostowe zmiany pojedynczych elementów rozwiązania. Jeśli zmiana przynosi lepsze rozwiązanie, jest wprowadzana do nowego rozwiązania. Kroki algorytmu powtarzane są dotąd, aż żadne "udoskonalenie" nie zostaje znalezione.

Rozdział 4

Metody planowania tras

4.1 Metody planowania tras

Spośród metod wykorzystywanych do planowania tras dla wielu robotów można wyróżnić dwie zasadnicze grupy [**?**]:

- **Zcentralizowane** - drogi wyznaczane są dla wszystkich agentów na raz (jednocześnie). Metody te potrafią znaleźć wszystkie możliwe rozwiązania (w szczególności to optymalne), ale mają bardzo dużą złożoność obliczeniową ze względu na ogromną przestrzeń przeszukiwania. Z tego powodu stosowane są heurystyki przyspieszające proces obliczania rozwiązania.

- **Rozproszone** (ang. *decoupled* lub *distributed*) - Podejście to dekomponuje zadanie na niezależne lub zależne w niewielkim stopniu problemy dla każdego agenta. Dla każdego robota droga wyznaczana jest osobno, w określonej kolejności, następnie rozwiązywane są konflikty (kolizje dróg). W pewnych przypadkach rozwiązanie może nie zostać znalezione, mimo, iż istnieje. Zastosowanie metod rozproszonych wiąże się najczęściej z koniecznością przydzielenia priorytetów robotom, co stanowi istotny problem, gdyż od ich wyboru może zależeć zupełność algorytmu. Nie należy mylić tej metody z zagadnieniem typu *Non-Cooperative Pathfinding*, w którym agenci nie mają wiedzy na temat pozostałych planów i muszą przewidywać przyszłe ruchy pozostałych robotów [**?**]. W podejściu rozproszonym agenci mają pełną informację na temat stanu pozostałych robotów, lecz wyznaczanie dróg odbywa się w określonej kolejności.

4.1.1 Metody zcentralizowane

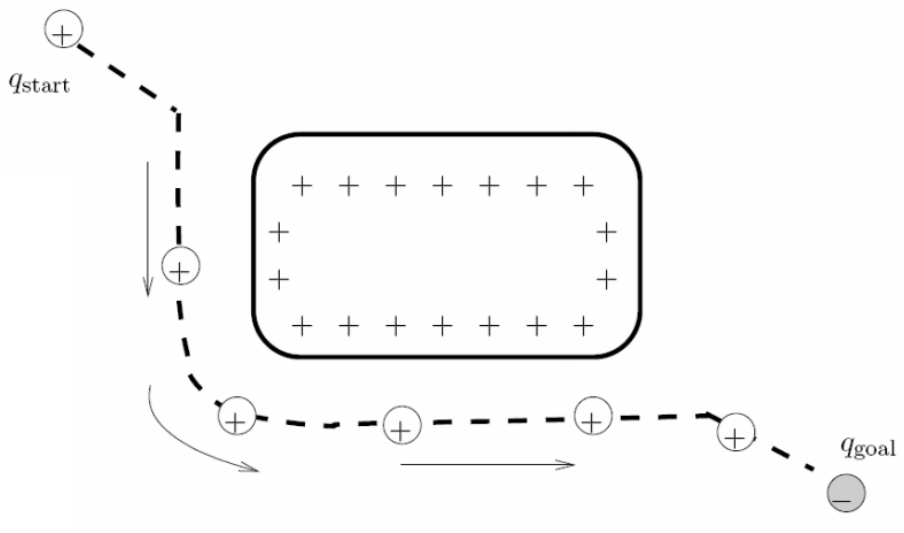
Wiele metod zcentralizowanych cechuje się planowaniem w zbiorowej przestrzeni konfiguracyjnej oraz możliwością wyznaczenia optymalnego rozwiązania. Wadą jest natomiast duża złożoność obliczeniowa algorytmu i konieczność posiadania pełnej informacji o stanie otoczenia i robotów.

W systemach czasu rzeczywistego istotne jest, aby rozwiązanie problemu planowania tras uzyskać w określonym czasie, dlatego w tego typu sytemach częściej używane są techniki rozproszone.

