Akademia Górniczo - Hutnicza

im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Agentowy system wielomodelowy do zarządzania grupą robotów mobilnych

mgr inż. Wojciech Turek

Rozprawa doktorska przygotowana pod kierunkiem
dr hab. inż. Krzysztofa Cetnarowicza prof. n. AGH

Kraków, 2009

Chciałbym bardzo serdecznie podziękować mojemu promotorowi
Panu dr hab. inż. Krzysztofowi Cetnarowiczowi, profesorowi n. AGH
za wszystkie pomysły, rady i wskazówki,
które stanowiły nieocenioną pomoc podczas pisania niniejszej pracy.

Spis treści

1 Wstęp

W ciągu ostatnich dziesięcioleci nastąpił bardzo znaczący rozwój technologii wytwarzania urządzeń mechanicznych, które są w stanie przemieszczać się, postrzegać otaczający środowisko i wykonywać w nim różnego rodzaju czynności. Urządzenia tego typu, zwane robotami mobilnymi, mogą mieć bardzo różnorodne przeznaczenie, które determinuje rodzaj stosowanej konstrukcji, rozmiary, rodzaj napędu czy stosowanych sensorów i efektorów.

Szczególnie duże nadzieje wiąże się z zastosowaniami grup robotów autonomicznych, które będą w stanie samodzielnie wykonywać złożone zadania. Przez autonomię robota rozumie się jego niezależność od elementów zewnętrznych, nie będących częścią robota. Robot może być autonomiczny pod względem źródła zasilania, jednostki sterującej, percepcji otoczenia i możliwości wpływania na otoczenie. Teoretyczne możliwości zastosowań grup współpracujących, autonomicznych robotów mobilnych wydają się niemal nieograniczone. Mogą dotyczyć takich obszarów tak odległych jak eksploracja kosmosu, lub tak codziennych jak sprzątanie supermarketu.

Łatwo jednak zauważyć, że pomimo rosnących możliwości sprzętu, roboty mobilne ciągle nie są obecne w codziennym życiu, a ich wykorzystanie w przemyśle jest bardzo ograniczone. Większość praktycznych zastosowań robotów mobilnych ogranicza się do wykorzystywania jednego, nieskomplikowanego urządzenia, bądź nie wykorzystuje możliwości współpracy wielu robotów przy wykonywaniu wspólnego zadania. Ogranicza to znacząco spektrum możliwych zastosowań.

Przyczyną jest bardzo wysoki stopień skomplikowania niezbędnego oprogramowania sterującego oraz **brak metodologii tworzenia złożonych systemów wielorobotowych**. Każdy, nawet najprostszy program sterujący robotami mobilnymi, musi rozwiązywać problemy z wielu dziedzin, takich jak analiza i przetwarzanie sygnałów z sensorów, komunikacja bezprzewodowa, planowanie i wykonanie trasy, budowanie modelu środowiska czy sterowanie różnego typu efektorami. Złożoność każdego z tych zagadnień spowodowała, ze aktualnie prowadzone badania nad oprogramowaniem robotów skupiają się nad rozwiązywaniem odseparowanych pod-problemów. Istnieje sporo prac poświęconych zagadnieniom nawigacji reaktywnej, budowania mapy czy koordynacji ruchu, które opisują skuteczne algorytmy rozwiązywania poszczególnych zagadnień.

Wdrożenie praktycznych zastosowań robotów mobilnych wymaga jednak, by rozwiązania poszczególnych problemów zostały zintegrowane w system. W niniejszej pracy system złożony z pewnej liczby robotów oraz sterującego nimi oprogramowania będzie postrzegany jako system informatyczny wykorzystujący roboty do wykonywania zadań w świecie rzeczywistym. Każdy system informatyczny musi spełnić szereg wymagań. Podstawowe wymagania opisują stawiane przed systemem zadania, czyli czynności, jakie system ma wykonywać. Są one nazywane wymaganiami funkcjonalnymi. Jednak poza zestawem posia-

danych funkcji, system musi spełniać również szereg wymagań pozafunkcjonalnych, które nie są bezpośrednio związane z celem jego istnienia, ale mają zagwarantować wysoką jakość systemu.

Do najważniejszych wymagań pozafunkcjonalnych, jakie powinien spełniać system informatyczny wykorzystujący roboty mobilne, można zaliczyć:

- Skalowalność, czyli możliwość powiększania systemu o dodatkowe elementy znanego typu. Przykładem powiększania skali systemu może być zwiększanie liczby robotów, czy poszerzanie środowiska ich działania.
- Rozszerzalność, czyli łatwość dodawania nowych funkcji systemu. Rozszerzeniem systemu może być na przykład dodanie nowych rodzajów zadań, które roboty są w stanie wykonywać.
- Wysoka dostępność i odporność na awarie, która gwarantuje nieprzerwane działanie systemu również w sytuacjach wyjątkowych. System powinien być w stanie samodzielnie wykryć i poprawnie zareagować na sytuacje takie jak awaria robota czy błąd komponentu programowego.
- Łatwość ponownego wykorzystywania elementów, która pozwala na szybkie tworzenie nowych rodzajów systemów. Wysoki stopień skomplikowania oprogramowania niezbędnego do sterowania robotami powoduje, że konieczne jest korzystanie z gotowych komponentów. Zmniejsza to znacznie koszty i skraca czas realizacji systemu.

Celem niniejszej pracy jest wykazanie, że przy zastosowaniu odpowiedniej metodologii projektowania i implementowania, możliwe jest stworzenie systemu zarządzającego grupami robotów, który będzie spełniał określone wymagania funkcjonalne oraz będzie się charakteryzował wymienionymi powyżej cechami pozafunkcjonalnymi. Zaproponowana metodologia jest oparta o zastosowanie paradygmatu agentów programowych. Wykorzystuje ona ideę rozdzielenia przestrzeni działania komponentów programowych od przestrzeni działania robotów oraz koncepcję jednoczesnego wykorzystania wielu różnych modeli środowiska działania robotów.

Rozdzielenie abstrakcji agenta programowego od obiektu robota pozwala na zdefiniowanie przestrzeni działania agentów, która jest logicznie niezależna od wykorzystywanych przez system robotów. Przestrzeń wirtualna, w której istnieją i działają agenty, jest tworzona przez platformę agentową, łączącą wiele fizycznych komputerów w jedną maszynę wirtualną. Podejście takie pozwala na uzyskanie pożądanych cech pozafunkcjonalnych. Umożliwia dodawanie niezbędnej agentom mocy obliczeniowej, pozwala na wykorzystanie mechanizmów komunikacji platformy agetnowej do przekazywania wiadomości pomiędzy wszystkimi komponentami systemu. Platforma agentowa pozwala także na dołączanie nowych komponentów bez konieczności zatrzymywania działania istniejących.

Wiele istniejących rozwiązań problemów, związanych z wykorzystaniem robotów, charakteryzuje się wysoką, nieliniową złożonością obliczeniową. Powoduje to trudności ze skalowalnościa rozwiązań i ogranicza maksymalną wielkość systemów. Dotyczy to szczególnie przetwarzania modeli środowiska działania robotów oraz metod zarządzania działaniem licznych grup robotów we wspólnym środowisku. Rozwiązaniem problemu skalowalności tego typu zagadnień może być zaproponowana w pracy metoda wielomodelowa. Polega ona na definiowaniu wielu modeli tego samego aspektu systemu na różnych poziomach abstrakcji. Różne agenty wykorzystują różne modele i są w stanie rozwiązywać problemy na różnym poziomie szczegółowości. Dzięki połączeniu agentów w odpowiednią hierarchię uzyskać można dokładne rozwiązanie każdego z problemów, które jest wyznaczane przez kilka współpracujących agentów.

Przedstawiona w pracy koncepcja projektowania i implementowania systemów informatycznych zarządzających działaniem grup robotów została wykorzystana do stworzenia projektu systemu wykonującego przy pomocy robotów abstrakcyjne zadania. Projekt definiuje trzy podstawowe podsystemu odpowiedzialne za zapewnienie poprawnego działania robotów, nawigację oraz wykonywanie zadań. Specyfikuje rodzaje agentów oraz sposoby interakcji pomiędzy nimi, które gwarantują spełnienie wymagań funkcjonalnych. Analiza własności zaprojektowanego systemu pozwala sądzić, że charakteryzuje się on wszystkimi pożądanymi własnościami pozafunkcjonalnymi.

W celu eksperymentalnego wykazania postawionej w niniejszej pracy tezy, zrealizowany został prototypowy system zarządzający ruchem robotów. Umożliwia on bezpieczne przemieszczanie wielu robotów w rozległym środowisku. Stosuje hierarchiczny, grafowy model środowiska do planowania optymalnych tras oraz algorytm koordynacji do zarządzania ruchem robotów w wybranych fragmentach. W pracy zostanie przedstawiony szczegółowy opis jego implementacji oraz wyniki przeprowadzonych badań, które zostały zrealizowane z wykorzystaniem rzeczywistych robotów oraz systemu do symulacji robotów.

W dalszych częściach niniejszej pracy przedstawione zostaną następujące zagadnienia:

Rozdział 2 stanowi wprowadzenie w zagadnienia związane z programowaniem robotów oraz grup robotów mobilnych. Przedstawione zostaną w nim najczęściej rozważane problemy dotyczących tej dziedziny.

W rozdziale 3 zdefiniowane zostaną wymagania, jakie powinien spełniać wysokiej jakości system informatyczny zarządzający działaniem grup robotów. Opracowanie metody projektowania i implementowania tego typu systemów stanowi cel niniejszej pracy.

Szczegółowa analiza istniejących podejść do tworzenia oprogramowania sterującego robotami zostanie przedstawiona w rozdziale 4. Poruszone zostaną także zagadnienia związane z nawigacją robotów oraz ze sposobami stosowania paradygmatu agentów programowych w programowaniu robotów.

W rozdziale 5 zostanie przedstawiona i omówiona teza pracy.

Rozdział 6 służy przedstawieniu teoretycznych założeń zaproponowanego podejścia do projektowania i implementowana systemów zarządzających robotami. Zostanie w nim omówiona idea rozdzielenia przestrzeni działania agentów i robotów oraz koncepcja podejścia wielomodelowego. W ostatniej części rozdziału zostanie przedstawiona formalna definicja agenta, która będzie podstawą do zdefiniowania typów agentów współpracujących w ramach systemu zarządzającego robotami.

W rozdziale 7 zaprezentowany zostanie projekt kompletnego systemu informatycznego wykorzystującego roboty do realizacji abstrakcyjnych zadań.

Implementacja prototypowego systemu zarządzającego ruchem robotów zostanie przedstawiona w rozdziale 8. Zostaną opisane algorytmy planowania trasy i koordynacji ruchu oraz sposób ich wykorzystania przez poszczególne agenty współpracujące w ramach systemu.

Rozdział 9 zawiera opis przeprowadzonych testów i wyniki badań zrealizowanego systemu.

Uzyskane wyniki, które zostały przedstawione w rozdziale 10, są podstawą do wyciągnięcia wniosków potwierdzających prawdziwość postawionej w pracy tezy.

2 Wprowadzenie w zagadnienia związane z programowaniem robotów mobilnych

Celem badań nad programowaniem robotów mobilnych jest tworzenie systemów wykorzystujących roboty do wykonywania określonych zadań. Wśród najbardziej odpowiednich zadań, jakie mogą być wykonywane przez roboty lub grupy robotów mobilnych, można wymienić takie jak:

- obsługa magazynów,
- automatyczny transport w zakładach przemysłowych,
- sprzątanie,
- inspekcja miejsc niedostępnych dla ludzi, jak kanalizacja czy przewody wentylacyjne,
- ochrona budynków i terenów otwartych,
- przeszukiwanie zbiorników wodnych.

Oprogramowanie sterujące działaniem robotów mobilnych będzie bardzo złożone, nawet w przypadku pozornie niezbyt skomplikowanej funkcjonalności. Tworzenie kompletnego, w pełni funkcjonalnego i niezawodnego systemu wielorobotowego wymaga rozwiązania problemów z wielu odrębnych dziedzin, takich jak:

- programowanie układów cyfrowych,
- komunikacja bezprzewodowa,
- analiza sygnałów analogowych i cyfrowych,
- przetwarzanie i rozpoznawanie obrazu,
- algorytmy grafowe,
- przetwarzanie współbieżne,
- inżynieria oprogramowania,
- sztuczna inteligencja.

Analizując literaturę naukową, poruszającą problemy programowania robotów mobilnych, można zaobserwować tendencję do dzielenia zagadnienia na szereg precyzyjnie zdefiniowanych pod-problemów, z których każdy jest rozważany oddzielnie. Dzieje się tak, ponieważ każdy z aspektów sterowania robotami lub grupami robotów jest sam w sobie

bardzo złożonym problemem. Rozwiązanie każdego z nich wymaga przyjęcia uproszczeń dotyczących pozostałych elementów systemu.

Ze względu na liczbę rozważanych robotów, kierunki prowadzonych badań można podzielić na:

- związane ze sterowaniem pojedynczym robotem,
- związane z zarządzaniem grupą robotów.

2.1 Sterowanie pojedynczym robotem mobilnym

Jednym z najbardziej podstawowych problemów, związanych ze sterowaniem pojedynczym robotem, jest opracowywanie algorytmów nawigacji. Nawigację definiuje się jako dział wiedzy zajmujący się kierowaniem przemieszczającego się obiektu do określonego celu poprzez ustalanie jego lokalizacji oraz planowanie optymalnej trasy [30]. Z pojęciem nawigacji pojedynczego robota mobilnego wiąże się szereg problemów, które w ostatnich dziesięcioleciach były rozważane w licznych pracach naukowych.

Najprostszymi realizacjami nawigacji pojedynczego robota są metody autonomicznej nawigacji reaktywnej. Są to algorytmy bezstanowe, które nie gromadzą ani nie wykorzystują wiedzy o całym środowisku działania. Na podstawie informacji na temat własnego położenia, która jest najczęściej uznawana za daną, mają za zadanie przemieścić się do zadanego celu. Do unikania kolizji z przeszkodami wykorzystują informacje o bieżącym otoczeniu, które najczęściej są pozyskiwane z ultradźwiękowych lub laserowych sensorów odległości.

W przypadku, gdy algorytm ma do dyspozycji model środowiska (zwany najczęściej mapą), rozwiązać trzeba problem lokalizacji robota, czyli wyznaczania jego pozycji i orientacji względem modelu. Wykorzystywane do tego celu są informacje dostarczane przez sensory odległości lub kamery. Kolejnym problemem jest wyznaczanie trasy lub trajektorii z wykorzystaniem dostępnego modelu.

Budowanie modelu środowiska może być również zadaniem robota. Jeżeli ma on działać w środowisku nieznanym i zmiennym w czasie, konieczne jest opracowanie metod jego rozpoznawania i reagowania na zachodzące w nim zmiany.

Zagadnieniem, które w ostatnim dziesięcioleciu było rozważane prawdopodobnie najczęściej, jest problem jednoczesnej lokalizacji i budowania mapy [74] (ang. Simultaneous Localization And Mapping – SLAM). Robot jest umieszczany w nieznanym środowisku i ma za zadanie jednoczesne budowanie modelu tego środowiska oraz ustalanie własnej lokalizacji w zbudowanym modelu. Jakość rozwiązania ocenia się na podstawie dokładności skonstruowanego modelu oraz rozmiarów środowiska, jakie metoda może przetworzyć.

2.2 Zarządzanie grupą robotów

Wśród problemów biorących pod uwagę współistnienie we wspólnym środowisku wielu robotów, również największy nacisk jest kładziony na zagadnienia związane z nawigacją. Stosunkowo nieskomplikowane i deterministyczne zadanie planowania optymalnej trasy pojedynczego robota staje się złożonym problemem optymalizacji w przypadku konieczności wyznaczenia tras dla wielu robotów, które będą gwarantowały bezpieczeństwo ruchu. Problem ten, zwany koordynacją ruchu, może mieć wiele wariantów. Kryterium optymalności zaplanowanych tras może być czas ich przebycia, długość czy ilość potrzebnej energii. Informacje o planach poszczególnych robotów mogą być znane i stałe, ale mogą również pojawiać się w trakcie wykonywania tras przez inne roboty.

Innym, często rozważanym problemem, jest przemieszczanie się robotów w formacjach [9]. Wzajemne położenie robotów podczas przemieszczania podlega określonym regułom (np. ma pozostać niezmienione), również w przypadku napotkania niespodziewanych przeszkód. Odmianą tego zadania są tzw. tańce robotów, czyli wykonywane przez grupę robotów sekwencje przemieszczeń, które są zsynchronizowane z podkładem muzycznym.

Przykładem zagadnienia niezwiązanego bezpośrednio z nawigacją jest problem przydziału zadań dla robotów [83]. Algorytm ma do dyspozycji grupę robotów oraz listę zadań do zrealizowania, z których każde jest skojarzone z lokalizacją w środowisku. Celem jest wyznaczenie optymalnego przydziału poszczególnych robotów do zadań, tak by sumaryczny czas ich wykonania był minimalny. Zadania mogą wymagać do realizacji jednego lub wielu robotów, mogą być znane, lub odkrywane przez przemieszczające się roboty.

Często rozważanym problemem jest również przeszukiwanie terenu przez robota lub grupę robotów [76]. Każdy robot ma określony zasięg postrzegania otoczenia, a celem działania jest takie przemieszczanie, by całe środowisko zostało pokryte. Przykładami zastosowania może być poszukiwanie osób lub przedmiotów w budynkach czy też sprzątanie pomieszczeń. Na pozór podobnym zagadnieniem jest problem ochrony terenu [1], w którym występują dwa rodzaje robotów: ochroniarze i intruzi. Ochroniarze patrolują określony fragment środowiska, a intruzi mają za zadanie przemieścić się do zadanego miejsca w chronionym obszarze, bez kontaktu z ochroniarzami. Co ciekawe, problem ten jest rozważany w sposób czysto teoretyczny, w oparciu o teorię gier. Na podstawie analizy modelu środowiska obliczana jest strategia ochroniarzy oraz intruzów i wyliczane jest prawdopodobieństwo sukcesu każdej ze stron.

Istnieje wiele rozwiązań wymienionych problemów. Charakteryzują się one określonymi właściwościami i ograniczeniami, w określonych warunkach mogą być stosowane. Trzeba jednak pamiętać, że są to oddzielnie rozważane algorytmy, rozwiązujące abstrakcyjnie sformułowane problemy, które nie biorą pod uwagę wielu praktycznych aspektów i wymagań związanych z tworzeniem wysokiej jakości systemów.

Próby zdefiniowania bardziej złożonych zadań, których rozwiązania wymagałyby integrowania kilku z wymienionych powyżej problemów, są realizowane przez organizacje takie jak FIRA [36] oraz RoboCup [68]. Definiują one precyzyjne specyfikacje urządzeń i zasady obowiązujące podczas rozgrywek, prowadzonych pomiędzy różnymi drużynami robotów. Podstawową dyscypliną, w której roboty współzawodniczą jest piłka robotów, wzorowana na zasadach obowiązujących w piłce nożnej. Różne ligi stosują roboty różnej wielkości i o różnych możliwościach – od miniaturowych, dwukołowych robotów o napędzie różnicowym, po roboty humanoidalne. Inną dyscypliną jest współzawodnictwo robotów ratunkowych, które mają za zadanie poszukiwanie ofiar katastrof budowlanych czy trzęsień ziemi [40]. Stworzenie drużyny startującej w tego typu zawodach wymaga zintegrowania rozwiązań problemów lokalizacji, przetwarzania danych z sensorów, planowania, współpracy i koordynacji.

Złożoność problemów związanych z programowaniem robotów mobilnych spowodowała wyodrębnienie i oddzielenie poszczególnych aspektów tej dziedziny. Trzeba jednak pamiętać, że konieczne jest opracowanie metodologii i standardów integrowania rozwiązań poszczególnych pod-problemów w spójne systemy, charakteryzujące się wysoką jakością. Dopiero wtedy możliwe będzie wyjście poza eksperymentalne realizacje i praktyczne stosowanie grup robotów mobilnych w przemyśle czy codziennym życiu.

3 Własności systemu sterującego grupą robotów mobilnych

Ogromny nakład pracy zrealizowanej w ciągu ostatnich dziesięcioleci przez naukowców i inżynierów pozwolił na rozwiązanie wielu problemów związanych z zastosowaniami robotów mobilnych. Łatwo jednak zauważyć, że roboty mobilne, a w szczególności grupy współpracujących robotów, nie są jeszcze na co dzień obecne w naszym życiu, ani nawet w zastosowaniach przemysłowych. Wydaje się, że przejście od realizacji demonstracyjnych i eksperymentalnych do praktycznych zastosowań i wdrożeń wymaga, by istniejące rozwiązania poszczególnych problemów związanych z wykorzystaniem robotów mobilnych zostały zintegrowane w **system informatyczny**.

System to zbiór powiązanych ze sobą elementów i zachodzących miedzy nimi relacji [58]. System informatyczny definiuje się jako system mający na celu przetwarzanie informacji przy pomocy technik komputerowych [31]. Na system informatyczny składają się:

- sprzęt,
- oprogramowanie,
- zbiory danych
- urządzenia do komunikacji,
- oraz zasoby osobowe i procedury związane z użytkowaniem systemu.

Sposobami analizy, projektowania i implementowania systemów informatycznych zajmuje się inżynieria oprogramowania [69]. Wyróżnia ona dwie grupy wymagań, które spełnić musi każdy system informatyczny:

- wymagania funkcjonalne, które definiują cel istnienia systemu,
- wymagania pozafunkcjonalne (zwane też niefunkcjonalnymi), które definiują warunki realizacji tego celu.

Wymagania funkcjonalne, stawiane przed systemem zarządzającym działaniem robotów, będą opisywały rodzaje czynności i zadań, jakie mają być przez roboty wykonywane. Każdy system będzie definiował różne wymagania funkcjonalne. Wymagania pozafunkcjonalne będą opisywały charakterystykę i właściwości systemu, określały jego niezawodność, wydajność, możliwość rozbudowywania i adaptowania do nowych zadań. Dla większości systemów stosujących roboty będą bardzo podobne, i dlatego warto się na nich skupić podczas definiowania metodologii tworzenia tego typu systemów.

Najważniejsze właściwości pozafunkcjonalne, którymi charakteryzować się powinien każdy zaawansowany system informatyczny, to:

- skalowalność,
- rozszerzalność,
- wysoka dostępność, odporność na awarie,
- łatwość ponownego wykorzystywania elementów.

3.1 Skalowalność

Skalowalność systemu jest definiowana jako zdolność do przetwarzania coraz większej ilość zadań [20]. System może w tym celu wymagać większej ilości zasobów, jednak nie powinien w znaczący sposób ograniczać maksymalnej ilości zadań, jaką jest w stanie przetworzyć w danej jednostce czasu. Oczywiście w praktyce zawsze istnieją limity, jednak powinny być one maksymalnie duże i znane. Przykładem czynnika, który może ograniczać skalowalność (tzw. wąskie gardło) może być na przykład wydajność stosowanych interfejsów komunikacyjnych. Rekonfiguracja systemu (dołączenie zasobów) nie powinna mieć wpływy na jego pracę, w szczególności nie powinna wymagać zatrzymywania systemu. Skalowalność można mierzyć za pomocą ilości (wielkości) zasobów wymaganych do przetworzenia danej ilości (złożoności) zadań.

W przypadku systemu wykorzystującego roboty mobilne skalowalność będzie najczęściej związana ze zwiększaniem liczby stosowanych robotów oraz liczby zadań realizowanych przez te roboty w jednostce czasu. System musi być w stanie zaakceptować nowe roboty bez przerywania działania już obecnych, oraz wykorzystać je w sposób optymalny. Innym aspektem skalowalności będzie rozszerzanie środowiska działania robotów. Od konkretnych rozwiązań technologicznych będzie zależało, jak skomplikowany będzie proces rozbudowania mapy, jednak zawsze powinien być on wykonalny.

Warto zwrócić uwagę na fakt, że z punktu widzenia skalowalności kluczowym elementem systemu może okazać się metoda organizacji ruchu robotów. Próba zwiększenia wydajności systemu poprzez dodanie nowych robotów może doprowadzić do efektów odwrotnych od zamierzonych. Wzrost zagęszczenia robotów pracujących na tym samym terenie będzie zawsze powodowało powstawanie większej liczby sytuacji kolizyjnych podczas przemieszczania robotów, co spowoduje wydłużenie czasu przemieszczania każdego z robotów. Ważne, by stosowana metoda zarządzania ruchem robotów zabezpieczała system przed zakleszczeniami i gwarantowała stabilne i wydajne przemieszczanie przy określonym zagęszczeniu.

3.2 Rozszerzalność

Kolejną, prawdopodobnie najważniejszą cechą systemu informatycznego, który jest projektowany z myślą o długotrwałym funkcjonowaniu, jest rozszerzalność. Istnieje wiele prac próbujących zdefiniować rozszerzalność oraz zidentyfikować metody jej osiągania [62].

Najczęściej przyjmuje się, że rozszerzalność to obecność w systemie mechanizmów przeznaczonych wyłącznie do ułatwiania dodawania nowych elementów i funkcjonalności. Miarą rozszerzalności może być wymagany koszt dodania konkretnego rozszerzenia, rozumiany zarówno jako wartość finansowa implementacji, czas realizacji, niezbędny sprzęt itp. Inne cechy brane pod uwagę to wpływ, jaki nowa funkcjonalność ma na wcześniejsze cechy systemu, czy też możliwość wprowadzenia zmian bez zatrzymywania działania elementów systemu.

Zapewnianie rozszerzalności systemu informatycznego jest osiągane poprzez przygotowywanie różnego rodzaju interfejsów ułatwiających dostęp programowy do elementów systemu. Podczas tworzenia interfejsów nie zakłada się, jakie konkretne zmiany będą w przyszłości wprowadzane, ale raczej definiuje się kierunki rozwoju i pokrywa się kompletnie całość funkcjonalności, którą można bezpiecznie udostępnić.

W przypadku systemów zarządzających działaniem grup robotów mobilnych można wyróżnić następujące kierunki rozwoju:

- Dodawanie nowych typów zadań realizowanych przez roboty – powinna istnieć możliwość dodania algorytmów wykorzystujących roboty obecnie w systemie do wykonania nowych typów operacji.
- Dodawanie nowych typów robotów – nowe roboty powinny być wykorzystywane do wykonywania zadań realizowanych wcześniej przez system, jeśli tylko jest to fizycznie możliwe.
- Rozbudowa systemu o infrastrukturę wymaganą do prawidłowego funkcjonowania nowych typów robotów, na przykład urządzenia do ładowania ich akumulatorów.
- Modyfikacja lub zmiana stosowanych algorytmów planowania tras i koordynacji ruchu – tego typu zmiany również powinny być wprowadzane bez zatrzymywania systemu.
- Rozbudowa środowiska działania robotów o fragmenty, które nie były wcześniej znane, w tym fragmenty zawierające nowe typy urządzeń wykorzystywanych do przemieszczania – przykładem mogą być drzwi, windy, pasy transmisyjne czy inne urządzenia, które wcześniej nie były wykorzystywane w systemie.
- Modyfikacja algorytmów przydziału zadań wykonywanych przez roboty.

Można stwierdzić, że najłatwiej jest zapewnić rozszerzalność w systemach rozproszonych, w których poszczególne elementy działają asynchronicznie i nie są ze sobą ściśle powiązane. Podczas projektowania tego typu systemu trzeba przewidzieć sytuację, w której niektóre jego elementy czasowo przestają funkcjonować. Pozwala to na podmianę fragmentów systemu bez konieczności zatrzymywania pozostałych elementów.

3.3 Wysoka dostępność

Wysoka dostępność systemu zwykle może być utożsamiana z odpornością na różnego typu awarie. W przypadku systemów wielorobotowych należy się spodziewać znacznie większej ilości awarii i problemów ze sprzętem, niż w przypadku typowych systemów informatycznych, w których nie występuje skomplikowany sprzęt, taki jak roboty mobilne. Awaria robota może nastąpić w dowolnym momencie jego działania. System powinien być w stanie poprawnie obsłużyć tego typu sytuacje, przy czym obsłużenie awarii składa się z dwóch elementów. Pierwszym jest poprawne wykonanie zadania, które uszkodzony robot realizował – konieczne jest wyznaczenie innego robota lub innej metody realizacji zadania. Drugi to przywrócenie sprawności samego robota, które może ograniczyć się do poinformowania administratorów o zaistniałej sytuacji, lub, w bardziej zaawansowanych rozwiązaniach, może zostać wykonane przez inne roboty.

Analizując problem dostępności systemu należy również zwrócić uwagę na fakt, iż aktualnie konstruowane eksperymentalne rozwiązania często nie biorą pod uwagę potrzeb samych robotów. Przed przystąpieniem do uruchomienia eksperymentu wszystkie roboty są sprawne i naładowane, a czas trwania testów jest na tyle krótki, by nie zachodziła sytuacja awaryjna. Oczywiście w systemach produkcyjnych, które miałyby funkcjonować nieprzerwanie przez dłuższy okres czasu problem zapewnienia i wykorzystania zautomatyzowanej infrastruktury serwisowej musi zostać rozwiązany.

3.4 Możliwość wielokrotnego wykorzystania elementów

Rosnące koszty tworzenia coraz bardziej złożonych systemów informatycznych wymusiły powstanie metodologii projektowania i implementowania elementów systemów w taki sposób, by możliwe było ich ponowne wykorzystanie. Podejście komponentowe oraz projektowanie orientowane na usługi są obecnie bardzo szeroko stosowane i dają bardzo wymierne efekty. Odpowiednio bogata kolekcja istniejących komponentów i usług umożliwia szybkie tworzenie zaawansowanego oprogramowania, które dodatkowo charakteryzuje się wysoką jakością (elementy są dokładnie przetestowane), zgodnością ze standardami i łatwością integracji z innymi systemami.

W tym aspekcie systemy wielorobotowe, jako złożone systemy informatyczne, powinny również korzystać z wniosków i standardów opracowanych w dziedzinie inżynierii oprogramowania. Nawet bez zagłębiania się w konkretne zastosowania robotów można wskazać wiele typowych problemów, których rozwiązania mogłyby zostać zaimplementowane jako komponenty czy usługi. Przykładami mogą być algorytmy analizy obrazu i innych sygnałów z sensorów, metody lokalizacji czy algorytmy nawigacji reaktywnej.

Zaprezentowane cechy i właściwości definiują cel, którego osiągnięcie powinny gwaran-

tować metodologie tworzenia systemów zarządzających działaniem robotów. W następnym rozdziale zaprezentowany zostanie przegląd istniejących architektur systemów sterujących robotami i grupami robotów. Zostaną one przeanalizowane pod kątem opisanych powyżej cech i właściwości w celu stwierdzenia, czy możliwe jest ich wykorzystanie w tworzeniu wysokiej jakości systemów informatycznych, wykorzystujących roboty mobilne.

4 Istniejące podejścia do problemu sterowania grupami robotów mobilnych

Wydawać by się mogło, że po kilkudziesięciu latach prac nad możliwościami zastosowań robotów mobilnych, metody budowania złożonych systemów wielorobotowych będą bardzo dokładnie rozpoznane i opracowane. Okazuje się jednak, że prac w tej dziedzinie jest bardzo niewiele. Przyczyn należy dopatrywać się w sposobie rozwoju tej dyscypliny – punktem wyjścia był zawsze pojedynczy robot, który wykonywał jakiś rodzaj zadań. Sposoby realizacji algorytmów sterujących pojedynczym robotem zostały podzielone i sklasyfikowane. Gdy zaczęły pojawiać się koncepcje wykorzystania wielu robotów, istniejące sposoby realizacji programów sterujących zaczęto adaptować do potrzeb systemów wielorobotowych. Takie podejście zaowocowało powstaniem wielu eksperymentalnych systemów dedykowanych do wykonywania konkretnych zadań przy pomocy ściśle określonego sprzętu.

Warto przeanalizować istniejące rozwiązania, nawet jeśli nie stawiają sobie one za cel opracowania zasad tworzenia systemów wielorobotowych. Wnioski z przeprowadzonych dotychczas eksperymentów i implementacji mogą pomóc w uogólnieniu cech, zrozumieniu ograniczeń i zidentyfikowaniu metod uzyskiwania pożądanych właściwości systemów sterujących grupami robotów.

Szczególnie interesującym kierunkiem badań nad sposobami programowania robotów oraz architekturami zarządzającymi działaniem grup robotów, jest zastosowanie agentów programowych oraz systemów agentowych. Powiązanie tych dwóch dziedzin, bardzo wyraźnie widoczne w licznych pracach, jest spowodowane podobieństwem cech przypisywanych agentom programowym i robotom mobilnym, takich jak autonomia, mobilność czy zdolność do komunikacji. Celem niniejszej pracy jest wykazanie, że odpowiednie zastosowanie paradygmatu agentowego pozwoli na stworzenie systemu charakteryzującego się pożądanymi właściwościami, dlatego analiza istniejących podejść do zarządzania robotami przy pomocy agentów zostanie przeprowadzona szczególnie starannie.

Grupa problemów, którym w rozważanej dziedzinie poświęcono do tej pory najwięcej uwagi, jest związana z przemieszczaniem robotów mobilnych. Konkretne zadania wykonywane przez roboty są ściśle uzależnione od rodzaju wykorzystywanych efektorów. Mają one charakter problemów specjalistycznych zależnych od poszczególnych zastosowań. Natomiast problemy związane ze sterowaniem kołami czy kończynami, planowaniem tras czy koordynacją ruchu muszą zostać rozwiązane w każdym systemie wykorzystującym roboty mobilne i wymagają rozwiązania problemów ogólnych, wspólnych dla wszystkich systemów wykorzystujących roboty mobilne. Rozwiązania tych problemów muszą spełniać stawiane wymagania pozafunkcjonalne, jeśli wykorzystujący je system ma charakteryzować się wysoką jakością. Dlatego warto przeanalizować sposoby realizacji nawigacji robotów, a w szczególności przyjrzeć się systemom koordynującym przemieszczanie wielu robotów.

4.1 Architektury systemów sterujących robotami

4.1.1 Taksonomie architektur

Systemy sterujące działaniem grup robotów mobilnych, czyli tak zwane systemy wielorobotwe (ang. Multi-Robot System, MRS) były, w ciągu ostatnich dziesięcioleci, wielokrotnie opisywane pracach naukowych. Badacze próbowali zastosować grupy współpracujących robotów do rozwiązania różnych problemów i wykonania różnego typu zadań. W zależności od zastosowania, rodzaju robotów oraz wcześniejszych doświadczeń twórców, powstawały najróżniejsze architektury systemów sterujących robotami.

Ta różnorodność podejść skłoniła kilka zespołów badawczych do przeprowadzenia porównań własności różnych architektur. Powstały dzięki temu zestawienia różnego typu rozwiązań oraz różne taksonomie systemów wielorobotowych.

Jednym z najstarszych tego typu podziałów jest taksonomia zaproponowana przez G. Dudka w [28]. Wprowadza on podział systemów wielorobotowych ze względu na:

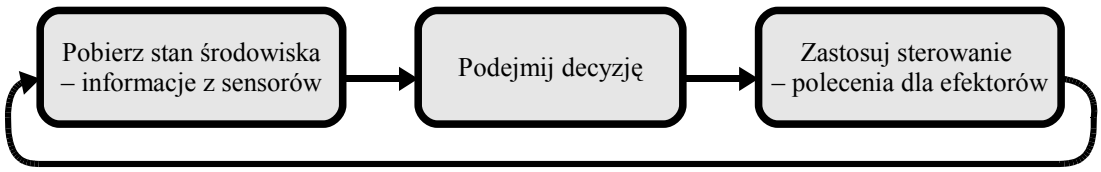
- rozmiar grupy robotów, wyróżniając systemy sterujące pojedynczym robotem, dwoma robotami, grupą o określonej liczności oraz grupą o nieograniczonej liczności,
- zasięg komunikacji, który może być ograniczony lub nie,
- topologię komunikacji, opisującą sposób przekazywania wiadomości pomiędzy robotami (rozgłaszanie, adresacja, grafy i drzewa przekazywania wiadomości),
- przepustowość komunikacji, która może być ograniczona lub nie,
- zdolność do zmian w konfiguracji grupy robotów, czyli liczby i rodzaju robotów w grupie,
- możliwości obliczeniowych poszczególnych robotów,
- różnic pomiędzy robotami w grupie, wyróżniając grupy robotów identycznych, homogenicznych (rozumianych jako posiadające tą samą charakterystykę), oraz heterogenicznych.

Wprowadzone kryteria charakteryzują rodzaj wykorzystywanego sprzętu i ograniczenia wynikające z charakterystyki środowiska. Pozwalają na rozróżnienie rodzajów grup robotów, ale nie architektur oprogramowania sterującego tymi robotami. Z całą pewnością taka taksonomia nie będzie przydatna podczas analizowania sposobów tworzenia systemów informatycznych zarządzających robotami.

Jednym z częściej proponowanych podziałów algorytmów sterujących robotami jest wyróżnienie dwóch podejść:

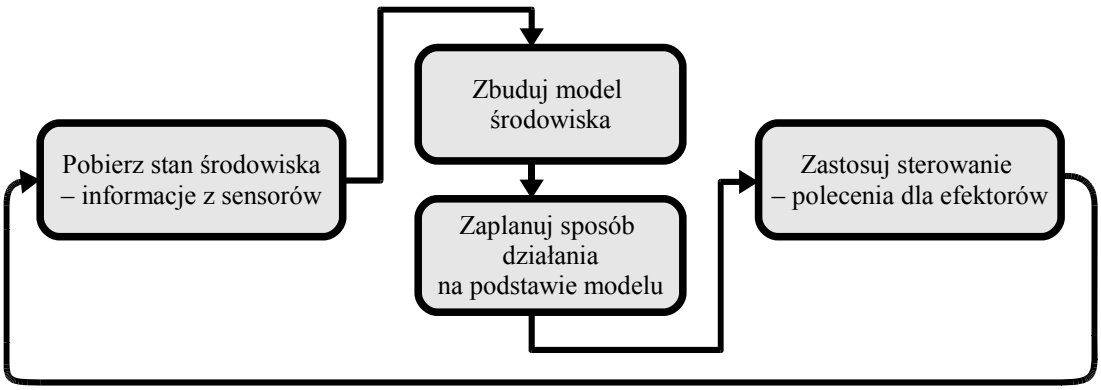
- celowego (ang. deliberative),
- reaktywnego.

Podział możliwych podejść na reaktywne i celowe pozwala wyróżnić dwa sposoby realizacji ogólnej pętli sterowania, której schemat jest widoczny na rysunku 4.1.



Rysunek 4.1: Ogólny cykl działania programu sterującego robotem.

W algorytmie realizującym podejście celowe, cykl sterowania składa się z czterech kroków (rysunek 4.2). Pierwszy krok to pobranie informacji z sensorów, ostatni to zastosowanie obliczonego sterowania. Dwa kroki, składające się na podejmowanie decyzji, to budowanie modelu środowiska i planowanie sposobu działania na podstawie stanu tego modelu. Rodzaj modelu, czyli sposób reprezentowania środowiska, jest uzależniony od wykorzystywanego algorytmu planowania. Zbudowany model jest przekazywany do modułu planowania, który ma za zadanie wyznaczenie szeregu operacji umożliwiających osiągnięcie pożądanego celu. Po wyznaczeniu poprawnej sekwencji operacji, zwanej planem, polecenia muszą zostać przetłumaczone na rozkazy dla efektorów robota.



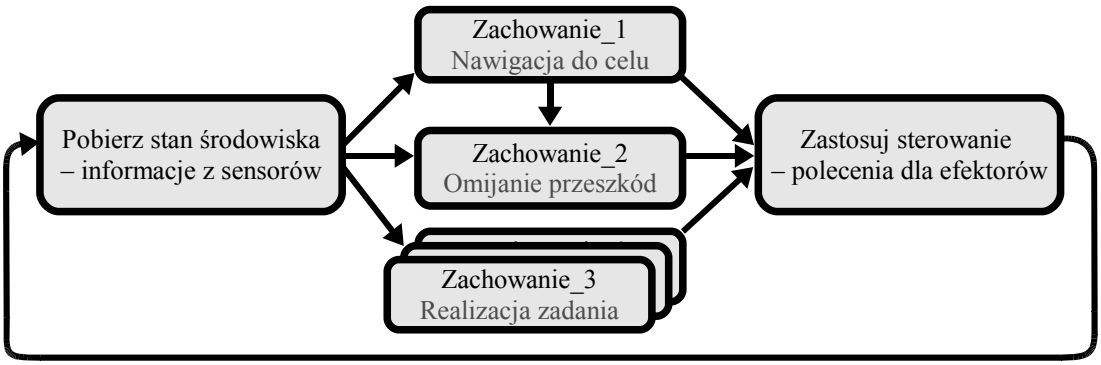
Rysunek 4.2: Cykl działania programu sterującego robotem, stosujący podejście celowe.

Sterowniki celowe umożliwiają realizację bardzo złożonych zadań. Przetwarzanie skonstruowanego modelu rzeczywistości pozwala na precyzyjne określenie zachowania robota w każdej możliwej sytuacji. Wymagają jednak relatywnie dużej ilości pamięci i mocy ob-

liczeniowej. Jedną z głównych wad tego typu podejścia jest długi czas reakcji na zmiany, spowodowany koniecznością sekwencyjnego wykonania etapów algorytmu sterującego.

W odróżnieniu od podejścia celowego, algorytm reaktywny nie konstruuje w pamięci modelu rzeczywistości, a jedynie reaguje w zaprogramowany sposób na wykryte zmiany otoczenia. Informacji o zmianach dostarczają sensory robota. Tego typu podejście zostało zaproponowane przez Profesora R. Brooksa. W pracy [22] zaprezentował on nowy sposób konstruowania programu sterującego robotem mobilnym, który składał się ze stosunkowo niezależnych algorytmów realizujących różne aspekty działania robota. Podejście to jest obecnie powszechnie nazywane behawioralnym [56].

Moduł sterownika behawioralnego składa się z implementacji różnych zachowań, które są reakcją na określony stan postrzeganego środowiska. Każde zachowanie ma bezpośredni dostęp do odczytów z sensorów robota, każde może podjąć decyzję o optymalnym sterowaniu efektorów (rysunek 4.3). Aktywacja poszczególnych zachowań jest realizowana przez zestaw reguł. Niektóre zachowania mogą aktywować inne, przekazując wyniki przeprowadzonych obliczeń.