4.1.2 Metoda pól potencjałowych

Nie wszystkie podejścia zcentralizowane gwarantują optymalne rozwiązanie. Przykładem takiej metody, która nie daje gwarancji optymalności (ani nawet zupełności) jest metoda pól potencjałowych.

Metoda pól potencjałowych (ang. *Artificial Potential Field* lub *Potential Field Techniques*) polega na zastosowaniu zasad oddziaływania między ładunkami znanych z elektrostatyki. Roboty i przeszkody traktowane są jako ładunki jednoimienne, przez co "odpychają się" siłą odwrotnie proporcjonalną do kwadratu odległości (dzięki temu unikają kolizji między sobą). Natomiast punkt docelowy robota jest odwzorowany jako ładunek o przeciwnym biegunie, przez co robot jest "przyciągany" do celu. Główną zasadę działania metody przedstawiono na rysunku 4.1. Technika ta jest bardzo prosta i nie wymaga wykonywania złożonych obliczeń (w odróżnieniu do pozostałych metod zcentralizowanych). Niestety bardzo powszechny jest problem osiągania minimum lokalnego, w którym suma wektorów daje zerową siłę wypadkową. Robot zostaje "uwięziony" w minimum lokalnym, przez co nie jest w stanie dotrzeć do wyznaczonego celu. Do omijania tego problemu muszą być stosowane inne dodatkowe metody. [**?**]



Rysunek 4.1: Zasada działania metody pól potencjałowych. Źródło: [?]

4.1.3 Rozproszone wyznaczanie tras

Popularne podejścia unikające planowania w wysoko wymiarowej zbiorowej przestrzeni konfiguracyjnej to techniki rozproszone i priorytetowane. Pomimo, że metody te są bardzo efektywne, mają dwie główne wady:

1. Nie są zupełne - czasami nie udaje się znaleźć rozwiązania, nawet gdy istnieje.

2. Wynikowe rozwiązania są często nieoptymalne.

W artykule [?] przedstawione zostało podejście do optymalizowania układu priorytetów dla rozproszonych i priorytetowanych technik planowania. Proponowana metoda wykonuje randomizowane przeszukiwanie z techniką hill-climbing do znalezienia rozwiązania i do skrócenia całkowitej długości ścieżek. Technika ta została zaimplemenotwana i przetestowana na prawdziwych robotach oraz w rozległych testach symulacyjnych. Wyniki eksperyentu pokazały, że metoda potrafi znacząco zmniejszyć liczbę niepowodzeń i znacznie zmniejszyć całkowitą długość tras dla różnych priorytetowanych i rozproszoncyh metod planowania dróg, nawet dla dużych zespołów robotów.

Najczęściej stosowanymi podejściami są metody oparte o algorytm A*. W szczególności są to:

- A* w konfiguracji czaso-przestrzennej

- Path coordination

Path coordination

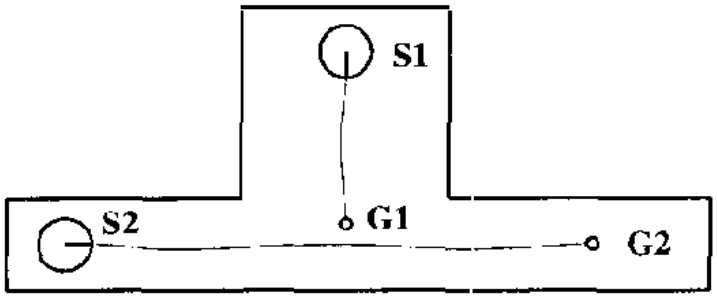
Idea metody Path coordination przedstawia się następująco [**?**]:

1. Wyznaczenie ścieżki dla każdego robota **niezależnie**
2. Przydział priorytetów
3. Próba rozwiązania możliwych konfliktów między ścieżkami. Roboty utrzymywane są na ich indywidualnych ścieżkach (wyznaczonych na początku), wprowadzane modyfikacje pozwalają na zatrzymanie się, ruch naprzód, a nawet cofanie się, ale tylko **wzdłuż trajektorii** w celu uniknięcia kolizji z robotem o wyższym priorytecie.