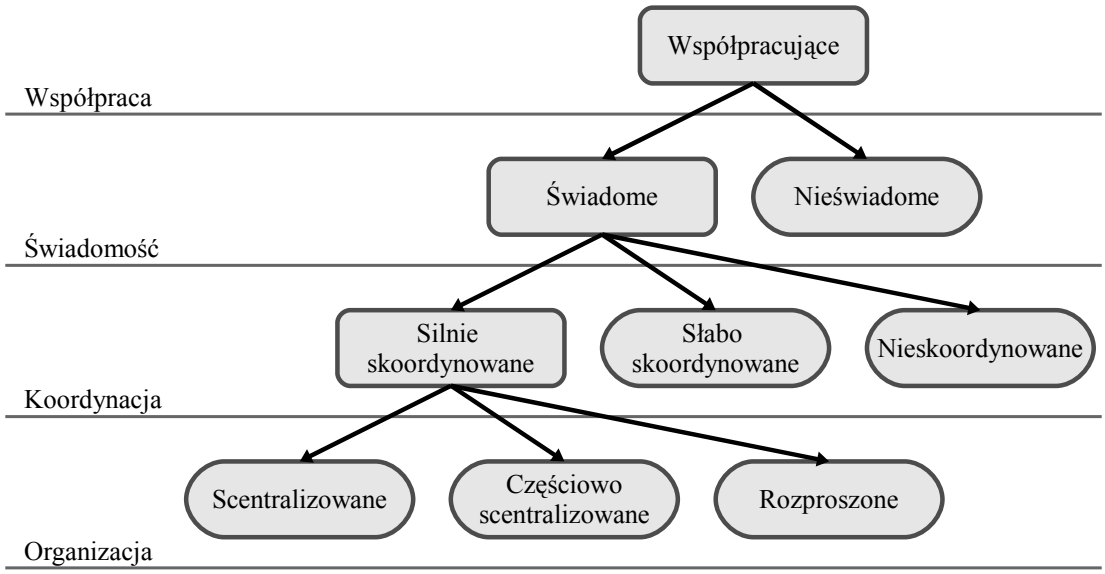
Rysunek 4.3: Cykl działania programu sterującego robotem, stosujący podejście behawioralne.

Sterowniki behawioralne charakteryzują się one bardzo krótkim czasem reakcji na zmiany w środowisku. Nie jest jednak możliwe projektowanie w ten sposób złożonych algorytmów sterujących, które pozwalałyby na osiąganie skomplikowanych celów. Ich działanie na ogół nie jest w pełni przewidywalne.

W pracach prezentujących praktyczne zastosowania robotów, rzadko stosowne są sterowniki czysto behawioralne, a prawie nigdy czysto celowe. Połączenie zalet obu podejść uzyskuje się poprzez stosowanie sterowników hybrydowych, w których moduł planowania i sterowania reaktywnego działają równolegle. Algorytm planowania wyznacza sposób osiągnięcia zadanego celu, a algorytmy reaktywne pozwalają na szybkie reagowanie na zmiany w środowisku, które nie zostały wzięte pod uwagę w realizowanym planie.

Podział algorytmów na celowe i behawioralne dotyczy sposobu realizacji sterownika pojedynczego robota. W pracy [45] zaproponowano analogiczny podział systemów zarządzających działaniem grup robotów. Podstawą podziału jest sposób podejmowania decyzji i sposób realizacji globalnego celu grupy. System wielorobotowy można nazwać celowym, jeżeli sposób osiągnięcia globalnego celu jest wyznaczany centralnie i optymalizuje działania wszystkich robotów pod kątem wspólnego celu. System można nazwać behawioralnym jeżeli globalny cel systemu jest osiągany poprzez autonomiczne osiąganie celów przez poszczególne roboty.

Cechy systemów sterujących grupami robotów, które zostały wyróżnione w [45] są podstawą taksonomii zaproponowanej w [33]. Taksonomia ta skupia się na podziale systemów ze względu na sposób wykonywania zadań wymagających współpracy wielu robotów. Autorzy wyróżniają cztery grupy cech, które mogą różnić poszczególne systemy. Schemat drzewiastej struktury reprezentującej taksonomię jest przedstawiony na rysunku 4.4.



Rysunek 4.4: Taksonomia systemów wielorobotowych oparta na sposobie koordynacji działań [33].

Na najwyższym poziomie hierarchii cech odrzucane są wszystkie systemy, które nie potrafią realizować zadań wymagających współpracy robotów. Rozróżnienie systemów na świadome i nieświadome ma za zadanie wyróżnienie systemów, w których współpraca nie jest celowa. Systemy świadome, to takie, w których każdy z robotów posiada jakąkolwiek wiedzę o pozostałych robotach działających w ramach grupy. Tylko w tego typu systemach może występować koordynacja prowadzonych działań. Systemy stosujące koordynację zo-

stały dodatkowo podzielone na „silnie skoordynowane", które wykorzystują do koordynacji protokół komunikacyjny, oraz „słabo skoordynowane", w których operacje są koordynowane w inny sposób.

Systemy, w których występuje koordynacja wykonywanych działań, zostały podzielone ze względu na organizację, czyli sposób podejmowania decyzji. W podejściu scentralizowanym występuje wyróżniony algorytm nadzorujący, który decyduje o sposobie realizacji zadania przez grupę. Wszystkie roboty podległe nadzorcy wykonują jedynie jego polecenia i nie mogą przejawiać autonomicznych aktywności. W systemach określanych jako rozproszone każdy robot podejmuje autonomicznie wszystkie decyzje. Systemy częściowo scentralizowane to klasa pośrednia, w której mogą występować elementy sterowania scentralizowanego i autonomicznego. Mogą tu występować autonomiczne podgrupy odpowiedzialne za wykonanie elementów zadania, wewnątrz których sterowanie jest scentralizowane.

W tej samej pracy autorzy wyróżnili również grupę własności systemowych, które rozróżniają poszczególne systemy:

- sposób komunikacji – przy pomocy dedykowanych urządzeń lub obserwacji zachowań,
- rodzaj robotów – homogeniczne lub heterogeniczne,
- architektura systemu – celowa lub behawioralna,
- rozmiary grupy – możliwość obsługi dużej liczby robotów lub brak takiej możliwości.

W literaturze przedmiotu można odnaleźć również inne taksonomie algorytmów sterujących pojedynczymi robotami i grupami robotów. Przykładem może być propozycja D. Spero, przedstawiona w [75]. Rozróżnia ona podejścia ze względu na sposób realizacji systemu nawigacji robotów. Kryteriami są:

- architektura sterownika – celowa lub behawioralna,
- rodzaj stosowanych napędów,
- rodzaj sensorów i sposób postrzegania otoczenia,
- sposób reprezentowania modelu środowiska,
- stosowany algorytm planowania trasy,
- sposób rozwiązania problemu lokalizacji.

Analizując cechy, jakie są rozważane w napotkanych w literaturze taksonomiach, można dojść do wniosku, że problem jakości systemów zarządzających działaniem robotów nie jest w ogóle brany pod uwagę. Systemy nie są analizowane pod kątem możliwości dodawania nowych typów zadań czy odporności na awarie poszczególnych elementów. Nie

bada się stosowanych architektur oprogramowania czy metod separowania funkcjonalności i integrowania komponentów. W niewielkim stopniu analizuje się skalowalność systemu, rozumianą najczęściej jako zdolność do wykorzystywania licznych grup robotów.

Przyczyn należy dopatrywać się w sposobie definiowania systemów wielorobotowych. Nie są one traktowane jako systemy informatyczne wykorzystujące roboty, ale raczej jako grupy robotów, które ze sobą współpracują. Powstające projektu i implementacje mają za zadanie zrealizowanie konkretnych zadań przy pomocy określonych robotów. Warunki ich realizacji, czyli właściwości pozafunkcjonalne powstającego oprogramowania, są najczęściej pomijane.

Warto jednak przyjrzeć się kilku ciekawszym architekturom programów sterującego grupami robotów. Prezentowany poniżej przegląd zostanie podzielony ze względu na sposób koordynacji:

- grupy robotów autonomicznych nie stosujące koordynacji,
- koordynacja oparta o obserwację działania pozostałych robotów w grupie,
- koordynacja wykorzystująca protokół komunikacyjny,
- scentralizowane planowanie i synchronizacja czynności.

4.1.2 Współpraca niezależnych robotów autonomicznych

W literaturze istnieją przykłady zadań, które mogą być realizowane przez grupy robotów, ale nie wymagają koordynacji ich działania. W takim przypadku każdy z robotów dąży autonomicznie do osiągnięcia swojego własnego celu. Roboty mogą być świadome swojej wzajemnej obecności, ale nie komunikują się ze sobą i nie synchronizują wykonania zadań. Spodziewać się można, że tego typu architektura powinna być dobrze skalowalna, ponieważ nie występują tu typowe wąskie gardła, jak przepustowość medium komunikacyjnego czy algorytmy centralnego planowania.

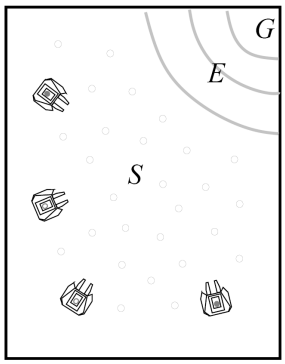
Przykładem problemu, który można skutecznie rozwiązać przy pomocy tego typu architektury, jest zadanie rozmieszczania sieci sensorycznej. Polega ono na rozlokowaniu pewnej liczby robotów w środowisku w taki sposób, by ich percepcja pokrywała możliwie największy jego fragment. Każdy z robotów jest wyposażony w układy sensoryczne, które mogą pobierać pożądaną informację z otoczenia robota, ale mają ograniczony zasięg działania. Jednym z najprostszych przykładów zastosowań sieci sensorycznej może być grupa robotów wyposażonych w mikrofony, które mają wykrywać obecność ludzi w chronionym budynku.

W pracy [12] zaprezentowano porównanie metod rozmieszczania sieci sensorycznej. Stosowane roboty były identyczne, każdy z nich był wyposażony w kamerę i skaner laserowy. Programy sterujące każdym z robotów były takie same i zrealizowane w podejściu behawioralnym. W najbardziej podstawowej wersji każdy z robotów implementował zachowanie

przemieszczania i unikania kolizji. W rozbudowanej wersji dodano zachowanie umożliwiające postrzeganie innych robotów w środowisku. Algorytm sterujący starał się przemieścić robota tak, by maksymalnie oddalić się od innych robotów. Powodowało to bardziej równomierne rozmieszczenie robotów w środowisku i w rezultacie znacznie lepsze pokrycie go przez zasięg sensorów sieci. Zrealizowane eksperymenty, w których wykorzystano grupy od 3 do 9 rzeczywistych robotów, pozwoliły na potwierdzenie poprawności podejścia.

Zaprezentowany system sprawnie radzi sobie z postawionym problemem, pomimo że jego architektura jest bardzo prosta. Brak w nim jakichkolwiek zależności pomiędzy poszczególnymi robotami, brak protokołu komunikacyjnego. Zarówno liczebność grupy jak i rozmiar środowiska mogą być więc swobodnie zwiększane; system jest więc bardzo dobrze skalowalny.

Innym zadaniem, które może zostać zrealizowane bez stosowania koordynacji działań, jest przeszukiwanie terenu. Roboty mają za zadanie odnaleźć w środowisku obiekty o określonych cechach. Przykładami praktycznych zastosowań rozwiązań tego zadania może być poszukiwanie min czy sprzątanie śmieci. W pracy [41] zaprezentowano kilka rozwiązań tego zadania w oparciu o różnego typu kontrolery behawioralne. Przeprowadzono testy z wykorzystaniem grupy czterech robotów typu IS Robotics R2e, z których każdy posiadał chwytak pozwalający na podnoszenie zlokalizowanych obiektów. Początkowa konfiguracja środowiska jest widoczna na rysunku 4.5.



Rysunek 4.5: Początkowy układ robotów i obiektów w testach opisanych w pracy [41].

Wszystkie obiekty miały zostać zgromadzone w miejscu oznaczonym literą „G”. Dwa spośród trzech testowanych podejść nie stosowały żadnej komunikacji. W pierwszym eksperymencie wszystkie roboty były sterowane takim samym algorytmem. Autorzy zdefiniowali 12 zachowań, które pozwalały robotom na unikanie kolizji, losowe przeszukiwanie terenu, wykrywanie obiektów, ich podnoszenie i odwożenie do celu. Tak skonfigurowany system był w stanie skutecznie czyścić teren z poszukiwanych obiektów.

Ciekawą miarą zastosowaną do oceny rozwiązania był czas spędzony przez roboty na realizacji zachowania odpowiedzialnego za unikanie kolizji. Ponieważ roboty nie komunikowały się ze sobą w żaden sposób, liczba potencjalnych kolizji była bardzo znaczna. Wykres zależności czasu spędzonego na unikaniu kolizji od położenia robotów pokazał, że najwięcej tego typu zdarzeń zachodziło w okolicy punktu „G".

Drugi eksperyment miał na celu ograniczenie tego efektu. Roboty zostały podzielone na dwie tak zwane „kasty", które okupowały różne fragmenty terenu. Cel robotów z kasty poszukującej został zmodyfikowany w taki sposób, że przenosiły one znalezione obiekty w okolicę punktu „E". Robot z drugiej kasty miał za zadanie przenoszenie obiektów z okolic punktu „E" do punktu „G". Taka modyfikacja pozwoliła znacznie ograniczyć liczbę kolizji zachodzących w okolicy punktu „G". Jednocześnie ponad dwukrotnie pogorszyła się skuteczność działania całego systemu, która mierzona była jako czas realizacji eksperymentu.

Autorzy podkreślają, że prezentowane rozwiązania są odporne na awarie poszczególnych robotów. Uszkodzenie któregokolwiek z robotów w systemie nie wpływa w żaden sposób na działanie pozostałych, więc system może nadal poprawnie funkcjonować. W rozwiązaniu definiującym kasty elementem mogącym spowodować awarię systemu był pojedynczy robot z kasty przenoszącej obiekty do celu.

Prezentowane rozwiązanie jest również dość dobrze skalowalne. Wielkość środowiska oraz liczba robotów mogą być swobodnie zwiększane. Trzeba jednak zaznaczyć, że w dużym środowisku, w którym obiekty byłyby rozłożone nierównomiernie, skuteczność rozwiązania byłaby bardzo niska, ponieważ zastosowane losowe przeszukiwanie jest bardzo zawodne.

W pracy [52] zaprezentowano algorytm behawioralny, który pozwalał na przepychanie pudełka przez grupę robotów. Wymiary pudła były na tyle duże, że nie było możliwe przemieszczenie go przez jednego robota. Każdy z robotów miał za zadanie znaleźć się z odpowiedniej strony pudła i naciskać na nie w odpowiednim kierunku. W tym systemie również nie występowały ograniczenia liczebności grupy ani rozmiarów środowiska, choć uzyskana w ten sposób skalowalność nie ma większego znaczenia w kontekście wykonywanego zadania. Zwiększenie liczby robotów wykonujących zadanie przepychania pudła nie wpływa na jego szybsze czy lepsze wykonanie.

Cechy pozafunkcjonalne architektur behawioralnych wynikają po części z przyjętych w nich założeń. Brak algorytmów wykonywanych centralnie oraz brak komunikacji pomiędzy robotami gwarantują dobrą skalowalność tego typu rozwiązań. Roboty są w zasadzie niezależne, więc dodanie kolejnych nie stanowi problemu. Od rodzaju zadania zależy, czy zwiększenie liczebności zespołu wpływa korzystnie na jakość rozwiązania.

Ograniczenie lub całkowite wyeliminowanie zależności pomiędzy członkami grupy pozwala na osiągnięcie wysokiego stopnia niezawodności systemu. Jedynym elementem, który może zawieść, jest jeden z robotów. Przy odpowiednim zaprojektowaniu zachowań i ogra-

niczeniu zależności, awaria jednego lub kilku członków grupy nie spowoduje awarii całego systemu, a co najwyżej, ograniczy jego wydajność.

Autonomia poszczególnych robotów w zakresie podejmowania decyzji, czyli wyznaczania sterowania przy wykorzystaniu lokalnej jednostki sterującej powoduje, że oprogramowanie jest na ogół bardzo mocno związane ze sprzętem. Algorytmy są tworzone od podstaw do sterowania określonym robotem i w określonym celu. Ogranicza to znacznie możliwość ponownego wykorzystania elementów oprogramowania.

Największą wadą podejścia behawioralnego do tworzenia systemów wielorobotowych jest ograniczenie rodzajów zadań, jakie mogą być przy ich pomocy rozwiązywane. W odniesieniu do stawianych systemom wymagań pozafunkcjonalnych oznacza to niską rozszerzalność, czyli bardzo znaczące ograniczenia możliwości dodawania nowych funkcjonalności.

4.1.3 Koordynacja w oparciu o obserwację

Zadania wymagające koordynacji działania wielu robotów mogą być realizowane bez wykorzystania urządzeń dedykowanych do komunikacji. Rezygnacja z przesyłania wiadomości ułatwia często realizację rozwiązania i nie wprowadza ograniczeń skali. Roboty nadal są w pełni autonomiczne, nie otrzymują poleceń od innych elementów systemu, ale są w stanie wykonać bardziej złożone zadania.

Przykładem problemu wymagającego koordynacji działań jest przemieszczanie robotów w formacjach. Polega ono na utrzymywaniu wzajemnego ułożenia robotów podczas przemieszczania całej grupy, bez względu na napotykane przeszkody. Może znaleźć zastosowanie w zadaniach takich jak sprzątanie czy przeszukiwanie terenu. W pracy [11], która została zrealizowana dla amerykańskiej agencji obrony DARPA, rozważano problem utrzymywania formacji automatycznie kierowanych wozów bojowych. Formacje, w których miały się przemieszczać pojazdy, widoczne są na rysunku 4.6.

Rysunek 4.6: Formacje robotów realizowane w systemie behawioralnym; od lewej: linia, kolumna, diament, klin [11].

Każdy z robotów miał przypisaną na stałe pozycję w formacji. W grupie występował jeden lider oraz trzy roboty utrzymujące pozycję względem lidera i pozostałych członków formacji. Lider decydował kierunku i prędkości przemieszczania grupy. Sterownik robota

utrzymującego się w formacji wykonywał cyklicznie dwa etapy algorytmu: wykrycie pozycji formacji i utrzymanie się w formacji. W pierwszym etapie ustalana była bieżąca lokalizacja i orientacja pozostałych robotów. Drugi etap polegał na wyznaczeniu pożądanej pozycji robota w formacji i obliczeniu odpowiedniego sterowania.

Analizując przedstawione wcześniej rodzaje sterowników (rysunki 4.2 oraz 4.3) można stwierdzić, że zaprezentowany algorytm sterowania pojedynczym robotem ma cechy podejścia celowego (wbrew założeniom autorów). Natomiast cały system można sklasyfikować jako behawioralny, ponieważ nie ma występuje tu centralne podejmowanie decyzji.

W pracy nie rozważano problemu skali środowiska, nie była budowana ani wykorzystywana mapa. Autonomia poszczególnych robotów oraz brak komunikacji pomiędzy mini zapewnia skalowalność rozwiązania. Kolejne roboty mogą być dołączane do grupy w celu powiększenia formacji. Problemem jest jednak konieczność definiowania miejsca każdego z robotów w formacji, która wymaga zmian oprogramowania robotów już w formacji będących. Sporą wadą proponowanego podejścia wydaje się być brak odporności na awarie, w szczególności w kontekście rozważanych zastosowań militarnych. Uszkodzenie lidera powoduje zatrzymanie całej formacji. Uszkodzenie poszczególnych członków zespołu może powodować kłopoty z działaniem algorytmu lokalizowania formacji.

Innym sposobem komunikacji, który nie wykorzystuje dedykowanych urządzeń ani protokołów, jest pozostawianie śladów, które mogą być postrzegane i interpretowane przez pozostałe roboty. Rozwiązanie tego typu, inspirowane sposobem porozumiewania wykorzystywanym przez mrówki, zostało zaprezentowane w [81]. Grupa robotów miała za zadanie sprzątanie podłogi budynku, którego plan nie jest znany. Problem ten jest bardzo podobny do prezentowanego wcześniej zadania rozstawiania sieci sensorycznej. Uogólnieniem obu problemów jest tak zwane zadanie pokrycia terenu (ang. multi-mobot coverage task).

Przemieszczająca się mrówka pozostawia za sobą niewielką ilość substancji, zwanej feromonem, która może być wykrywana przez inne mrówki. Tą metodą grupa mrówek potrafi odnaleźć drogę do pokarmu i do mrowiska. Nieużywane drogi zanikają, ponieważ feromony się ulatniają. Zastosowanie tego mechanizmu w systemie współpracujących robotów wymaga zdefiniowania modelu środowiska, w którym możliwe będzie oznaczanie fragmentów odwiedzonych przez roboty. Model zaproponowany w [81] jest grafem skonstruowanym z siatki zajętości pokrywającej środowisko. Z każdą krawędzią grafu jest skojarzona liczba oznaczająca ilość feromonu. Robot przejeżdżający przez fragment terenu modelowany krawędzią zwiększa liczbę o stałą wartość. Z upływem czasu wszystkie liczby są zmniejszane o stałą wartość.

Algorytm sterujący każdym z robotów sprzątających podłogę był bezstanowy. Każdy krok składał się z pokonania jednaj krawędzi grafu. W każdym węźle robot wybierał tą krawędź, na której ilość feromonu była najmniejsza. Takie podejście pozwala na pokrycie

całego terenu. Autorzy opisują dodatkowo kilka modyfikacji poprawiających właściwości algorytmu oraz dowodzą jego poprawności.

Prezentowane rozwiązanie jest dobrze skalowalne – dodawanie kolejnych poprawia jakość rozwiązania, czyli skraca czas pokrywania terenu. Znaczącą wadą z punktu widzenia skalowalności i niezawodności jest konieczność istnienia wspólnej mapy. Sposób jej realizacji oraz współdzielenia nie jest przedmiotem rozważań; wszystkie eksperymenty zostały zrealizowane w z wykorzystaniem metod symulacyjnych a nie prawdziwych robotów.

Rezygnacja z jawnej komunikacji na rzecz obserwacji zachowań pozostałych robotów pozwala uniknąć konieczności wyposażania robotów w urządzenia do bezprzewodowej komunikacji oraz definiowania protokołów. Dzięki temu powstające systemy są dobrze skalowalne. Tworzenie zależności pomiędzy robotami, zwłaszcza zależności jednostronnych, powoduje jednak najczęściej ograniczenie odporności na awarie. Jeżeli funkcjonowanie pewnej grupy robotów zależy od poprawnego działania jednego robota, staje się on wrażliwym elementem systemu.

Systemy, które koordynują działania poprzez wzajemną obserwację działań, zaliczają się do grupy systemów behawioralnych. Są na ogół konstruowane z robotów autonomicznych pod względem podejmowania decyzji. Oznacza to, że również w tej klasie systemów algorytmy sterujące są mocno powiązane ze sprzętem. Ich ponowne wykorzystanie jest przez to bardzo utrudnione. Rozbudowa funkcjonalności takiego systemu wymaga modyfikowania programów poszczególnych robotów, co wymaga zatrzymania ich działania.

4.1.4 Koordynacja wykorzystująca komunikację

Grupy robotów wykorzystujące komunikację powinny być w stanie realizować bardziej złożone zadania, działając przy tym bardziej przewidywalnie i wydajnie. Jednak urządzenia do komunikacji są elementami podatnymi na awarię i mają ograniczony zasięg. Dodatkowo mogą generować ograniczenia skali systemu, ponieważ każdy mechanizm komunikacji ma ograniczoną przepustowość. Tak więc, wraz z większymi możliwościami pojawiają się dodatkowe problemy, które muszą zostać rozwiązane.

Bardzo ciekawą architekturą programów sterujących autonomicznymi robotami jest platforma ALLIANCE prezentowana w pracach [64, 65, 66]. Jest ona oparta na podejściu behawioralnym i przyjmuje następujące założenia dotyczące działania robotów:

1. Grupa składa się z n robotów, które mogą postrzegać swoje akcje i akcje innych robotów.

2. Dostępne jest medium komunikacji pomiędzy robotami, ale nie jest ono niezawodne.

3. Działanie wszystkich układów sensorycznych jest niedokładne,

4. Każdy robot może przestać działać w dowolnej chwili.

5. Robot nie zawsze będzie w stanie poinformować o swojej awarii.

Zasady funkcjonowania robotów definiowane są tak zwanymi motywacjami. Motywacja jest metodą aktywacji poszczególnych zachowań i pozwala na elastyczne reagowanie na zmiany zachodzące w środowisku. Jeżeli robot wykonuje jakieś zadanie, a zmiana w środowisku spowoduje, że zadanie traci sens, motywacja przestanie aktywować bieżące zachowanie. Podczas działania robota, motywacje otrzymują dane z sensorów robota, z odbiornika komunikatów oraz z pozostałych aktywnych zachowań. Wykrycie zadania powoduje aktywację zachowania, które pozwala na realizację tego zadania.

Najważniejszą właściwością gwarantowaną przez platformę ma być odporność na awarie poszczególnych robotów. Każdy robot wykonujący zadanie rozgłasza wiadomości o swojej bieżącej aktywności oraz o zakończeniu wykonywanego zadania. Bezczynne roboty obserwują wykonanie zadań poprzez odbieranie tych komunikatów. Jeżeli robot przestanie rozgłaszać komunikaty, lub realizuje zadanie zbyt długo, pozostałe roboty mogą założyć, że uległ on awarii i przejąć wykonanie tego zadania.

Pierwszym z prezentowanych zastosowań platformy ALLIANCE było wspólne przepychanie pudła [64]. Celem było przesunięcie długiego pudła przez pokój, co wymagało współpracy dwóch robotów. Zdefiniowane zostały dwa rodzaje zadań: „popchnij trochę po lewej" oraz „popchnij trochę po prawej". Motywacja każdego z zadań była aktywowana zakończeniem poprzedniego (z wyjątkiem pierwszego zadania). Tak skonfigurowany sterownik został zainstalowany na dwóch robotach, które skutecznie przepychały pudło. Podczas gdy pierwszy robot pchał jeden z końców pudła, drugi obserwował poprawność wykonania tego zadania. Zakończenie zadania aktywowało zachowanie drugiego robota i pozwalało na przepchnięcie drugiego końca pudła.

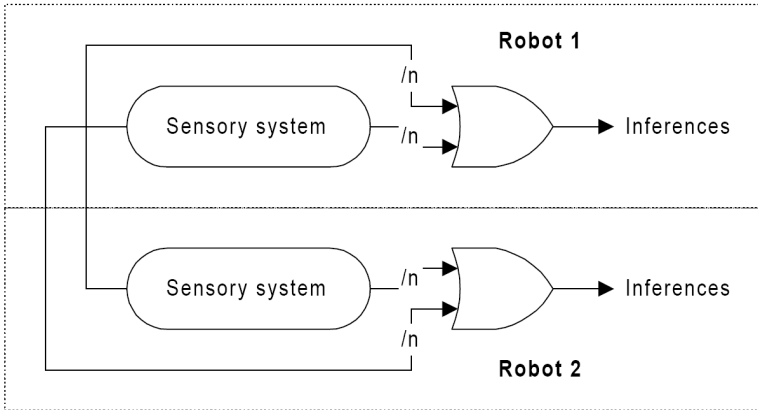
Demonstracja odporności na awarie polegała na wyłączeniu jednego z robotów w trakcie eksperymentu. Drugi robot rozpoczął automatycznie wykonywanie obu typów zadań na zmianę.

Innymi zastosowaniami platformy było przeszukiwanie terenu i zbieranie obiektów [65]. Zdefiniowany zestaw zachowań był bardzo podobny do prezentowanego wcześniej rozwiązania, które nie stosowało komunikacji ([41]). Roboty autonomicznie wykonywały poszczególne zadania, a komunikacja była wykorzystywana głównie do wykrywania awarii.

Platforma ALLIANCE stosuje komunikację do zapewnienia odporności na awarię w systemie behawioralnym złożonym z robotów autonomicznie podejmujących decyzje. Takie podejście nie rozwiązuje jednak problemu rozszerzalności. Dodanie nowych funkcji ciągle wymaga zatrzymania działania systemu i modyfikacji algorytmów w wszystkich robotów. Problem braku możliwości ponownego wykorzystania fragmentów oprogramowania również nie został rozwiązany. Dodatkowo, wprowadzenie mechanizmu komunikacji przez rozgła-

szanie może ograniczyć skalowalność systemu, ponieważ każde medium komunikacyjne ma ograniczoną przepustowość.

Innym przykładem architektury wykorzystującej komunikację autonomicznych robotów jest podejście o nazwie HIVEMind prezentowane w [49]. Komunikacja jest tu wykorzystywana do rozgłaszania danych z sensorów poszczególnych robotów. Dzięki temu każdy z algorytmów sterujących autonomicznymi robotami ma dostęp do kompletu odczytów pobieranych przez wszystkie sensory. Schemat współpracy sterowników dwóch robotów jest widoczny na rysunku 4.7.



Rysunek 4.7: Schemat ideowy działania sterowników behawioralnych w architekturze HIVEMind [49].

Architektura została przetestowana na przykładowym zadaniu wyszukiwania obiektu o zadanych właściwościach, wokół któego miały zostać zgromadzone wszystkie roboty. Każdy z robotów autonomicznie przemieszczał się w nieznanym środowisku, aż do chwili gdy obiekt został znaleziony. Wszystkie roboty były w stanie stwierdzić fakt odnalezienia obiektu i zmienić cel na przemieszczanie w jego kierunku.

Realizacja mechanizmu wymiany danych z sensorów wymaga bardzo wydajnego medium komunikacyjnego nawet dla niezbyt licznych grup robotów. Rozwiązanie to charakteryzuje się więc bardzo znaczącym ograniczeniem skali.

Bardziej bezpośrednie wykorzystanie protokołu komunikacyjnego do koordynacji działań zostało zaprezentowane w pracy [46]. Zaprezentowana tu architektura wykorzystuje mechanizm dynamicznego przydziału ról dla poszczególnych robotów, które mają pozwolić na osiągnięcie celu grupy. Została ona zastosowana do zrealizowania systemu robotów grających w piłkę w lidze RoboCup [68].

Do rozwiązania zadania konieczne jest przypisanie grupie n robotów ról, które mogą być utożsamiane z zachowaniami. Każdy z robotów oblicza dla każdej z ról współczynnik

użyteczności. Obliczone współczynniki są rozgłaszane przy pomocy medium komunikacyjnego. Następnie każdy z robotów stosuje ten sam algorytm, który pozwala na obliczenie najbardziej odpowiedniego przypisania ról do robotów.

Do grupy systemów robotów autonomicznych wykorzystujących komunikację należy też zaliczyć wszystkie rozwiązania wzorowane na metodach rynkowych. Szczegółowy przegląd zastosowań tego podejścia zaprezentowany został w opracowaniu [26]. Typowym zastosowaniem jest problem przydziału zadań. Każdy z robotów jest uznawany za uczestnika pewnej gry rynkowej, każdy ma możliwość składania ofert realizacji poszczególnych zadań. Zadania są przyznawane robotom oferującym najlepsze warunki. Mechanizm przyznawania zadań jest najczęściej realizowany na wzór aukcji.

Tego typu rozwiązania są jednak najbardziej adekwatne w sytuacjach, w których nie występuje konieczność współpracy robotów przy wspólnym zadanie, a raczej możliwe jest zdefiniowanie niezależnych zadań, z których każde może zostać wykonane przez jednego robota.

Zastosowanie protokołu komunikacyjnego przez grupę współpracujących robotów autonomicznych pozwala na zwiększenie skuteczności działania oraz na realizowane szerszego spektrum zadań. Wprowadza jednak ryzyko ograniczenia skalowalności systemu związane z ograniczeniami przepustowości medium komunikacyjnego. Nie rozwiązuje także problemów z dodawaniem nowych funkcjonalności systemu oraz możliwością wielokrotnego wykorzystywania oprogramowania.

4.1.5 Scentralizowane podejmowanie decyzji

Centralizacja podejmowania decyzji pozwala teoretycznie na lepsze zoptymalizowanie wykorzystania robotów przy realizacji działań. Moduł podejmujący decyzję centralnie powinien mieć dostęp do szerszej wiedzy, niż poszczególne, autonomicznie decydujące roboty. Umieszczenie logiki proaktywnej systemu w jednym miejscu powinno także ułatwiać modyfikowanie funkcjonalności całego systemu.

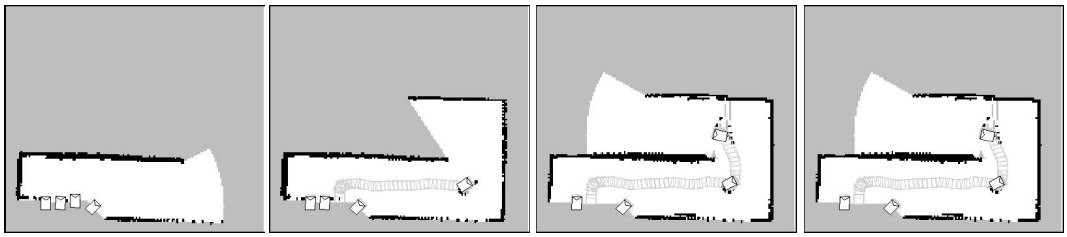
Oczywistą wadą centralizacji jest tworzenie elementu podatnego na uszkodzenia. Jeżeli moduł podejmujący decyzje przestanie poprawnie działać, system nie może funkcjonować. Dodatkowo konieczne jest przekazywanie znacznych ilości informacji do i z modułu centralnego oraz prowadzenie w nim złożonych obliczeń. Może to powodować ograniczenia skalowalności systemu. Dlatego też stosowane są różnego typu podejścia hybrydowe i hierarchiczne, by wykorzystać zalety centralizacji oraz autonomii w podejmowaniu decyzji.

Przykładem rozwiązań w pełni scentralizowanych jest większość systemów koordynacji ruchu. Algorytm planujący trajektorie oraz sterujący efektorami poszczególnych robotów

jest uruchamiany na zewnętrznym komputerze, który ma dostęp do wszystkich niezbędnych informacji i interfejsów. Kwestia lokalizacji robotów nie jest najczęściej rozważana. Przykładami takich rozwiązań są algorytmy opisywane w [15] czy [16].

Scentralizowane podejmowanie decyzji zostało zastosowane w pracy Profesora C. Zielińskiego [85], w której opisano metodę realizacji zadania przepychania pudła przez trzy roboty. Tylko jeden z robotów był w stanie postrzegać orientację pudła w przestrzeni; pozostałe były wyposażone w sensory umożliwiające lokalizowanie się względem pudła. Wyróżniony robot cyklicznie obliczał odpowiednie sterowanie dla pozostałych robotów i przekazywał im decyzję. W zrealizowanym systemie sterującym wykorzystano formalny opis algorytmów sterujących poszczególnymi robotami, skupiając się na szczegółowym przedstawieniu sposobu realizacji sterowania. Właściwości pozafunkcjonalne zrealizowanego systemu nie są rozważane.

Metoda wykorzystująca centralne podejmowanie decyzji w zadaniu rozbudowywania sieci sensorycznej została zaproponowana w [44]. Zaproponowana metoda w sposób przyrostowy rozbudowuje sieć sensoryczną umieszczając kolejnego robota na granicy zasięgu pokrycia, jaki został uzyskany przez poprzednie roboty. Etapy eksperymentu z wykorzystaniem czterech rzeczywistych robotów prezentuje rysunek 4.8



Rysunek 4.8: Kolejne etapy rozszerzania pokrycia z eksperymentu opisane w pracy [44].

Opisany system może pracować w środowisku nieznanym, ale przyjmuje, że pozycje wszystkich robotów względem dowolnego układu współrzędnych są dane. Nie są tu rozważane kwestie związane z ograniczeniami komunikacji czy możliwymi awariami systemu zarządzania lub poszczególnych robotów.

Przykładem systemu częściowo scentralizowanego może być rozwiązanie opisane w [39]. Celem grupy robotów jest przepychanie pudła, które wymaga współpracy przynajmniej dwóch robotów. Przyjęto założenie, że grupa składa się z robotów heterogenicznych, wśród których tylko jeden robot (obserwator) posiada zdolność postrzegania celu, do którego pudło ma zostać przetransportowane. Pozostałe dwa roboty (pchacze) mogą zlokalizować pudło i odbierać komunikaty od robota obserwującego cel. Obserwator przesyła do pchaczy

informacje o położeniu pudła względem celu, a pchacze przepychają pudło w odpowiednim kierunku.

Taki podział ról i zadań w systemie wydaje się nie być najlepszym rozwiązaniem z punktu widzenia wymagań pozafunkcjonalnych. Algorytmy realizujące konkretną funkcjonalność systemu są rozdzielone pomiędzy roboty i nie mogą być łatwo modyfikowane. Robot realizujący funkcję obserwatora może być tylko jeden, a jego awaria powoduje unieruchomienie całego systemu.

Pożądane właściwości systemów zarządzania robotami, które zostały zdefiniowane w rozdziale 3, są bardzo rzadko wspominane w pracach związanych ze sterowaniem grupami robotów. Prawie nigdy nie są wskazywane jako cele tych prac. Pomimo intensywnych poszukiwań autorowi nie udało się znaleźć opracowań, które jawnie rozważałyby problem jakości budowanych systemów, czy też spełnienia poszczególnych wymagań pozafunkcjonalnych.

4.2 Podejście agentowe do sterowania i zarządzania grupami robotów

Agenty i systemy agentowe są wielokrotnie wspominane w opisanych wcześniej publikacjach. Zostaną one przeanalizowane oddzielnie. Zastosowanie paradygmatu agentowego jest celem niniejszej pracy, więc przypadkom i sposobom zastosowania agentów w istniejących systemach wielorobotowych warto poświęcić więcej miejsca.

4.2.1 Agent i system agentowy

Podejście agentowe do projektowania i implementowania systemów jest uznawane za ugruntowane i dojrzałe. Nie istnieje jednak jedna, spójna definicja agenta, która byłaby uznawana przez wszystkich badaczy i projektantów. Encyklopedyczne definicje twierdzą, że agent jest komponentem programowym wykonującym autonomicznie długotrwałe zadania w imieniu osoby lub organizacji [32]. Z punktu widzenia twórców oprogramowania, taka definicja jest jednak dalece niewystarczająca.

Jedną z bardziej uznanych jest definicja Profesora N. Jenningsa, prezentowana między innymi w pracy [47]. Jest to definicja bardzo ogólna, zakreślająca pewne ramy, w których mieścić się powinny wszystkie pozostałe, bardziej szczegółowe lub formalne definicje agenta. Brzmi ona następująco:

Agent jest to system komputerowy, osadzony w pewnym środowisku, który jest w stanie działać elastycznie i autonomicznie w celu zrealizowania zadań, do których został stworzony.

Autor zwraca szczególną uwagę na trzy kluczowe elementy:

1. Agent jest osadzony w środowisku, co oznacza, że jest w stanie postrzegać stan środowiska poprzez sensory oraz wywierać wpływ na ten stan.
2. Agent jest autonomiczny, czyli może podejmować działania bez bezpośredniego polecenia od innego agenta i posiada kontrolę nad własnymi akcjami i stanem wewnętrznym.
3. Agent jest elastyczny, czyli:
    - reaguje na na zmiany zachodzące w środowisku,
    - jest proaktywny, czyli przejawia inicjatywę, inicjuje wykonanie czynności przybliżających go do celu,
    - przejawia zachowania społeczne, czyli potrafi postrzegać inne agenty, wchodzić z nimi w interakcje, a w szczególności komunikować się z nimi.

Dodatkową cechą, która często jest wymieniana jako podstawowa właściwość agentów, jest mobilność [82]. Jest ona rozumiana jako możliwość zmiany lokalizacji agenta w środowisku, która powoduje zmianę stanu środowiska oraz zmianę możliwości wpływania na środowisko przez agenta.

Jeżeli agent ma przejawiać zachowania społeczne, to w środowisku jego działania musi znajdować się przynajmniej jeszcze jeden agent. Najczęściej rozważane są grupy agentów współistniejących we wspólnym środowisku. Agenty oraz środowisko ich działania tworzą **system agentowy**.

Próbą bardziej formalnego zdefiniowania zasad działania agenta jest architektura BDI prezentowana między innymi w pracach [67, 50]. Jest on nawiązaniem do sposobu wnioskowania i działania wykonywanego przez człowieka. Opiera się na trzech pojęciach:

- Przekonania (ang. belief), które stanowią wiedzę agenta na temat otaczającego go środowiska.
- Pragnienia (ang. desire), będące celami działania agenta.
- Intencje (ang. intention), czyli ten podzbiór celów, które agent zdecydował się osiągnąć poprzez wykonanie określonego planu.

Agent obserwuje środowisko za pomocą posiadanych sensorów i konstruuje w swojej świadomości jego model. Przekonania agenta na temat stanu środowiska są podstawą do wybrania sekwencji akcji, które mają pozwolić na osiągnięcie jednego z celów (pragnień) agenta. Plany są więc elementem algorytmu agenta i są zależne od jego implementacji.

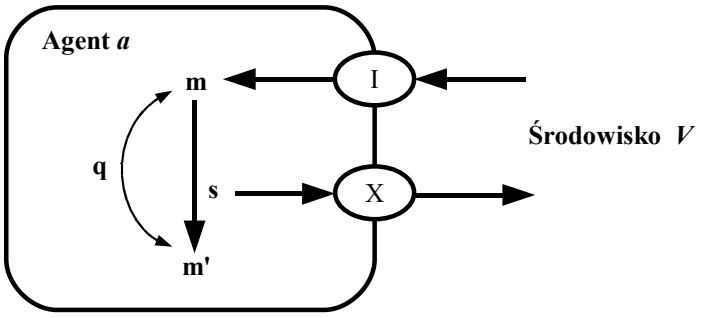
Przeprowadzana w dalszej części pracy analiza rodzajów agentów jest oparta na bardziej formalnym modelu agenta, który jest nazywany M-Agentem. Został on zaproponowany przez Profesora K. Cetnarowicza w pracy [23]. Główny nacisk został położony na sformalizowanie sposobu działania agenta w środowisku. Ogólny cykl działania M-Agenta to:

1. obserwacja stanu środowiska,
2. podjęcie decyzji o najbardziej adekwatnym sposobie działania,
3. wykonanie operacji zgodnie z podjętą decyzją.

Agent ma do dyspozycji wiele możliwych sposobów działania. Proces podejmowania decyzji ma na celu wybranie najwłaściwszej, a jest realizowany w następujący sposób:

1. agent buduje model postrzeganego środowiska (m),
2. wyznacza przewidywane zmiany modelu (m'), jakie mogą być skutkiem poszczególnych sposobów działania (s),
3. ocenia przewidywane zmiany, by wybrać najbardziej odpowiedni sposób działania z punktu widzenia bieżącego celu (q),
4. wybiera i wykonuje najlepszą strategię działania.

Schemat ideowy działania M-Agenta jest widoczny na rysunku 4.9.



Rysunek 4.9: Schemat działania M-Agenta [23].

Agenta a można więc formalnie zdefiniować jako:

$$a = (M, Q, S, I, X, L, m, q, s) \tag{4.1}$$

gdzie:

M – zbiór możliwych modeli środowiska, wiedza agenta a o środowisku, $m \in M$;
Q – zbiór uporządkowany celów agenta agenta, $q \in Q$;
S – zbiór możliwych strategii agenta, $s \in S$;
I – operator obserwacji, $m = I(M, V)$, V – stan środowiska,
X – operator wykonania strategii, $V' = X(s, V)$, V' – stan środowiska przewidywany przez agenta po zastosowaniu strategii s.,
L – operator adaptacji (uczenia).