Złożoność metody wynosi $O(n \cdot m \cdot log(m))$, m - maksymalna liczba stanów podczas planowania, n - liczba robotów

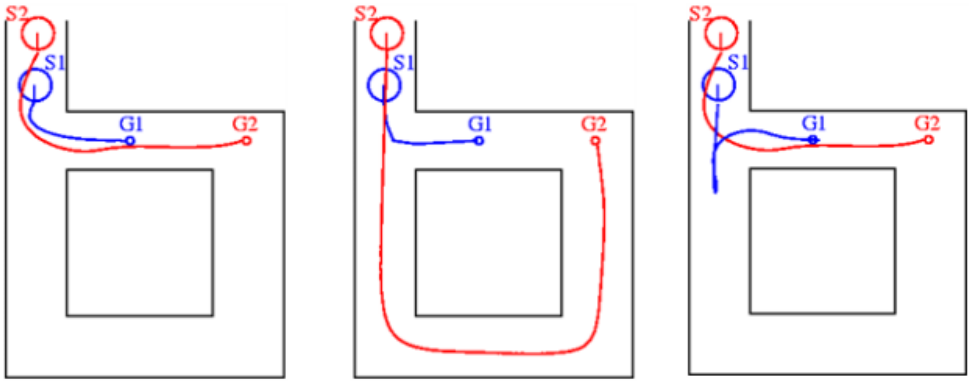
4.1.4 Wybór priorytetów

TODO nie dotyczy tylko Path coordination, ale też A* time-space Istotną rolę doboru priorytetów robotów w procesie planowania tras ukazuje prosty przykład przedstawiony na rysunku 4.2. Jeśli robot 1 (zmierzający z punktu S1 do G1) otrzyma wyższy priorytet niż robot 2 (zmierzający z S2 do G2), spowoduje to zablokowanie przejazdu dla robota 2 i w efekcie prawidłowe rozwiązanie nie zostanie znalezione.



*Rysunek 4.2: Sytuacja, w której żadne rozwiązanie nie zostanie znalezione, jeśli robot 1 ma wyższy priorytet niż robot 2. Źródło: [**?**]*

Układ priorytetów może mieć również zasadniczy wpływ na długość uzyskanych tras. Odpowiedni przykład został przedstawiony na rysunku 4.3. W zależności od wyboru priorytetów, wpływających na kolejność planowania tras, otrzymujemy różne rozwiązania.



Rysunek 4.3: a) Niezależne planowanie optymalnych tras dla 2 robotów; b) suboptymalne rozwiązanie, gdy robot 1 ma wyższy priorytet; c) rozwiązanie, gdy robot 2 ma wyższy priorytet

Rozdział 5

Algorytmy oparte na A*

Kiedy pojedynczy agent dokonuje znalezienia drogi do wyznaczonego celu, prosty algorytm A* sprawdza się bardzo dobrze. Jednak w przypadku, gdy wiele agentów porusza się w tym samym czasie, to podejście może się nie sprawdzić, powodując wzajemne blokowanie się i zakleszczenie jednostek. Rozwiązaniem tego problemu może być kooperacyjne znajdowanie tras. Roboty będą mogły skutecznie przemieszczać się przez mapę, omijając trasy wyznaczone przez inne jednostki oraz schodząc innym jednostkom z drogi, jeśli to konieczne. [**?**]

Zagadnienie znajdowania drogi jest ważnym elementem sztucznej inteligencji zaimplementowanej w wielu grach komputerowych. Chociaż klasyczny algorytm A* potrafi doprowadzić pojedynczego agenta do celu, to jednak dla wielu agentów wymagane jest zastosowanie innego podejścia w celu unikania kolizji. Algorytm A* może zostać zaadaptowany do replanowania trasy na żądanie, w przypadku wykrycia kolizji tras (Local Repair A* lub D*). Jednak takie podejście nie jest zadowalające pod wieloma względami. Na trudnych mapach z wieloma wąskimi korytarzami i wieloma agentami może to prowadzić do zakleszczenia agentów w wąskich gardłach lub do cyklicznego zapętlenia ruchu agentów. [**?**]