Formalizm M-Agenta pozwala na opisanie i analizowanie właściwości i współdziałania różnego rodzaju agentów programowych. Skupia się on głównie na modelowaniu sposobu działania agenta, w szczególności na analizie algorytmu podejmowania decyzji. Wyróżnienie poszczególnych typów agentów współistniejących w prezentowanym w niniejszej pracy systemie zarządzającym działaniem grup robotów wymaga, by model został nieco rozwinięty. Modyfikacja będzie szła w kierunku doprecyzowania modelu środowiska agenta i jego interakcji ze środowiskiem. Zostanie ona szczegółowo zaprezentowana w rozdziale 6.4.

4.2.2 Agent a robot

Zajmując się badaniami nad sterowaniem robotami mobilnymi łatwo dopatrzyć się analogii pomiędzy cechami urządzenia mechanicznego, jakim jest robot, a cechami charakteryzującymi agenta. Warto w tym kontekście przeanalizować poszczególne cechy wynikające z ogólnej definicji Jenningsa:

- Robot mobilny jest osadzony w środowisku – jest to fizyczne środowisko jego działania.
- Jeżeli jest wyposażony w sensory, to może być w stanie postrzegać stan tego środowiska.
- Może mieć możliwość wpływania na stan środowiska, np. poprzez zmianę swojej pozycji.
- Może być mobilny, jeżeli może zmienić lokalizację w środowisku.
- Autonomię robota rozumie się najczęściej jako niezależność urządzenia mechanicznego od infrastruktury. Można rozważać autonomię w zakresie zasilania, zdolności percepcji oraz autonomicznego sterowania efektorami. W definicji agenta, autonomię rozumie się jako kontrolę nad własnym stanem i zdolność działania bez bezpośredniego polecenia.
- Proaktywność agenta jest rozumiana jako zdolność samodzielnego inicjowania działania. W przypadku robota jest to możliwe dopiero w połączeniu go z odpowiednim programem sterującym.

- Robot może być wyposażony w urządzenia umożliwiające komunikację. Może postrzegać inne roboty w swoim otoczeniu, i może przejawiać zachowania socjalne, jednak jest to efektem działania oprogramowania sterującego.

Można stwierdzić, że autonomiczny robot mobilny, rozumiany jako urządzenie mechaniczne umieszczone w świecie rzeczywistym, nie jest agentem. Można też stwierdzić, że jeżeli taki robot zostanie wyposażony w odpowiedni program sterujący, to może być agentem w rozumieniu definicji Jenningsa. Program sterujący jest niezbędny do zapewnienia wszystkich powyższych cech, poza osadzeniem w środowisku.

Warto zauważyć, że jeżeli robot w połączeniu z programem sterującym jest agentem, to sam program sterujący również jest agentem. Jego środowiskiem jest robot wraz z otoczeniem, w którym działa. Przy takim założeniu posiada wszystkie cechy ogólnej definicji.

W literaturze przedmiotu pojecie agenta i systemu agentowego jest stosowane bardzo często i bardzo chętnie. Bardzo rzadko natomiast analizuje się kwestie związane z definiowaniem agenta, rozróżnianiem typów agentów czy rodzajów interakcji między niemi. Stosuje się często pojęcie agenta upostaciowionego czy też ucieleśnionego (ang. embodied agent) [84]. Bardzo często też pojęcie agenta i robota stosuje się wymiennie [37, 61, 10].

Można zauważyć ogólne przekonanie o tym, że każdy robot mobilny jest agentem, natomiast system agentowy to grupa robotów mobilnych działających we wspólnym środowisku. Takie stwierdzenia są najczęściej prawdziwe, ale z całą pewnością nie wyczerpują możliwości i korzyści płynących z zastosowania paradygmatu agentowego.

Analizę sposobów wykorzystania agentów programowych do tworzenia programów sterujących robotami można znaleźć w pracy Profesora S. Ambroszkiewicza [4]. Wskazuje ona na problemy wynikające z bezpośredniego utożsamiania agenta z robotem i proponuje odmienne podejście, które jest oparte na paradygmacie SOA (ang. Service Oriented Architecture), czyli na architekturze orientowanej na usługi. Sterownik pojedynczego robota ma się składać z grupy agentów, z których każdy jest odpowiedzialny za realizację jednej funkcji robota (usługi) W ramach robota funkcjonują jeszcze inne agenty zajmujące się przekazywaniem informacji o usługach czy realizacją zadań. Takie podejście pozwala na elastyczną rozbudowę funkcjonalności robota poprzez dodawanie nowych agentów. Proponowana architektura przewiduje możliwość konstruowania agentów złożonych z innych agentów, co jest całkowitym odejściem od powszechnie stosowanego powiązania agenta z pojedynczym robotem. Podejście wydaje się bardzo obiecujące, jednak ocena jego własności pozafunkcjonalnych wymaga przeprowadzenia eksperymentów z wykorzystaniem wielu robotów.

Głównym celem stosowania paradygmatu agentowego jest rozwiązanie problemów pojawiających się w złożonych systemach informatycznych, w których wiele niezależnych zadań ma być wykonywane współbieżnie. Pozwala on na rozwiązanie problemów takich jak:

- Separacja wykonania zadań, osiągana poprzez współbieżne wykonanie kodu poszczególnych agentów.
- Bezpieczeństwo w dostępie do wspólnych zasobów (danych), które jest zapewnione dzięki asynchronicznej komunikacji.
- Skalowalność istniejących rozwiązań dzięki stosowaniu wirtualnej maszyny, w której uruchamiane są agenty. Moc obliczeniową można zwiększać poprzez przyłączenie do maszyny kolejnych komputerów. Wykonanie algorytmów można optymalizować poprzez przemieszczanie agentów pomiędzy komputerami.
- Możliwość modyfikacji poszczególnych modułów systemu, bez ryzyka zakłócania działania całego systemu. Poszczególne agenty nie powinny zakładać ciągłego poprawnego funkcjonowania, a nawet istnienia innych agentów. Dlatego możliwe jest bezpieczne usunięcie pojedynczego agenta i zastąpienie go nową implementacją.

Paradygmat agentowy umożliwia osiągnięcie powyższych właściwości systemu, ale nie gwarantuje ich. Aktualnie stosowane podejście do definiowania systemów agentowych zarządzających robotami mobilnymi, z całą pewnością nie jest optymalnym sposobem wykorzystania właściwości podejścia agentowego. Utożsamienie agenta z robotem i definiowanie systemu agentowego jako grupy współistniejących robotów, tworzy często dodatkowe, poważne problemy:

- Skomplikowana modyfikacja funkcjonalności systemu, spowodowana rozproszeniem implementacji poszczególnych zadań systemu pomiędzy programy uruchamiane na różnych komputerach. Zmiana sposobu działania wymaga zatrzymania wszystkich robotów i zmodyfikowania działania wszystkich programów sterujących. Brak maszyny wirtualnej, tworzącej wspólne środowisko agentów programowych, uniemożliwia wykonywanie zmian „w biegu".
- Złożone protokoły synchronizacji działań, które są skutkiem w pełni rozproszonego podejmowania decyzji. Protokoły muszą przewidywać wszystkie sytuacje, jakie mogą zajść podczas wykonania zadania i muszą być modyfikowane gdy przestrzeń sytuacji ulegnie zmianie.
- Kłopotliwa detekcja uszkodzeń oprogramowania i sprzętu, spowodowana bezpośrednim powiązaniem i uzależnieniem tych dwóch elementów w każdym robocie. Uszkodzenie mechaniczne robota często powoduje przerwanie wykonania programu sterującego. Analogicznie, błąd programu unieruchamia robota. Przyczyna uszkodzenia jest trudna do zdiagnozowania przez pozostałe elementy systemu, ponieważ objawy są takie same

- Wymagania dużej mocy obliczeniowej na urządzeniach mobilnych, spowodowane brakiem możliwości przemieszczania wykonywanych programów pomiędzy różnymi komputerami.

Można więc stwierdzić, że zalety, jakie mogą płynąć z zastosowania agentów programowych i systemów agentowych nie zostały jeszcze w pełni wykorzystane w dziedzinie projektowania i implementowania systemów zarządzających działaniem robotów mobilnych.

4.3 Przemieszczanie robotów mobilnych

Jak wcześniej wspomniano, problemy związane z przemieszczaniem są jednymi z najważniejszych zagadnień, które muszą zostać rozwiązane w każdym systemie wykorzystującym roboty mobilne. Stosowane metody powinny gwarantować szczególnie wysoką jakość i niezawodność, ponieważ są podstawą realizacji wszystkich wymagań funkcjonalnych stawianych przed tego typu systemami.

Ze względu na liczbę rozważanych robotów, zagadnienia związane z przemieszczaniem można podzielić na dwie grupy:

- przemieszczanie pojedynczego robota, czyli **nawigacja** robota mobilnego,
- jednoczesne przemieszczanie wielu robotów we wspólnym środowisku, czyli **organizacja ruchu** robotów.

4.3.1 Nawigacja robota mobilnego

Nawigację definiuje się jako dział wiedzy zajmujący się kierowaniem przemieszczającego się obiektu do określonego celu poprzez ustalanie jego lokalizacji oraz planowanie optymalnej trasy [30]. Z pojęciem nawigacji pojedynczego robota mobilnego wiąże się szereg problemów, które w ostatnich dziesięcioleciach były rozważane w licznych pracach naukowych. Ponieważ każdy z nich jest sam w sobie bardzo złożony, rozważane były najczęściej oddzielnie.

Reaktywne metody sterowania ruchem robota.

Algorytm sterujący ruchem robota mobilnego nie musi wykorzystywać mapy środowiska ani planować trajektorii ruchu. Metodę sterowania robotem nazywa się reaktywną, jeżeli funkcja obliczająca sterowanie dla efektorów robota korzysta jedynie z bieżących odczytów z sensorów robota. Tego typu metody mogą być wykorzystywane do unikania kolizji oraz do sterowania robotem w nieskomplikowanym środowisku [60, 80]. Głównymi zaletami algorytmów reaktywnych jest prostota implementacji, przejrzystość i bardzo niska złożoność obliczeniowa.

Najbardziej popularnym zastosowaniem tego typu metod są algorytmy wykorzystujące koncepcję pola potencjałów. Funkcja sterująca oblicza sterowanie, które przybliży robota do pożądanej lokalizacji unikając jednocześnie kolizji z przeszkodami. Cel podróży jest interpretowany jako potencjał ujemny, natomiast robot i przeszkody jako potencjały dodatnie. Robot jest więc „odpychany" od przeszkód i „przyciągany" przez cel. Praktyczne realizacje algorytmu są na ogół oparte o wykorzystanie odczytów z sensorów odległości oraz zewnętrzny system lokalizacji (np. GPS). Jeżeli na drodze do celu znajduje się przeszkoda, robot skręca w tę stronę, z której przeszkody są bardziej odległe, lub nie ma ich wcale. Algorytm jest trywialny i działa niezawodnie, jeżeli w środowisku nie ma przeszkód, których kształt jest figurą wklęsłą. Dokładna analiza tego typu metody została przedstawiona w [51]. Autorzy wskazali cztery główne wady podejścia:

- Wpadanie robota w pułapki w okolicach przeszkód których kształt jest figurą wklęsłą – w środowisku mogą istnieć lokalizacje, z których robot nie wydostanie się; będzie w nieskończoność, cyklicznie powtarzał sekwencję ruchów.
- Brak możliwości przejechania pomiędzy przeszkodami umieszczonymi blisko siebie, nawet jeśli rozmiary robota nie wykluczają takiej ścieżki. Potencjały odpychające od przeszkód sąsiadujących łączą się w jedno pole i uniemożliwiają przejechanie robota pomiędzy przeszkodami.
- Oscylacje w bliskiej obecności przeszkód – niewielka zmiana orientacji robota względem przeszkody może powodować znaczną zmianę obliczonego sterowania. Jest to powodowane nagłą zmianą decyzji funkcji sterującej pomiędzy omijaniem przeszkody i skręcaniem w kierunku celu.
- Oscylacje w wąskich przejściach, czyli „odbijanie" się robota od potencjałów sąsiadujących przeszkód.

Trzeba jednak zaznaczyć, że pomimo oczywistych niedoskonałości, metody sterowania reaktywnego mają wiele zalet. Nie należy ich za wszelką cenę dostosowywać do rozwiązania wszystkich zadań nawigacji, ale wykorzystanie ich właściwości do wydajnego i eleganckiego rozwiązania niektórych problemów wydaje się uzasadnione.

Lokalizacja robota, czyli ustalanie bieżącej pozycji i orientacji na znanej mapie.

Najlepsze wyniki, przy stosunkowo nieskomplikowanej implementacji, dają metody oparte o wykorzystanie filtra cząstek. Cząstki reprezentują możliwe lokalizacje robota na mapie. Podczas inicjalizacji algorytmu są one równomiernie rozmieszczane na mapie. Następnie, po każdym odczycie z sensorów, usuwane są te lokalizacje, które ewidentnie nie są z odczytem zgodne. Po każdym przemieszczeniu robota wszystkie pozostałe lokalizacje stają się źródłami nowych cząstek, które są przesuwane zgodnie z przemieszczeniem robota.

Po kilku iteracjach wszystkie możliwe lokalizacje powinny zgromadzić się wokół rzeczywistej pozycji robota. Rozmieszczenie cząstek na mapie można interpretować jako rozkład prawdopodobieństwa możliwych lokalizacji robota. Przegląd metod lokalizacji opartych na filtrach cząstek można znaleźć w [55].

Ciekawą optymalizację algorytmu lokalizacji opartego o wykorzystanie filtra cząstek zaproponowano w [70]. Ma ona na celu zmniejszenie liczby cząstek wykorzystywanych przez algorytm przy zachowaniu odpowiedniej skuteczności i dokładności lokalizacji. Jest oparta o wykorzystanie metody badania przesunięcia histogramów kątowych, która pozwala na określenie orientacji cząstek reprezentujących położenie robota.

Budowanie mapy środowiska przy założeniu znajomości własnej pozycji w otaczającym środowisku.

Najbardziej typowe reprezentacje mapy, jakie są wykorzystywane w nawigacji, są oparte na siatkach zajętości (ang. occupancy grid) lub lokalizacjach obiektów charakterystycznych (ang. landmarks) – stosowana reprezentacja jest uzależniona od rodzaju sensorów wykorzystywanych przez roboty. Siatka zajętości jest dwuwymiarową macierzą, której elementy reprezentują fragmenty środowiska działania robotów. Każdy fragment środowiska ma takie same wymiary. Wartości zapisane w elementach macierzy oznaczają prawdopodobieństwo wystąpienia przeszkody w odpowiednim fragmencie środowiska. Algorytm budowania siatki wykorzystuje iteracyjny wzór Bayesa, który pozwala na wyznaczenie poszukiwanego prawdopodobieństwa wykorzystując pomiary z wielu sensorów pobrane w wielu różnych sytuacjach [29]. Siatki zajętości są wykorzystywane zazwyczaj do reprezentacji niewielkich środowisk, w których występuje duża ilość przeszkód, jak na przykład wnętrza budynków. Lokalizacja wykorzystująca taką reprezentacje jest oparta na sensorach odległości, takich jak dalmierze laserowe, radarowe czy ultradźwiękowe. Zasadniczym problemem związanym z wykorzystaniem siatek zajętości jest konieczność przechowywania i przetwarzania dużych struktur danych, których wielkość nie jest zależna od ilości przeszkód, a jedynie od wielkości środowiska.

Metody reprezentacji mapy oparte na wykorzystaniu obiektów charakterystycznych są stosowane do reprezentowania rozległych środowisk, w których liczba przeszkód jest stosunkowo niewielka. Mapa składa się z listy jednoznacznie identyfikowalnych obiektów, które mogą być zlokalizowane względem innych obiektów lub skojarzone z konkretną lokalizacją [63]. Największym problemem podczas implementacji algorytmu budowania takiej mapy, lub jej późniejszego wykorzystania do lokalizacji, jest detekcja i rozróżnianie obiektów. Po rozpoznaniu obiektów poprawienie mapy jest realizowane z wykorzystaniem iteracyjnego wzoru Bayesa. Zazwyczaj tego typu metody są oparte o sensory wizyjne i analizę obrazu.

Jednoczesna lokalizacja i budowanie mapy, czyli SLAM.

SLAM (ang. Simultaneous Localization And Mapping) jest jednym ze sztandarowych problemów nawigacji robotów. Zadanie polega na zbudowaniu mapy nieznanego środowiska jedynie na podstawie odczytów z sensorów robota, przy jednoczesnym lokalizowaniu robota w budowanej mapie. W ciągu ostatnich dziesięcioleci naukowcy poświęcili problemowi SLAM wiele uwagi [73, 54], dzięki czemu udało się opracować teoretyczne podstawy jego rozwiązani oraz zrealizować praktyczne implementacje i przeprowadzić udane eksperymenty.

Rozwiązania problemu SLAM są oparte o wykorzystanie odometrii oraz pomiarów z różnego typu sensorów, które pozwalają na zidentyfikowanie obiektów charakterystycznych. W każdym kolejnym kroku algorytm SLAM wykonuje zestaw następujących operacji:

- pomiar przemieszczenia robota i wyznaczenie jego nowego stanu na podstawie odometrii,
- lokalizacja obiektów charakterystycznych oparta na odczytach z sensorów,
- identyfikacja obserwowanych obiektów z obiektami zaobserwowanymi w poprzednich krokach,
- oszacowanie pozycji robota na podstawie wyznaczonej pozycji obiektów,
- poprawienie mapy obiektów charakterystycznych, dodanie nowych obiektów.

Należy pamiętać, że wszystkie pomiary realizowane przez roboty (odometria, lokalizacja znaczników) są obarczone błędami. Dlatego też do wyznaczania stanu robota i obiektów charakterystycznych wykorzystywany jest filtr Kalmana [48], czyli algorytm estymacji stanu liniowych systemów dynamicznych na podstawie sterowania i pomiaru przy znanym modelu niedokładności urządzeń. Ponieważ system, jakim jest rzeczywisty robot mobilny, nie może zostać opisany równaniami liniowymi, stosowana jest nieliniowa modyfikacja, zwana rozszerzonym filtrem Kalmana (ang. Extended Kalman Filter, EKF). Z uwagi na kwadratową złożoność algorytmów aktualizacji macierzy kowariancji wykorzystywanej w EKF, algorytm SLAM może być wykorzystywany w czasie rzeczywistym dla najwyżej kilkuset obiektów charakterystycznych.

Planowanie i wykonanie trasy lub trajektorii.

W bardziej skomplikowanych środowiskach, w których układ przeszkód może być dowolny, zadanie przemieszczania robota mobilnego jest podzielone na dwa etapy. Pierwszy etap to zaplanowanie trasy przemieszczania, które wykorzystuje model środowiska, czyli mapę. W zależności od sposobu reprezentacji mapy, planowanie może wykorzystywać różne algorytmy znajdowania optymalnej ścieżki [24], jednak najczęściej będzie ono oparte o metody

grafowe. Kryterium optymalności ścieżki najczęściej będzie jej długość, jednak możliwe są też inne parametry, jak na przykład wydatek energetyczny czy średni ruch innych robotów na trasie. Jeżeli zaplanowana trasa jest skojarzona z funkcją określającą czas osiągnięcia poszczególnych jej punktów, wówczas taką trasę nazywa się trajektorią. Planowanie trajektorii jest zagadnieniem znacznie bardziej złożonym od planowania trasy, ponieważ wymaga wzięcia pod uwagę właściwości dynamicznych robota: maksymalnych przyspieszeń, prędkości czy sił odśrodkowych. Trajektoria pozwala nie tylko na wyznaczenie lokalizacji robota w określonej chwili czasu, ale także chwilowej prędkości oraz przyspieszenia.

Drugim etapem jest wykonanie trasy lub trajektorii, czyli uruchomienie algorytmu cyklicznie obliczającego sterowanie dla efektorów robota, które pozwoli na dostatecznie dokładne wykonanie przygotowanego planu. Nie jest oczywiście możliwe zaplanowanie całego ciągu sterowań, ponieważ wykonanie sterowania przez rzeczywiste urządzenie zawsze obarczone jest błędem. W każdym cyklu pracy algorytmu wyznaczany jest bieżący stan robota oraz jego odchylenie względem planu. Na tej podstawie obliczane jest sterowanie, które pozwoli na zminimalizowanie odchylenia w następnych krokach.

W literaturze można znaleźć wiele prac poświęconych wyznaczaniu i wykonaniu tras i trajektorii dla różnego typu robotów [43] [38].

4.3.2 Zarządzanie ruchem robotów

Zagadnienie jednoczesnego przemieszczania wielu robotów jest dalece bardziej skomplikowane. Zawiera ono w sobie wszystkie problemy związane ze sterowaniem ruchem pojedynczego robota, jednocześnie zwiększając złożoność obliczeń i wzbogacając je o kwestie koordynacji ruchu. Można wyróżnić dwa ogólne podejścia do problemu koordynacji ruchu:

- centralne planowanie tras i wykorzystanie algorytmu nawigacji autonomicznej robotów.
- centralne planowanie i wykonywanie trajektorii każdego z robotów,

Pierwsze z podejść można porównać do metody zarządzania ruchem samolotów w przestrzeni powietrznej. Trasy dla każdego z samolotów są ustalane przez kontrolera lotu, który posiada komplet informacji o położeniu i planach wszystkich uczestników ruchu. W zastosowaniu do robotów mobilnych rolę kontrolera przejmuje algorytm, który modyfikuje trasy robotów znajdujących się w nadzorowanym obszarze. Każdy z robotów stosuje się do otrzymanego planu, jednak do wyznaczania sterowania nadal wykorzystuje autonomiczny algorytm. Takie podejście do problemu koordynacji zostało zaproponowane przez Profesora A. Borkowskiego w [5], a jego skuteczność została porównana z innymi metodami w [21].

Drugie podejście do problemu koordynacji ruchu jest podobne do metody stosowanej w kontroli ruchu statków w okolicach portów. Sterowanie nad statkiem przejmuje pilot,

który planuje i wykonuje trasę jednostki. System, który wykorzystuje takie podejście do zarządzania ruchem robotów, przejmuje pełną kontrolę nad efektorami każdego z robotów, planuje trajektorie ruchu i wykonuje je. Rozwiązania tego typu można podzielić na dwie grupy [53]:

- jednoczesne planowanie i optymalizacja trajektorii wszystkich robotów,
- wyznaczanie trajektorii kolejnych robotów w taki sposób, by nie kolidowały one z wcześniej zaplanowanymi.

Teoretycznie jedynie metody z pierwszej grupy są w stanie zagwarantować wyznaczenie optymalnych trajektorii. Operują one na kompletnej przestrzeni rozwiązań, która jest kombinacją przestrzeni trajektorii wszystkich robotów. Złożoność tych algorytmów jest wykładnicza ze względu na wymiar przestrzeni rozwiązań, więc nie jest możliwe ich bezpośrednie, praktyczne stosowanie. Metody, które dają rozwiązanie w akceptowalnym czasie muszą zatem heurystycznie przeszukiwać przestrzeń rozwiązań lub stosować drugie możliwe podejście.

Wyróżnić można dwie wersje problemu koordynacji ruchu robotów:

- przed rozpoczęciem ruchu robotów dostępna jest kompletna lista punktów docelowych wszystkich robotów,
- kolejne punkty docelowe robotów pojawiają się w trakcie pracy systemu, gdy inne roboty już się przemieszczają.

Z praktycznego punktu widzenia druga wersja wydaje się mieć znacznie więcej zastosowań. Co ciekawe, znacząca większość prac poświęconych tej tematyce skupia się jednak na pierwszej wersji, która nie narzuca ścisłych ograniczeń na czas realizacji algorytmu i pozwala łatwo ocenić wyniki. Przegląd typowych rozwiązań tego typu został zaprezentowany w pracy [16]. Większość z nich jest oparta o priorytety przypisane poszczególnym robotom. Każdy robot planuje swoją optymalną trajektorię autonomicznie, a następnie wszystkie trajektorie są analizowane pod kątem występowania kolizji. Najprostszym sposobem obsłużenia wykrytej kolizji jest zmniejszenie prędkości robota o niższym priorytecie [17] lub optymalizacja prędkości wszystkich robotów [42], jednak w przypadku robotów mobilnych łatwo pokazać przykład zakleszczenia dwóch robotów jadących naprzeciw siebie. Rozwiązanie opisane w [34] stosuje losowe objazdy dla robotów o niższych priorytetach, które pozwalają uniknąć kolizji i zakleszczeń. Metoda opisana w [8] wyznacza trajektorie robotów poczynając od tych z najwyższym priorytetem; kolejne roboty muszą traktować wcześniejsze trajektorie jako poruszające się przeszkody i planować swoje rozwiązanie tak, by nie kolidować z już istniejącymi. Taki algorytm może jednak nie wyszukać poprawnego rozwiązania, nawet gdy takie istnieje – robot o wysokim priorytecie może uniemożliwić dotarcie

do celu innym robotom. Lepsze rozwiązania można uzyskać stosując zmienne priorytety robotów. W pracach [15] oraz [16] autorzy prezentują metodę optymalizacji polegającą na wielokrotnym planowaniu ścieżek, przy zastosowaniu różnych układów priorytetów. Pozwala ona na wybranie rozwiązania o najmniejszej sumarycznej długości trajektorii oraz minimalizuje prawdopodobieństwo pominięcia poprawnego rozwiązania.

Wszystkie rozwiązania, które planują trajektorie dla kolejnych robotów tak, by nie kolidowały one z wcześniej zaplanowanymi, można zaadaptować do problemu wyznaczania tras „w czasie rzeczywistym", czyli w sytuacji gdy kolejne punkty docelowe pojawiają się w trakcie realizacji ruchu przez inne roboty. W tym celu konieczne jest oszacowanie maksymalnego czasu trwania obliczeń i wyznaczenie trajektorii robota, zakładając że jej wykonanie rozpocznie się dokładnie po oszacowanym czasie. Niestety wszystkie istniejące rozwiązania charakteryzują się nieliniową (zwykle przynajmniej kwadratową ze względu na liczbę robotów) złożonością obliczeniową, więc nie nadają się do bezpośredniego zastosowania do koordynacji ruchu wielu robotów w rozległym środowisku.

Zagadnienia związane z nawigacją i zarządzaniem ruchem robotów znajdują się obecnie w centrum zainteresowań badaczy zajmujących się robotami mobilnymi. Efektem jest ogromna liczba publikacji prezentujących rozwiązania poszczególnych problemów. Można stwierdzić, że wszystkie typowe zadania z tej dziedziny zostały już rozwiązane. Istniejące podejścia nie są jednak doskonałe. Każde rozwiązanie charakteryzuje się ograniczeniami dokładności lub skali, może też nie być przydatne we wszystkich sytuacjach, lub działać tylko dla określonego typu robotów.

Pomimo tak ogromnego zainteresowania tą problematyką, autorowi nie udało się znaleźć prac, których celem byłoby analizowanie różnych własności pozafunkcjonalnych algorytmów związanych z przemieszczaniem robotów. Zagadnienia integracji opisanych rozwiązań w spójne systemy, które charakteryzowałyby się skalowalnością, rozszerzalnością i niezawodnością pozostają nierozwiązane.

Skuteczne rozwiązanie problemów przemieszczania jest niezbędne w każdym systemie wykorzystującym roboty mobilne. Proponowane w niniejszej pracy podejście do projektowania i implementowania systemów zarządzających robotami pozwala na realizację takich rozwiązań. Przykładowa implementacja systemu zarządzającego ruchem robotów zostanie zaprezentowana w dalszej części pracy.

5 Teza pracy

Systemy wykorzystujące grupy robotów mobilnych do realizacji zadań mają szansę znaleźć liczne zastosowania w przemyśle i życiu codziennym. Mogą realizować różnorodne zadania związane z transportem, produkcją, magazynowaniem, ochroną czy sprzątaniem, zapewniając przy tym wysoką dokładność i wydajność. Warunkiem wdrożenia tego typu systemów jest, by charakteryzowały się one cechami gwarantującymi skuteczne działanie i możliwości rozbudowy, czyli spełniały podstawowe wymagania pozafunkcjonalne. Muszą one być skalowalne, czyli muszą umożliwiać zwiększanie wydajności przetwarzania poprzez dodanie nowych zasobów (robotów, jednostek obliczeniowych). Muszą być rozszerzalne, by możliwe było modyfikowanie zakresu ich zadań bez przerywania działania. Muszą także być odporne na awarie, zapewniając wydajność i niezawodność działania. Sposób ich budowy powinien umożliwiać wykorzystanie istniejących komponentów programowych, aby koszty realizacji były jak najmniejsze.

Przedstawiony w poprzednim rozdziale przegląd architektur systemów zarządzających działaniem robotów pozwala stwierdzić, że nie istnieją dostatecznie zaawansowane metodologie i paradygmaty tworzenia wysokiej jakości systemów zarządzających działaniem grup robotów mobilnych. Badania związane z programowaniem robotów mobilnych skupiają się na rozwiązywaniu oddzielnych problemów, związanych głównie z nawigacją i organizacją ruchu. Realizowane implementacje mają charakter eksperymentalny i najczęściej są silnie związane z wykorzystywanym sprzętem. Publikowane rozwiązania są na ogół wartościowe, jednak ich praktyczne zastosowanie wymaga, by zostały zintegrowane w spójny system.

Można stwierdzić, że opracowanie sposobów projektowania i implementowania systemów informatycznych zarządzających działaniem robotów mobilnych jest potrzebne. Jest warunkiem wykorzystania potencjalnych korzyści, jakie mogą płynąć z praktycznych zastosowań robotów mobilnych. W niniejszej pracy zaproponowane zostanie podejście do projektowania tego typu systemów, oparte o wykorzystanie paradygmatu agentów programowych. Celem pracy jest wykazanie, że:

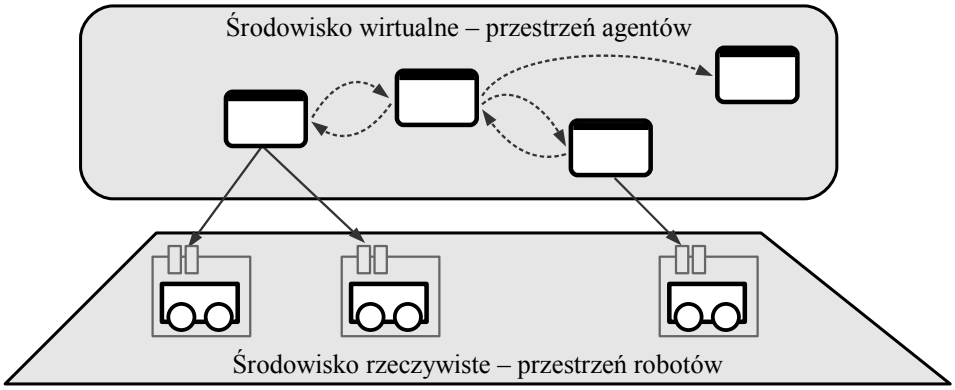
Metodologia tworzenia systemów agentowych zarządzających grupami robotów mobilnych, oparta na oddzieleniu abstrakcji agenta od obiektu robota, oraz wykorzystaniu wielu modeli środowiska, pozwala na zaprojektowanie systemu zapewniającego skalowalność, rozszerzalność, odporność na awarie oraz możliwość wielokrotnego wykorzystania fragmentów oprogramowania sterującego.

Oddzielenie abstrakcji agenta od obiektu robota osiągnięte zostanie poprzez zdefiniowanie przestrzeni działania agentów, która jest odseparowana od przestrzeni działania robotów. Będzie ona nazywana przestrzenią wirtualną, lub cyberprzestrzenią. Cyberprze-

strzeń to podstawowe środowisko istnienia i funkcjonowania agentów programowych. Jest realizowane jako maszyna wirtualna łącząca wszystkie komputery dostępne dla agentów systemu. Agenty mogą działać na dowolnym z połączonych komputerów, mogą się ze sobą komunikować za pomocą asynchronicznych wiadomości, mogą korzystać z usług platformy agentowej oraz zewnętrznych interfejsów programowych (np. interfejsów efektorów i sensorów robota).

Separacja komponentu programowego, jakim jest agent, od urządzenia działającego w przestrzeni fizycznej, jakim jest robot, ma charakter logiczny, a nie fizyczny. Platforma agentowa może integrować również komputery lokalne mobilnych robotów, pozwalając agentom na wykonywanie programu na tych komputerach. Możliwe jest zrealizowanie prezentowanego podejścia z wykorzystaniem wyłącznie komputerów robotów, tworząc w ten sposób system wielorobotowy, który nie wymaga infrastruktury w środowisku działania.

Każdy z robotów w systemie musi definiować interfejs programowy udostępniający wszystkie operacje wspierane przez sprzęt. Interfejsy różnych typów robotów będą różne, ale wszystkie muszą być dostępne dla przynajmniej jednego agenta w platformie agentowej. Agent ten będzie implementował zestaw usług robota udostępnionych innym agentom w systemie. Pozostałe agenty mogą traktować te usługi sprzętowe podobnie jak usługi czysto programowe – nie muszą być świadome sposobu ich realizacji. Dzięki takiemu podejściu proaktywna logika systemu, która jest odpowiedzialna za realizację funkcjonalności, może być niezależna od rodzaju robotów wykonujących czynności w rzeczywistym środowisku. Ogólną ideę logicznego rozdzielenia systemu agentowego od zarządzanych robotów prezentuje rysunek 5.1.



Rysunek 5.1: Ogólna koncepcja oddzielenia abstrakcji agenta od obiektu robota. Agenty działające w przestrzeni wirtualnej komunikują się między sobą za pomocą asynchronicznych wiadomości oraz sterują robotami, wywołując metody ich interfejsów programowych.

Należy zaznaczyć, że agent wykorzystujący interfejs pojedynczego robota jest częścią

systemu agentowego, a nie robota. Różnica jest bardzo znacząca, ponieważ umożliwia wykorzystanie cennych zalet, jakie oferuje paradygmat agentowy. W szczególności pozwala na:

- Wykonywanie programu agenta na dowolnym komputerze w platformie agentowej oraz przenoszenie programu agenta pomiędzy różnymi komputerami. Pozwala to na udostępnienie algorytmom potrzebnej mocy obliczeniowej i ograniczenie wymagań dotyczących autonomicznych komputerów robotów. Rozbudowa systemu o nowe funkcje, które mogą wymagać złożonych obliczeń będzie się mogła odbyć bez wymiany lub modyfikacji robotów.

- Monitorowanie poprawności działania agenta sterującego robotem przez inne agenty w platformie. Awaria wbudowanego programu sterującego robotem unieruchamia urządzenie i czyni go bezużytecznym z punktu widzenia systemu. Możliwość monitorowania poprawności działania agenta sterującego robotem może zabezpieczyć system przed taką sytuacją, zwiększając jego niezawodność.

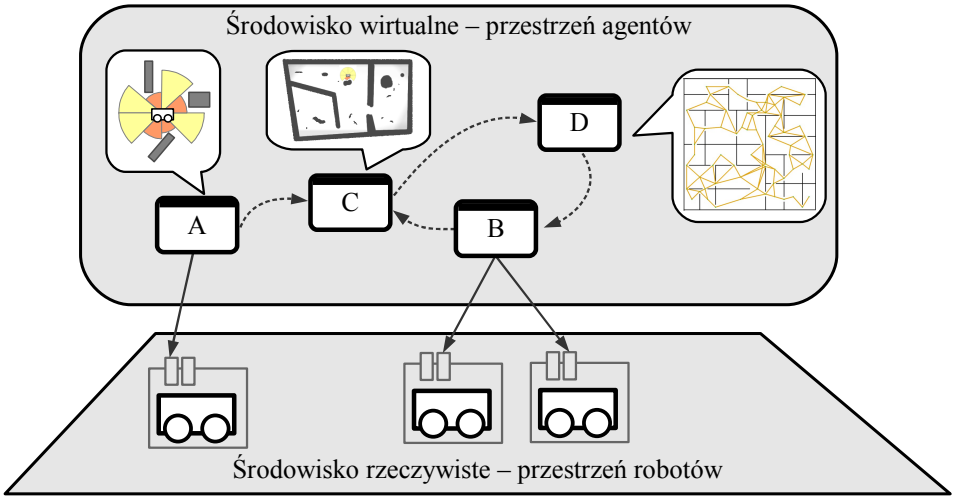
- Wykrywanie uszkodzeń sprzętu przez agenta sterującego wadliwym robotem. Agent uruchomiony na zewnętrznym komputerze jest w stanie wykryć uszkodzenie robota i uruchomić odpowiednie procedury obsługi błędu.

- Modyfikowanie sposobu działania robota bez konieczności usuwania go z systemu. Podmiana agenta sterującego robotem może zmienić sposób wykonywania czynności przez robota oraz dodać nowe usługi robota. Platforma agentowa umożliwia dokonanie zmiany programu agenta bez konieczności zatrzymywania systemu.

Wprowadzenie logicznej separacji pomiędzy fizycznym urządzeniem, jakim jest robot, a komponentem programowym, który robotem steruje, jest więc podstawowym warunkiem osiągnięcia pożądanych własności pozafunkcjonalnych systemów zarządzających działaniem robotów.

Zastosowanie paradygmatu agentów programowych wprowadza modularyzację systemu. Funkcjonalność jest podzielona pomiędzy poszczególne agenty, a asynchroniczna komunikacja wymusza obsługiwanie sytuacji, w której jeden z agentów nie działa poprawnie. Dzięki temu stosunkowo łatwo można wyłączyć cześć funkcji systemu bez wpływania na działanie pozostałych. Można także zrealizować kilka rozwiązań tego samego problemu i stosować je w zależności od potrzeb lub sytuacji. Takie podejście zostanie zastosowane wielokrotnie w prezentowanym dalej systemie agentowym zarządzającym robotami. Różne agenty, odpowiedzialne za sterowanie jednym lub kilkoma robotami, będą stosowały algorytmy bazujące na różnych **modelach** robotów oraz ich środowiska. Agent realizujący algorytm nawigacji reaktywnej będzie postrzegał środowisko robota jako listę wartości

zwracanych przez sensory odległości. Agent budujący mapę będzie konstruował siatkę zajętości na podstawie tych wartości. Dla agenta planującego trasę środowisko będzie grafem nieskierowanym, a robot będzie skojarzony z wierzchołkiem lub krawędzią tego grafu. Schemat współpracy w takim systemie prezentuje rysunek 5.2. Agenty, które pracują na wyższym poziomie abstrakcji będą postrzegać roboty jedynie jako usługi innych agentów w systemie.



Rysunek 5.2: Przykład wykorzystania różnych modeli środowiska przez agenty w systemie. Agenty **A** i **B** postrzegają otoczenie robota poprzez jego sensory odległości. Agent **C** buduje mapę środowiska na podstawie informacji od agentów **A** i **B**. Agent **D** planuje trasy dla robotów wykorzystując model grafowy.

Stosowanie różnych modeli tego samego środowiska może również pozwolić na rozwiązanie problemu skalowalności elementów systemu. Problemy występujące w systemie wykorzystującym roboty można, ze względu na zasięg, podzielić na lokalne i globalne. Rozwiązania problemów lokalnych, dotyczących jednego robota, niewielkiej grupy robotów czy fragmentu środowiska, można skalować poprzez powielanie rozwiązań i dodawanie mocy obliczeniowej do systemu. Problemy globalne, dla których koszt obliczeniowy jest zależny od rozmiarów całego środowiska czy systemu, są znacznie trudniej skalowalne. W pracy zaproponowane zostanie rozwiązanie tego typu problemów oparte o wykorzystanie hierarchii modeli tego samego środowiska, które reprezentują różne jego fragmenty na różnych poziomach abstrakcji. Przedstawiona zostanie przykładowa implementacja rozwiązania problemu globalnego planowania tras robotów w relatywnie dużych środowiskach.

Zaprezentowana teza pracy zostanie wykazana w dwóch etapach. Pierwszym będzie teoretyczna analiza własności projektu zrealizowanego systemu zarządzającego robotami

mobilnymi. Każdy element projektu zostanie przeanalizowany pod kątem wymagań pozafunkcjonalnych, które zostały zawarte w tezie. Następnie zaprezentowana zostanie implementacja systemu zarządzającego ruchem grupy robotów, która została stworzona na potrzeby niniejszej pracy. Przedstawiona zostanie architektura zastosowanego systemu agentowego, algorytmy rozwiązujące poszczególne problemy nawigacji, oraz sposób integracji i współdziałania wszystkich komponentów. Stworzony system zostanie przetestowany, w celu wykazania, że cechuje się wymienionymi w tezie właściwościami pozafunkcjonalnymi.

6 Koncepcja rozdzielenia przestrzeni działania agentów i robotów oraz wykorzystania wielu modeli rzeczywistości przez agenty sterujące robotami

W rozdziale 4.2.2 wskazane zostały problemy, jakie powstać mogą na skutek utożsamienia robota mobilnego i agenta. Zastosowanie takiego bezpośredniego podejścia powoduje trudności z wprowadzaniem modyfikacji do algorytmów sterujących, komplikacje podczas dodawania nowych typów zadań oraz konieczność definiowania skomplikowanych protokołów komunikacyjnych. Powstawanie takich problemów jest zaskakujące, jeżeli zastanowić się nad typowymi zaletami płynącymi ze stosowania podejścia agentowego w innych dziedzinach. Spodziewanym efektem są najczęściej bardzo elastyczne architektury, które dzięki osłabionym powiązaniom pomiędzy komponentami z łatwością dają się skalować i rozszerzać.

Osiągnięcie takich właściwości w systemach zarządzających grupami robotów mobilnych jest możliwe. Wymaga jednak bardziej dogłębnej analizy procesów zachodzących podczas pracy grupy robotów.

Współpraca asynchronicznych bytów, jakimi są agenty programowe, wymaga komunikacji, której wydajność jest często wąskim gardłem systemów. Jeżeli system agentowy generuje zbyt wiele komunikatów, możliwość jego skalowalności jest ograniczona. Pomimo tego, że algorytmy implementowane przez agenty teoretycznie nie mają wymagań przewyższających dostępne zasoby, sama obsługa komunikacji, realizowana przez niższe warstwy środowiska agentowego, powoduje spadek wydajności lub nawet całkowite wstrzymanie działania systemu. Dlatego podział implementowanej funkcjonalności na wątki asynchroniczne należy wykonać tak, by ilość komunikatów była minimalizowana. Taki podział zwiększa również stopień uzyskanej autonomii poszczególnych agentów. Każdy agent powinien być w stanie wykonywać zadania możliwie samodzielnie lub współpracując z innymi agentami, ale nie powinien synchronizować wykonania każdej atomowej czynności.