5.1 Algorytm A*

A* jest algorytmem heurystycznym służącym do przeszukiwania grafu w celu znalezienia najkrótszej ścieżki. Algorytm ten jest powszechnie stosowany w zagadnieniach sztucznej inteligencji oraz w grach komputerowych [**?**]. Jest modyfikacją algorytmu Dijkstry, wprowadza pojęcie funkcji heurystycznej $h(n)$. Wartość funkcji heurystycznej powinna określać przewidywaną drogę do węzła docelowego. W każdym kroku przeszukiwany jest węzeł o najmniejszej wartości funkcji $f(n)$.

$$f(n) = g(n) + h(n) \tag{5.1}$$

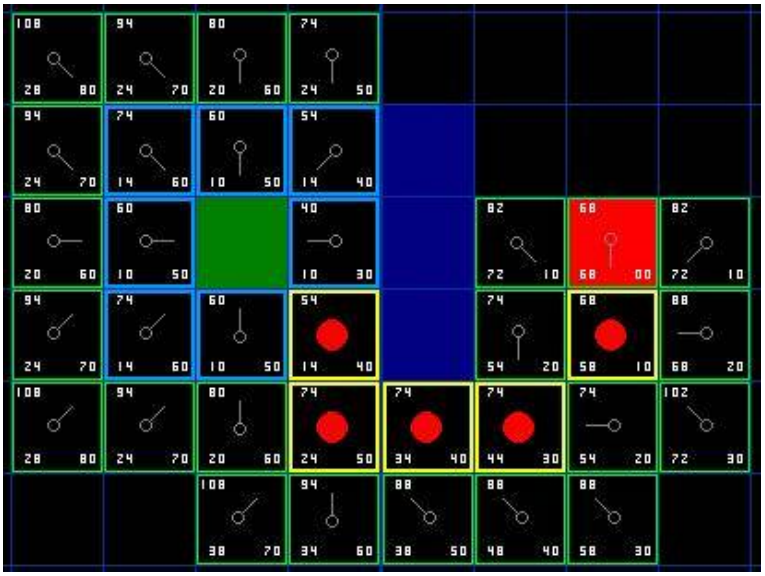
gdzie:

$g(n)$ - wartość kosztu, dokładna odległość miedzy węzłem n a węzłęm startowym

$h(n)$ - heurystyka, przewidywana droga do węzła docelowego

n - węzeł, wierzchołek przeszukiwanego grafu

Dzięki takiemu podejściu najpierw sprawdzane są najbardziej "obiecujące" rozwiązania, co pozwala szybciej otrzymać rozwiązanie (w przeciwieństwie do algorytmu Dijkstry). Algorytm kończy działanie w momencie, gdy napotka węzeł będący węzłem docelowym. Dla każdego odwiedzonego węzła zapamiętywane są wartości $g(n)$, $h(n)$ oraz węzeł będący rodzicem w celu późniejszego odnalezienia drogi powrotnej do węzła startowego po napotkaniu węzła docelowego (por. rys. 5.1).



Rysunek 5.1: Przykład działania algorytmu A. Źródło: [?]*

5.1.1 Heurystyki

Od wyboru sposobu obliczania heurystyki zależy czas wykonywania algorytmu oraz optymalność wyznaczonego rozwiązania. Poniżej przedstawiono najczęściej wykorzystywane heurystyki.

Heurystyka Euklidesowa

Dla dwuwymiarowej mapy heurystyka euklidesowa wyraża dokładną odległość między przeszukiwanym węzłem (x_n, y_n) a węzłem docelowym (x_g, y_g):

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \tag{5.2}$$

Heurystyka Manhattan

W przypadku, gdy robot może poruszać się po mapie jedynie poziomo lub pionowo (nie na ukos) wystarczającym przybliżeniem jest metryka Manhattan:

$$h(n) = |x_n - x_g| + |y_n - y_g| \tag{5.3}$$

Heurystyka zerowa

Przyjęcie heurystyki równej $h(n) = 0$ powoduje, że algorytm A* sprowadza się do algorytmu Dijkstry.

5.2 Local Repair A*

W algorytmie Local Repair A* (LRA*) każdy z agentów znajduje drogę do celu, używając algorytmu A*, ignorując pozostałe roboty oprócz ich obecnych sąsiadów. Roboty zaczynają podążać wyznaczonymi ścieżkami do momentu, aż kolizja z innym robotem jest nieuchronna. Wtedy następuje ponowne przeliczenie drogi pozostałej do przebycia, z uwzględnieniem nowo napotkanej przeszkody.