Dodatkowo zastanowić się należy, które algorytmy czy też grupy algorytmów będą często modyfikowane lub rozbudowywane. Taka analiza pozwoli zmniejszyć ilość komponentów (agentów), których implementację trzeba zmodyfikować dla dodania nowych funkcjonalności systemu.

Konieczność zapewnienia wysokiej dostępności systemu również będzie miała swoje odzwierciedlenie w architekturze i sposobie współpracy agentów. Niedopuszczalna jest sytuacja, w której awaria sprzętu może nie zostać wykryta lub awaria oprogramowania może spowodować utratę kontroli nad sprzętem. Architektura systemu agentowego powinna zapewniać kontrolę nad przebiegiem prac wykonywanych przez urządzenia sprzętowe.

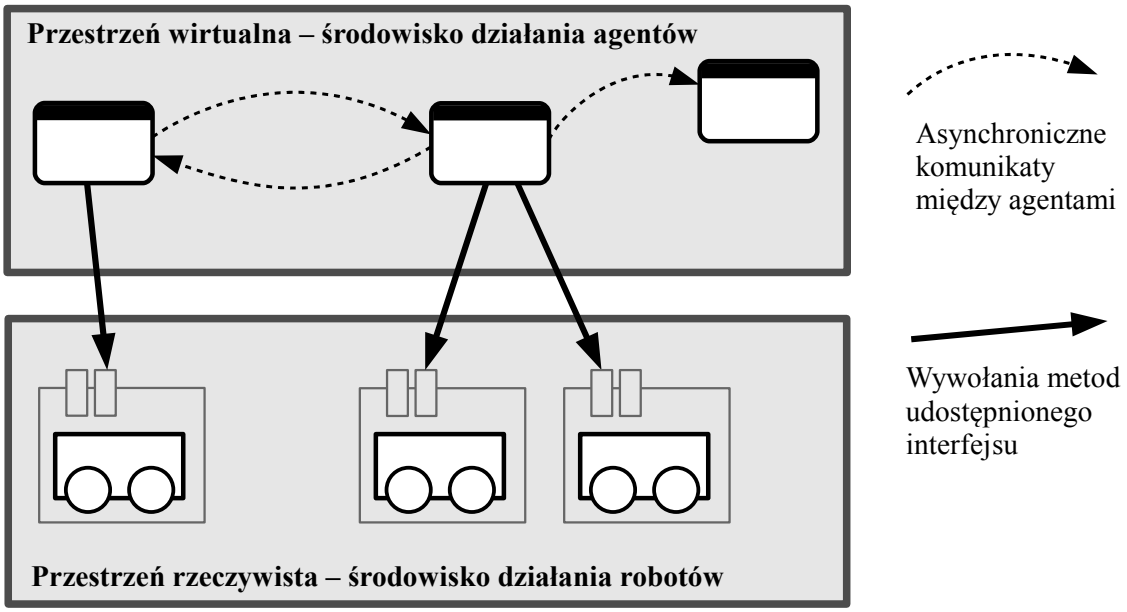
Opisywany w dalszej części rozdziału paradygmat projektowania i implementowania

systemów zarządzających robotami mobilnymi jest oparty na koncepcji przedstawionej w pracy [6], a rozwiniętej w [7], [5] oraz [77].

6.1 Przestrzeń działania robotów i agentów

Jak wskazano w poprzednich rozdziałach, utożsamianie agenta z obiektem robota jest stosowane bardzo często. Porównanie cech przypisywanych obu agentom i robotom skłania projektantów do takiego podejścia. Powoduje jednak liczne problemy w kontekście możliwości zapewnienia wysokiej jakości konstruowanego systemu. Można stwierdzić, że spełnienie wszystkich wymagań pozafunkcjonalnych, jakie stawiane są przed złożonymi systemami informatycznymi, wymaga porzucenia „klasycznego" podejścia, które zakładało bezpośrednie utożsamianie obiektu robota z autonomicznym agentem.

Przy takim założeniu można skupić się na projektowaniu systemu agentów programowych, który spełniał będzie stawiane wymagania (funkcjonalne i pozafunkcjonalne) wykorzystując w tym celu dostępne roboty. System agentowy ma zostać zaprojektowany jako samodzielny byt, który do istnienia i działania nie musi wymagać ustalonej ilości robotów określonego typu.



Rysunek 6.1: Ogólna koncepcja separacji przestrzeni działania robotów i agentów.

Powstająca logiczna separacja robotów i agentów obecnych w systemie tworzy potrzebę zaistnienia nowego środowiska, nowej przestrzeni, w której działać będą agenty. Nowa przestrzeń działania agentów będzie nazywana przestrzenią wirtualną lub cyberprzestrzenią. Z implementacyjnego punktu widzenia będzie ona środowiskiem agentowym czyli maszyną

wirtualną, w której uruchamiane będą agenty programowe.

Roboty są w tym kontekście traktowane jako narzędzia, fizyczne efektory systemu informatycznego, które mają znane możliwości i potrzeby. Roboty są obecne w przestrzeni fizycznej, która jest środowiskiem ich działania. Środowisko to może mieć określone ograniczenia (wymiary), charakterystykę, może być znane lub nie, może być statyczne lub dynamiczne itp. Każdy robot znajduje się w dokładnie określonej lokalizacji w środowisku, a dzięki swojej mobilności może lokalizację zmieniać. Właściwości otaczającego go fizycznego środowiska może postrzegać dzięki posiadanym sensorom, a dzięki posiadanym efektorom może zmieniać stan środowiska.

Całość dostępnej funkcjonalności każdego robota jest udostępniona poprzez jawnie zdefiniowany interfejs. Metody interfejsu są wywoływane przez agenta, który aktualnie sprawuje kontrolę nad danym robotem. Ogólna idea podejścia została zilustrowana na rysunku 6.1.

Podejście nie zakłada utożsamiania obiektu robota z pojedynczym agentem, dzięki czemu konstruowany system będzie się charakteryzował następującymi własnościami:

- W systemie może istnieć agent, który nie steruje żadnym robotem.
- Agent może sterować jednym robotem, może też jednocześnie sterować wieloma robotami.
- Jednym robotem może jednocześnie sterować wiele agentów.
- Agent może, na pewien czas lub na stałe, przekazać sterowanie nad przypisanym robotem innemu agentowi.
- Agent może utworzyć innego agenta, implementującego algorytm sterowania robotem potrzebny w danej sytuacji.

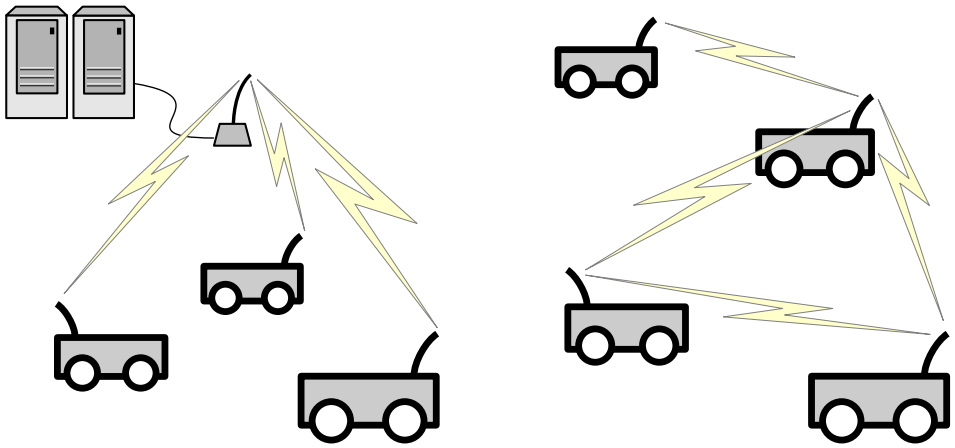
Wymienione cechy zapewniają projektantowi systemu swobodę i elastyczność, która pozwala na skuteczne spełnienie wszystkich funkcjonalnych i pozafunkcjonalnych wymagań. Przykład projektu systemu, który charakteryzuje się wszystkimi pożądanymi cechami zaawansowanego systemu informatycznego, zostanie przedstawiony w następnym rozdziale.

6.2 Sposoby realizacji przestrzeni działania agentów

Implementacja opisywanego do tej pory, abstrakcyjnego środowiska działania agentów wymaga dostarczenia mocy obliczeniowej, pamięci i infrastruktury komunikacyjnej, czyli odpowiedniej liczby komputerów połączonych w sieć. W zależności od rodzaju budowanego systemu, przestrzeń wirtualna, w której działać mają agenty, może zostać zrealizowana na różne sposoby:

- jako sieć komputerów połączonych z robotami bezprzewodowo (rysunek **6.2a**),
- jako sieć typu ad-hoc utworzona przez komputery robotów (rysunek **6.2b**),
- jako połączenie powyższych rozwiązań.

W przypadku gdy roboty mają działać w znanym środowisku, w którym możliwe jest zainstalowanie infrastruktury, najlepszym rozwiązaniem wydaje się umieszczenie większości realizowanych obliczeń na klastrze komputerów fizycznie odseparowanych od robotów. Komputery muszą być połączone szybką siecią, musi też zostać zapewniona metoda wydajnej komunikacji z robotami – możliwie szybka sieć bezprzewodowa.



Rysunek 6.2: Metody realizacji przestrzeni wirtualnej, w której działają agenty programowe. W sytuacji **a.** wykorzystana jest sieć komputerów połączona bezprzewodowo z robotami. W sytuacji **b.** roboty tworzą sieć typu ad-hoc.

Utworzona w ten sposób maszyna wirtualna charakteryzuje się kilkoma bardzo pożądanymi cechami:

- jest łatwo skalowalna – dodanie kolejnych komputerów nie stanowi problemu, a pozwala na swobodne zwiększanie mocy obliczeniowej udostępnionej agentom,
- koszt jej realizacji jest niski – stacjonarne jednostki obliczeniowe są znacznie tańsze od montowanych na robotach mobilnych,
- zmniejsza wymagania energetyczne robotów, które nie muszą posiadać autonomicznych, energochłonnych komputerów o dużej mocy.

Najpoważniejszym problemem są z pewnością bardzo wysokie wymagania dotyczące wydajności komunikacji bezprzewodowej, zarówno pod kątem przepustowości, jak i opóźnień. Umieszczenie algorytmu sterującego silnikami robota na zdalnym komputerze powoduje konieczność przesyłania wszystkich poleceń do niskopoziomowych sterowników za pośrednictwem sieci bezprzewodowej, co generuje duże obciążenia i ogranicza dopuszczalne maksymalne prędkości robotów. Rozwiązaniem tego problemu jest przeniesienie podstawowych algorytmów sterujących silnikami i efektorami na lokalny komputer robota – są to algorytmy o niskiej złożoności, więc nie zwiększają znacząco wymagań sprzętowych.

W przypadku braku możliwości zainstalowania infrastruktury w środowisku rzeczywistym, wszystkie obliczenia realizowane przez agenty muszą być wykonywane na połączonych w sieć typu ad-hoc [57] komputerach robotów (rysunek 6.2, sytuacja **b**). Z punktu widzenia agentów efekt powinien być niemal identyczny, jak w poprzednim przypadku. Utworzona przestrzeń działania agentów powinna być logicznie odseparowana od fizycznych robotów.

Oczywiście znacząco zwiększa to wymagania dotyczące wydajności komputerów, w które wyposażone są roboty, ma jednak również sporo zalet. Przede wszystkim umożliwia szybkie uruchamianie systemu w środowiskach nieznanych i nieprzystosowanych, co w wielu sytuacjach jest podstawowym wymaganiem funkcjonalnym. Przykładem może być system wielorobotowy wykorzystywany podczas akcji ratowniczych: w pożarach, katastrofach budowlanych czy trzęsieniach ziemi. Nie może tu być mowy o założeniach dotyczących infrastruktury obecnej w środowisku.

Poza znacznie większymi kosztami realizacji, implementacja maszyny wirtualnej w sieci typu ad-hoc powoduje dodatkowe problemy, takie jak:

- konieczność obsługiwania sytuacji rozerwania sieci,
- możliwość zniszczenia części agentów w przypadku awarii jednego z robotów.

Można stwierdzić, że implementacja oparta o sieć typu ad-hoc powinna być wykorzystywana tylko w sytuacji, gdy nie jest możliwe wykorzystanie zewnętrznej infrastruktury komunikacyjnej i obliczeniowej.

Logiczna separacja abstrakcji agenta od obiektu robota i stworzenie niezależnego środowiska działania agentów programowych daje nowe możliwości w zakresie projektowania systemów zarządzających robotami mobilnymi. Pozwala na dzielenie implementowanej funkcjonalności pomiędzy agenty w sposób zwiększający ich autonomię, poprawiający skalowalność i ułatwiający rozszerzanie systemu o nowe elementy. W następnej części rozdziału opisana zostanie metoda konstruowania hierarchii agentów działających w przestrzeni wirtualnej, które zapewniają uzyskanie tych cech systemu.

6.3 Modele przestrzeni rzeczywistej

Agenty programowe wykorzystywane w systemach zarządzających robotami mobilnymi mogą być postrzegane jako implementacje algorytmów sterujących urządzeniami fizycznymi. Oczywiście mogą wykonywać znacznie więcej funkcji, jednak pewne elementy klasycznego sterownika również posiadają. Najważniejszym z nich jest **model** fragmentu rzeczywistości, który jest podstawą podejmowania decyzji, czyli obliczania sterowania. Matematyczny model rzeczywistości jest zawsze obecny w programach komputerowych sterujących urządzeniami rzeczywistymi. Zawsze występuję tzw. cykl sterowania, w którym algorytm pobiera informacje z sensorów i oblicza sterowanie, które zbliża układ do stanu pożądanego.

Jak wspomniano w rozdziale 4.1.1, algorytmy sterujące grupami robotów często dzieli się na celowe i behawioralne (reaktywne) [75]. Metody celowe tworzą plan działania na podstawie jawnie zdefiniowanego modelu rzeczywistości, którym jest np. mapa pomieszczenia, w którym robot ma się przemieszczać. Metody reaktywne obliczają sterowanie na podstawie bieżących odczytów z sensorów. W przypadku reaktywnej nawigacji będzie to nadawanie odpowiednich prędkości kół na podstawie pomiarów odległości od przeszkód. Model systemu nie jest tu definiowany jawnie; jest zaszyty w równaniach implementowanych przez algorytm obliczający sterowanie. Jest to podejście znacznie bliższe rozwiązaniom klasycznej teorii sterowania.

Rodzaj wykorzystywanego modelu jest uzależniony od stosowanego algorytmu. Jednak z punktu widzenia rozważanej w niniejszej pracy jakości systemu, ważniejsze są ograniczenia w możliwościach przetwarzania określonego modelu. Jak wcześniej wspomniano, większość kluczowych problemów związanych ze sterowaniem robotami mobilnymi, została już rozwiązana. Trzeba jednak zaznaczyć, że istniejące rozwiązania takich zagadnień jak lokalizacja, planowanie trasy czy koordynacja ruchu mają bardzo znaczące ograniczenia dotyczące wydajności, wielkości środowiska działania, liczby robotów, czyli maksymalnej wielkości przetwarzanego modelu.

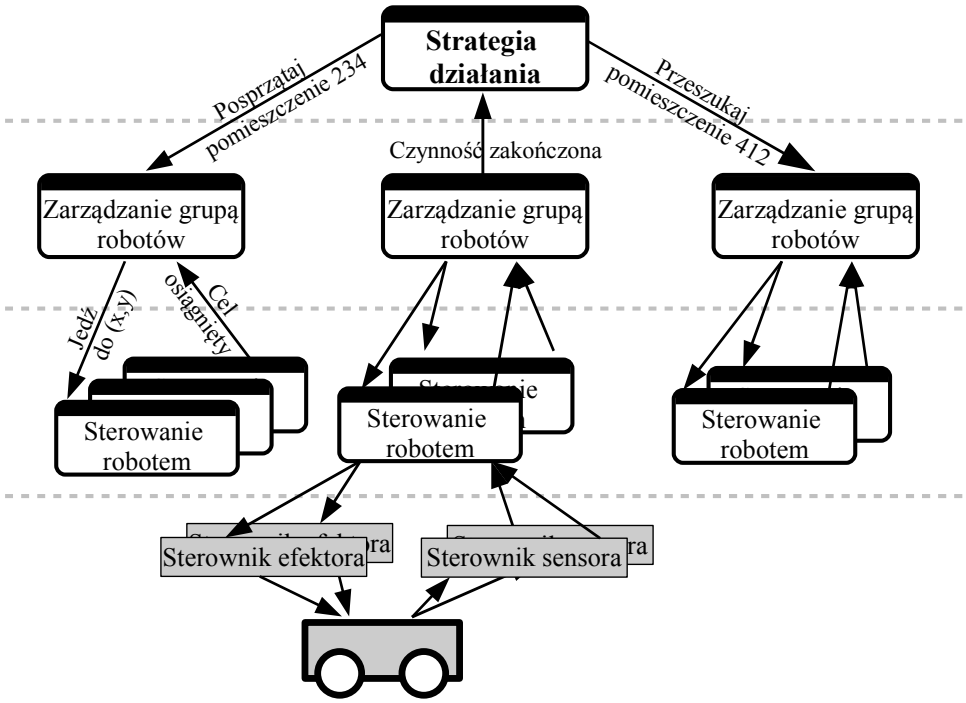
Rozwiązaniem problemu skali, czyli wielkości przetwarzanych w systemie modeli, może być hierarchizacja sterowników, czyli w zbudowanie systemu agentowego zarządzającego grupą lub grupami robotów na różnych poziomach abstrakcji. Takie podejście jest oparte na idei sterowania hierarchicznego, dość szeroko opisywanego w pracach z dziedziny teorii sterowania [3], które w połączeniu z podejściem agentowym pozwala rozwiązać wiele problemów związanych ze skalowalnością i rozszerzalnością systemów.

Model szczegółowy, przetwarzany przez agenta sterującego bezpośrednio efektorami robota, nie może (i nie musi) obejmować całego środowiska. Model agentów zarządzających całym systemem nie musi być przesadnie szczegółowy. Do podjęcia odpowiednich, ogólnych decyzji wystarcza wiedza o wybranych, kluczowych aspektach rzeczywistości. Ilość

poziomów w hierarchii nie musi być ograniczona. W miarę wzrostu skali systemu możliwe jest dodawanie kolejnych warstw, odpowiedzialnych za coraz bardziej ogólne przetwarzanie coraz większych fragmentów systemu, z wykorzystaniem coraz bardziej ogólnych modeli. Taka architektura systemu agentowego będzie w niniejszej pracy nazywana architekturą **wielomodelową**.

Ogólna koncepcja sterowania wielomodelowego była rozwijana już w latach siedemdziesiątych XX wieku [19, 18]. Może ona być z powodzeniem stosowana w wielu zagadnieniach, w których zastosowanie jednego modelu systemu nie jest wystarczające.

Przykład hierarchii agentów zarządzających systemem wykorzystującym roboty ilustruje rysunek 6.3. Agenty na różnych poziomach hierarchii wykorzystują modele systemu na różnych poziomach abstrakcji.



Rysunek 6.3: Hierarchia agentów zarządzających dużym systemem wielorobotowym. Wyższe warstwy operują na modelach o wyższym poziomie abstrakcji; protokoły komunikacyjne muszą opisywać zdarzenia, które można odwzorować w modelu warstwy wyższej.

Warto zaznaczyć, że hierarchia nie musi być statyczna. Implementacja powinna przewidywać możliwość zmiany struktury hierarchii w zależności od potrzeb. Przykładowo struktura agentów widoczna na rysunku 6.3 mogłaby zostać przebudowana poprzez przesunięcie agenta sterującego robotem pomiędzy agentami zarządzającymi grupami, lub poprzez rozdzielenie agenta zarządzającego grupą na kilka tego typu agentów, gdy zajdzie potrzeba

równoległego wykonania większej ilości zadań. Operacje te powinien inicjować agent warstwy wyższej.

Kluczowym problemem podczas projektowania hierarchii agentów zarządzających robotami jest zdefiniowanie protokołów komunikacyjnych pomiędzy agentami z różnych warstw. Protokoły powinny być niezależny od sposobu implementacji warstwy niższej i powinny opisywać zjawiska, zdarzenia i polecenia, które można odwzorować w modelu warstwy wyższej. Tak zaprojektowana komunikacja pozwala na zmianę implementacji algorytmów warstwy niższej, a nawet na dodanie nowych elementów do niższej warstwy, bez konieczności ingerowania w działanie całego systemu, który jest obsługiwany przez wyższe warstwy.

Dobrym przykładem zastosowania architektury wielomodelowej w zarządzaniu robotami mobilnymi może być rozwiązanie problemu wielkości środowiska. Każdy algorytmy planowania trasy operujący na modelu grafowym czy rastrowym, ma ograniczone możliwości przetwarzania. Spowodowane jest to nieliniową złożonością stosowanych algorytmów wyszukiwania ścieżek. Rozwiązaniem jest podzielenie szczegółowego modelu środowiska pomiędzy kilka agentów oraz utworzenie agenta warstwy wyższej, który będzie operował na mapie o mniejszej szczegółowości. Planowanie nie wykraczające poza obszar jednego z agentów warstwy niższej będzie realizowane autonomicznie, bez kontaktowania się z agentem nadrzędnym. Planowanie dłuższych tras będzie realizowane przez agenta warstwy wyższej, który zaplanuje trasę ogólną a następnie zleci opracowanie fragmentów szczegółowych agentom warstwy niższej, przez których fragmenty przebiega trasa ogólna. W miarę powiększania terenu wykorzystywanego przez system, możliwe jest dodawanie kolejnych warstw. Szczegółowy opis implementacji takiego systemu zostanie przedstawiony w następnych rozdziałach.