Możliwe (i całkiem powszechne [**?**]) jest uzyskanie cykli (tych samych sekwencji ruchów powtarzających się w nieskończoność), dlatego zazwyczaj wprowadzane są pewne modyfikacje, aby rozwiązać ten problem. Jedną z możliwości jest zwiększanie wpływu losowego szumu na wartość heurystyki. Kiedy agenci zachowują się bardziej losowo, prawdopodobne jest, że wydostaną się z problematycznego położenia i spróbują podążać innymi ścieżkami.

Algorytm ten ma jednak sporo poważnych wad, które szczególnie ujawniają się w trudnych środowiskach z dużą liczbą przeszkód. Wydostanie się z zatłoczonego wąskiego gardła może trwać bardzo długo. Prowadzi to również do ponownego przeliczania trasy w prawie każdym kroku. Wciąż możliwe jest również odwiedzanie tych samych lokalizacji w wyniku zapętleń.

5.3 Algorytm D*

D* (*Dynamic A* Search*) jest przyrostowym algorytmem przeszukiwania. Jest modyfikacją algorytmu A* pozwalającą na szybsze replanowanie trasy w wyniku zmiany otoczenia (np. zajmowania wolnego pola przez innego robota). Wykorzystywany jest m.in. w nawigacji robota do określonego celu w nieznanym terenie. Początkowo robot planuje drogę na podstawie pewnych założeń (np. nieznany teren nie zawiera przeszkód). Podążając wyznaczoną ścieżką, robot odkrywa rzeczywisty stan mapy i jeśli to konieczne, wykonuje ponowne planowanie trasy na podstawie nowych informacji. Często wykorzystywaną implementacją (z uwagi na zoptymalizowaną złożoność obliczeniową) jest wariant algorytmu *D* Lite* [**?**].

5.4 Cooperative A*

TODO

5.4.1 Tablica rezerwacji

Tablica rezerwacji (ang. *Reservation Table*)

A* w konfiguracji czaso-przestrzennej

TODO Algorytm:

- Przypisanie priorytetów do poszczególnych robotów
- Wykonywanie kroków A* z rozpatrywaniem czasu i przestrzeni - podział otoczenia na siatkę pól, zapisywanie prawdopodobieństwa zajętości pola w danej chwili

5.5 Hierarchical Cooperative A*

TODO

5.6 Windowed Hierarchical Cooperative A*

TODO

Rozdział 6

Podsumowanie

Bibliografia

[1] Bennewitz M.; Burgard W.; Thrun S. *Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems*. 2001.

[2] Hart P. E.; Nilsson N. J.; Raphael B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4, 1968.

[3] Latombe J. *Robot Motion Planning*. Boston, MA: Kluwer Academic, 1991.

[4] Mówiński K.; Roszkowska E. *Sterowanie hybrydowe ruchem robotów mobilnych w systemach wielorobotycznych*. Postępy Robotyki, 2016.

[5] Siemiątkowska B. *Uniwersalna metoda modelowania zachowań robota mobilnego wykorzystująca architekturę uogólnionych sieci komórkowych*. 2009.

[6] Silver D. *Cooperative Pathfinding*. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, 2005.

[7] Koenig S.; Likhachev M. *D* Lite*. Proceedings of the AAAI Conference of Artificial Intelligence, 2002.

[8] Zhu Q.; Yan Y.; Xing Z. *Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing*. Intelligent Systems Design and Applications, 2006.

[9] Roboty TUG i HOMER firmy Aethon. `http://www.aethon.com/tug/tughealthcare/`. Dostęp: 2018-01-02.

[10] Choset H. Robotic motion planning: Potential functions. `https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf`. Dostęp: 2018-01-02.

[11] Searching using a*. `http://web.mit.edu/eranki/www/tutorials/search/`. Dostęp: 2018-01-02.

[12] A* pathfinding for beginners. http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm. Dostęp: 2018-01-02.

Wykaz skrótów

API	Application Programming Interface
SDK	Software Development Kit

Spis rysunków