Inne aspekty działania systemu, jak na przykład przydział zadań czy zapewnianie serwisu robotów, również mogą i powinny być implementowane zgodnie z opisanym schematem. Tworzy to strukturę wielu hierarchii agentów współpracujących ze sobą na różnych poziomach i zapewnia wiele właściwości i możliwości bardzo pożądanych w złożonych systemach informatycznych. Najważniejsze z nich to:

- **Skalowalność**. Agent z założenia jest bytem asynchronicznym, posiadającym przynajmniej jeden własny, niezależny wątek wykonania obliczeń, który może funkcjonować na dowolnym komputerze wchodzącym w skład maszyny wirtualnej tworzącej środowisko agentowe. Rozbudowa zasobów sprzętowych udostępnianych agentom nie stanowi problemu, więc liczba uruchamianych agentów nie jest trwale ograniczana ilością dostępnych zasobów. Natomiast zastosowanie architektury zakładającej istnienie wielu modeli reprezentujących ten sam aspekt systemu na różnych poziomach abstrakcji pozwala na podzielenie wykonywanych obliczeń tak, by żaden z agentów nie miał wymagań niemożliwych do zapewnienia przez dostępną platformę sprzętową.

- **Rozszerzalność**. Implementacja jest podzielona na aspekty, realizowane przez odseparowane, wielowarstwowe hierarchie agentów. W ramach pojedynczej hierarchii, niższe elementy realizują coraz bardziej szczegółowe funkcje. Dlatego im mniejszy jest zakres zmian, tym mniejszego fragmentu implementacji będzie dotyczył. Zmiana wprowadzana na określonym poziomie hierarchii nie powinna mieć wpływu na implementację poziomów wyższych, pod warunkiem zachowania przez nową implementację zgodności protokołów komunikacji pomiędzy warstwami.

6.4 Definicja agenta i typy agentów

W dalszych rozdziałach pracy model formalny agenta i systemu agentowego nie będzie wykorzystywany, ponieważ nie jest to niezbędne do udowodnienia tezy. Podana zostanie tu jednak wykorzystywana definicja agenta oraz przyjęty model jego architektury i działania. Wyszczególnione zostaną również typy agentów, jakie istnieć będą w prezentowanym dalej systemie, oraz opisane zostaną różnice między nimi.

Przyjęta definicja agenta i systemu agentowego jest oparta na modelu M-Agenta, który został zaprezentowany w rozdziale 4.2.1. Zastosowane modyfikacje formalizmu mają na celu jego przystosowanie do specyfiki prezentowanego w pracy podejścia do projektowania systemów agentowych zarządzających robotami.

Agent to komponent programowy, który:

- posiada własny wątek przetwarzania i może działać współbieżnie z innymi agentami,
- posiada zdolność odbierania i wysyłania wiadomości,
- do poprawnego uruchomienia nie wymaga innych agentów.

Agenty są uruchamiane w **platformie agentowej**. Platforma agentowa tworzy podstawowe środowisko działania agentów (cyberprzestrzeń), umożliwia ich uruchamianie i działanie oraz zapewnia możliwość komunikacji za pomocą asynchronicznych wiadomości. Platforma agentowa z uruchomionymi w niej agentami tworzą **system agentowy**.

Każdy agent implementuje **algorytm**, który pozwala mu na osiąganie **celów**. Algorytm agenta to definicja rodzaju akcji, jaka zostanie podjęta w określonych warunkach. Warunki są zdeterminowane stanem wewnętrznym agenta, stanem modelu środowiska obserwowanym przez agenta oraz bieżącym celem agenta. Można więc stwierdzić, że algorytm s agenta to funkcja:

$$s(n, m, q) = (x, n') \tag{6.1}$$

gdzie:

n, n' – stany agenta, $n, n' \in N$, N – przestrzeń stanów agenta;
m – stan modelu środowiska, $m \in M$, M – przestrzeń stanów modelu środowiska;
q – cel agenta, $q \in Q$, Q – przestrzeń celów agenta;
x – akcja agenta, $x \in X$, X – przestrzeń akcji agenta.

Formalnie, agenta a można zdefiniować jako:

$$a = (N, n, M, m, Q, q, X, x, i, s) \tag{6.2}$$

gdzie i jest funkcją obserwacji, która przekształca stan środowiska μ w stan modelu środowiska:

$$i(\mu) = m \tag{6.3}$$

Środowiskiem działania agentów jest cyberprzestrzeń. Percepcja agentów może jednak wychodzić poza nią, ponieważ mogą one być w stanie obserwować stan robotów oraz ich otoczenia, czyli postrzegać środowisko rzeczywiste. Stan całego środowiska μ, podobnie jak stan modelu środowiska m, należy więc zdefiniować jako pary:

$$\mu = (\mu_r, \mu_v), \qquad m = (m_r, m_v) \tag{6.4}$$

gdzie:
μ_r – stan środowiska rzeczywistego – pozycje robotów, lokalizacja zadań, itp.
m_r – stan modelu środowiska rzeczywistego,
μ_v – stan środowiska wirtualnego – działające agenty, relacje pomiędzy nimi, wykonywane zadania, wymieniane wiadomości, itp.
m_v – stan modelu środowiska wirtualnego, w tym odebrane wiadomości.

Taki podział środowiska i modelu środowiska dzieli na podzbiory również cele agentów:

$$Q = Q_r \cup Q_v \tag{6.5}$$

Cel w środowisku rzeczywistym, $q_r \in Q_r$, to modyfikacja stanu robota, robotów lub środowiska działania robotów. Funkcjonalność systemu wykorzystującego roboty będzie więc determinowana przez elementy zbioru Q_r.

Każdy element q_r przestrzeni Q_r można wyrazić jako ciąg innych elementów zbioru Q_r.

$$q_r = q_{r1}, q_{r2}, ..., q_{rn} = (q_{ri})_{i=1}^{n}, \qquad q_r, q_{r1}, q_{r2}, ..., q_{rn} \in Q_r \tag{6.6}$$

Przykładem takiego przekształcenia może być rozbicie celu $move(r_m, (x_c, y_c))$, którego wypełnienie wymaga przemieszczenia robota r_m do punktu (x_c, y_c). Cel ten może zostać przekształcony w ciąg celów:

$$move(r_m, (x_c, y_c)) = move(r_m, (x_p, y_p)), move(r_m, (x_q, y_q)), move(r_m, (x_c, y_c)) \quad (6.7)$$

Jeżeli agent a akceptuje cel q_r, to jego algorytm definiuje sposób osiągnięcia celu q_r. Sposobem może być wyliczenie odpowiedniego sterowania dla efektorów robota, ale także przekształcanie celu w ciąg celów, lub też przekazanie celu innemu agentowi. Dekompozycja i rozproszenie wykonania zadania będzie osiągana poprzez rozdzielenie elementów ciągu celów pomiędzy różne agenty, które potrafią poszczególne cele składowe zrealizować.

Cele w przestrzeni wirtualnej, $q_v \in Q_v$, to modyfikacja stanu cyberprzestrzeni. Może ona dotyczyć utworzenia nowego agenta, wykonania zadania obliczeniowego czy wysłania określonego rodzaju wiadomości. Tego typu zadania mogą być wykonane bardzo szybko, w porównaniu do celów w przestrzeni rzeczywistej. Podobnie, jak cele z przestrzeni Q_r, mogą one być przekształcane w ciągi celów składowych, jednak najczęściej będą realizowane w całości, przez jednego agenta.

Tak zdefiniowane cele agentów pozwalają na wprowadzenie podziału agentów. Ze względu na rodzaj wykonywanych celów agenty można podzielić na:

- realizujące cele w środowisku rzeczywistym,
- realizujące wyłącznie cele w środowisku wirtualnym.

Ze względu na sposób realizacji celów, wyróżnić można agenty:

- wykorzystujące inne agenty do realizacji fragmentów własnego celu,
- realizujące cel samodzielnie, poprzez wykonanie algorytmu modyfikującego stan środowiska.

Ze względu na źródło celu wyróżnić można agenty:

- otrzymujące cele od innych agentów za pośrednictwem wiadomości – tzw. agenty reaktywne,
- wykrywające cele na podstawie analizy pozostałych zmian stanu środowiska – tzw. agenty proaktywne.

Taki podział pozwala wyróżnić osiem różnych typów agentów. Jednak nie wszystkie kombinacje cech mogą występować samodzielnie. Następujące trzy rodzaje agentów będą się składały na opisywany w następnych rozdziałach system agentowy zarządzający działaniem robotów:

- ***Master Agent***, czyli **MA** – agent realizujący cele w środowisku rzeczywistym, który rozbija cel na ciąg celów składowych i wykorzystuje inne agenty do ich realizacji. Realizuje on złożone zadania, które mogą wymagać wielu robotów i długiego czasu realizacji. Odpowiada za współpracę oraz koordynację działania agentów sterujących robotami. Może być proaktywny lub reaktywny.
- ***Worker Agent***, czyli **WA** – agent realizujący cele w środowisku rzeczywistym, który steruje działaniem robota lub robotów w celu zmodyfikowania stanu środowiska rzeczywistego. Jego cele są stosunkowo nieskomplikowane i nie wymagają do realizacji wykorzystania innych agentów. Są dodawane przez agenta typu MA. Jest to agent reaktywny.
- ***Server Agent***, czyli **SA** – agent realizujący cele w środowisku wirtualnym. Realizuje on zadania obliczeniowe zlecane przez inne agenty, a wyniki zwraca za pomocą wiadomości. Pełni więc funkcję serwera, choć zadania realizuje asynchronicznie. Jest to agent reaktywny.

Właściwości i możliwości każdego z agentów będą determinowane jego typem. Na podstawie typu będzie można stwierdzić, jakie wymagania ma określony agent, z jakich usług musi korzystać oraz jakie aktywności będzie przejawiał.

Agenty typu MA muszą być świadome istnienia innych agentów, które mogłyby zostać wykorzystane do realizacji zadania. Będą więc korzystały z mechanizmu lokalizowania agentów realizujących usługi w systemie agentowym, choć same nie będą usług świadczyć. Każdy agent MA będzie posiadał aktywne wątki realizujące obserwację środowiska rzeczywistego i wirtualnego. Zmiany w którymś ze środowisk będą mogły powodować akcje agenta. Algorytm agentów MA nie powinien bezpośrednio wywoływać metod interfejsu programowego robotów. Z punktu widzenia agenta MA, robot powinien być abstrakcyjną usługą, dostępną za pośrednictwem odpowiedniego agenta WA.

Cele agentów typu WA będą związane ze zmianą stanu środowiska rzeczywistego. Agent WA będzie wykorzystywał metody interfejsu programowego robota lub robotów, czyli sterował urządzeniami istniejącymi w przestrzeni rzeczywistej. Implementacja algorytmu agenta będzie więc uzależniona z rodzajami robotów, które agent będzie obsługiwał. Agent WA będzie musiał odbierać wiadomości definiujące cele oraz wysyłać informacje o zakończeniu działania lub błędzie podczas wykonania zadania. Wykonanie zadania w środowisku

rzeczywistym będzie wymagało obserwacji zmian stanu tego środowiska, która będzie realizowana przez aktywny wątek agenta.

Agenty typu SA muszą poprawnie reagować na odebrane wiadomości. Wątek ich wykonania musi być wybudzany jedynie po odebraniu wiadomości z poleceniem wykonania zadania obliczeniowego. Innych akcji ten agent nie będzie wykonywał.

Przyjęta klasyfikacja agentów dzieli komponenty systemu na:

- usługi obliczeniowe, realizowane przez agenty SA,
- usługi sprzętowe, realizowane przez agenty WA,
- wysokopoziomowe usługi będące realizacją wymagań funkcjonalnych systemu – agenty MA.

Podział ułatwi analizę i projektowanie systemu agentowego, który ma spełnić szereg wymagań pozafunkcjonalnych. Ustali jasne relacje pomiędzy komponentami, co pozwoli na identyfikowanie zależności i pomoże we wprowadzaniu modyfikacji w systemie.

7 Architektura systemu zarządzającego grupą robotów mobilnych

Opisana w poprzednim rozdziale koncepcja projektowania agentowych systemów zarządzających grupami robotów mobilnych została zastosowana do zaprojektowania systemu realizującego abstrakcyjne zadania z wykorzystaniem heterogenicznych robotów. W tym rozdziale opisana zostanie architektura systemu oraz podstawowe algorytmy wykorzystywane przez agenty. W następnych rozdziałach zaprezentowana zostanie implementacja prototypowego systemu oraz wyniki eksperymentów.

Projektowany system ma za zadanie rozwiązywać tzw. abstrakcyjny problem wykonywania zadań, który był rozważany w pracach [77] i [7]. Jest on definiowany następującymi warunkami i założeniami:

- Środowisko działania robotów jest z punktu widzenia systemu dwuwymiarowe.
- Środowisko jest znane i jest statyczne lub dynamiczne. Problem detekcji zmian nie jest rozważany, ale zakłada się, że informacja o zmianie jest poprawnie obsługiwana.
- Dostępna jest grupa robotów różnych typów, o różnych właściwościach, możliwościach i wymaganiach, wyposażonych w odpowiednio dobrane sensory i efektory.
- Ograniczenia zasobów energetycznych robotów muszą być uwzględniane i poprawnie obsługiwane przez system.
- Liczba robotów w systemie może nie być stała. Możliwe jest usuwanie robotów oraz ich dodawanie, w tym także robotów nieznanego wcześniej typu.
- Zakłada się, że każdy robot jest w stanie poprawnie wyznaczyć swoją lokalizację i oszacować jej błąd. Konkretne algorytmy realizujące to zadanie nie są rozważane.
- Zadania pojawiają się w systemie z losową częstotliwością i w losowych lokalizacjach.
- Każde zadanie może wymagać do wykonania jednego lub więcej robotów. Mogą występować zadania, które większa liczba robotów wykona szybciej.
- Wykonanie zadania może odbywać się w początkowej lokalizacji tego zadania, ale może także być związane z przemieszczaniem robotów (np. sprzątanie czy poszukiwanie).
- Problem detekcji zadań nie jest rozważany. Informacja o rodzaju i lokalizacji zadania jest znana w chwili pojawienia się zadania.

Przyjęte uproszczenia – znana lokalizacja robotów i zewnętrzna detekcja zadań – nie zmniejszają znacząco ogólności rozwiązania. Dowolna metoda lokalizacji robotów udostępnia na wyjściu oszacowaną, najbardziej prawdopodobną pozycję robota oraz dokładność tego oszacowania. Detekcja zadań jest natomiast często realizowana zewnętrznie względem systemu. W opisywanym problemie można także detekcję traktować jako dodatkowe zadanie.

Przykładami praktycznymi wykorzystania rozwiązania tego problemu mogą być zadania takie jak:

- różnego rodzaju problemy transportowe,
- automatyzacja tzw. elastycznych procesów produkcyjnych,
- inspekcja pomieszczeń, poszukiwania ludzi i przedmiotów,
- sprzątanie pomieszczeń,
- oprowadzanie po muzeach i inne.

Opisywany system składa się z trzech głównych podsystemów:

- podsystem zapewniania dostępności robotów, odpowiedzialny za utrzymywanie urządzeń w gotowości do działania (np. ładowanie akumulatorów, wykonywanie napraw),
- podsystem wykonywania zadań, który realizuje funkcjonalność systemu,
- podsystem planowania tras i koordynacji ruchu, realizujący nawigację robotów – jest wykorzystywany przez dwa pozostałe podsystemy.

7.1 Roboty i zapewnienie ich poprawnego funkcjonowania

Zgodnie z przyjętym paradygmatem projektowania systemów agentowych zarządzających robotami mobilnymi, komponentem sterującym pojedynczym robotem jest agent reaktywny, wykonujący zadania w środowisku rzeczywistym. Powinien on być w stanie realizować zadania długotrwałe, np. polecenia dotyczące jazdy po zadanej trajektorii czy wykorzystania efektorów. W dalszej części opisu będzie on nazywany *Robot Worker Agent* lub RWA.

Jeżeli dany robot jest w stanie utrzymywać na swoim lokalnym komputerze fragment środowiska agentowego, RWA powinien działać na tym właśnie komputerze. Operacje komunikacji RWA z interfejsem sprzętu robota mogą być bardzo częste, a czas reakcji algorytmu sterującego na zmianę stanu obiektu robota powinien być minimalizowany.

Trzeba podkreślić, że agent RWA nie jest częścią oprogramowania robota i nie jest z nim utożsamiany. Jest on częścią systemu agentowego. Różnica jest bardzo znacząca, ponieważ jako część systemu agentowego, agent RWA może wykonywać swój kod na dowolnym komputerze w platformie. Może również zostać w dowolnym momencie zastąpiony przez innego agenta, co umożliwia łatwą modyfikację oprogramowania sterującego pojedynczym robotem.

Protokół komunikacji z każdym RWA jest podzielony na dwie części: wspólną i specjalizowaną. Część wspólna, która jest identyczna dla każdego RWA, pozwala na pobranie informacji o identyfikacji robota, o jego aktualnej lokalizacji oraz o udostępnianych usługach specjalizowanych. Część specjalizowana powinna zawierać przynajmniej komplet metod umożliwiających dostęp do wszystkich niskopoziomowych funkcji sprzętu. Ponadto powinna udostępniać dodatkowe funkcjonalności, pozwalające na realizację bardziej złożonych celów w środowisku rzeczywistym. Funkcjonalność interfejsu specjalizowanego jest pogrupowana w usługi, których nazwy jednoznacznie specyfikują wspierane prze RWA operacje.

Przykładem usługi oferowanej przez robota mobilnego jest zestaw metod sterujących silnikami kół czy też serwomechanizmami kończyn. Specyfika konstrukcji każdego rodzaju robota może powodować konieczność definiowania różnego rodzaju usług niskopoziomowych dla każdego modelu robota. By umożliwić bardziej abstrakcyjne sterowanie ruchem robota, konieczne jest stworzenie usług wyższego poziomu, które umożliwią zlecanie robotowi wykonywania określonej trajektorii. Interfejs powinien umożliwiać przemieszczenie robota po prostej lub po łuku przez zadany okres czasu z zadaną prędkością; powinien również umożliwiać zlecenie wykonania sekwencji tego typu ruchów. Dzięki takiemu podejściu możliwe będzie sterowanie różnymi rodzajami robotów mobilnych przy pomocy jednego algorytmu.

Z punktu widzenia klientów usług robotów, sposób przemieszczenia urządzenia do zadanego punktu, nie jest interesujący. Ważne są jedynie informacje o tym, czy przemieszczenie jest możliwe oraz o szacowanym czasie zakończenia przemieszczania. Dlatego warto, by RWA dostarczały usługi związane z przemieszczaniem na jeszcze wyższym poziomie abstrakcji. RWA powinien implementować możliwie skuteczny algorytm reaktywnej nawigacji. Sposób wykorzystania tego algorytmu do realizacji usługi przemieszczania robota do zadanej lokalizacji w przestrzeni zostanie opisany w podrozdziale poświęconym planowaniu tras i koordynacji ruchu (8.5).

Warto podkreślić podstawową ideę, jaka przyświeca tworzeniu usług robotów. Jest nią ukrywaniu sposobu realizacji fizycznej czynności za jawnie zdefiniowanym interfejsem programowym. Celem takiego postępowania jest umożliwienie programowania robotów na wysokim poziomie abstrakcji, czyli uniezależnienie implementacji algorytmu realizujące-

go określone zadanie od modelu robota, który będzie to zadanie wykonywał. Dlatego też przy dodawaniu nowego typu robota do systemu trzeba zidentyfikować wszystkie czynności wcześniejszych robotów, które nowy robot jest w stanie wykonać i dodać do wspieranych przez nowego RWA usługi związane z tymi czynnościami. Powinno się tak robić nawet wtedy, gdy podstawowym celem dodania nowego robota jest wykonywanie zupełnie innych czynności. Takie postępowanie zwiększa elastyczność systemu i pozwala na zredukowanie ogólnej liczby robotów.

Każdy rodzaj robota może mieć inne potrzeby, które muszą być zaspokajane w celu utrzymania go w stanie gotowości do wykonywania zadań. Najczęściej będą to czynności związane z ładowaniem akumulatorów lub innym sposobem dostarczania energii. Warto również rozważać inne operacje serwisowe, które mogłyby być wykonywane automatycznie.

W środowisku działania robotów wyznaczony musi być jeden lub kilka obszarów, w których umieszczane są roboty nie wykonujące w danej chwili żadnych zadań. Obszary te, zwane zajezdniami robotów, powinny być wyposażone w infrastrukturę umożliwiającą przeprowadzanie operacji serwisowych. Każda zajezdnia jest zarządzana przez jednego agenta, zwanego *Depot Master Agent*. DMA obserwuje stan robotów i w razie potrzeby podejmuje odpowiednią akcję. Czynności serwisowe są wykonywane gdy robot nie realizuje żadnego zadania, lub gdy jest to niezbędne dla zagwarantowania funkcjonowania robota.

Gdy nowy robot pojawia się w systemie, jest przypisywany do jednego z agentów DMA, który staje się jego nadzorcą i opiekunem. Agent DMA odpytuje nowego RWA o implementowane usługi, by być w stanie dostarczyć te informacje agentom odpowiedzialnym za wykonywanie zadań. Funkcjonuje więc jako **broker usług** robotów, dbając jednocześnie o ich prawidłowe działanie.

Protokół komunikacji elementów systemy z agentami DMA przewiduje następujące rodzaje operacji:

- rejestracja oraz wyrejestrowanie robota,
- zapytanie o dostępność robota dostarczającego określoną usługę,
- rezerwacja wybranego robota.

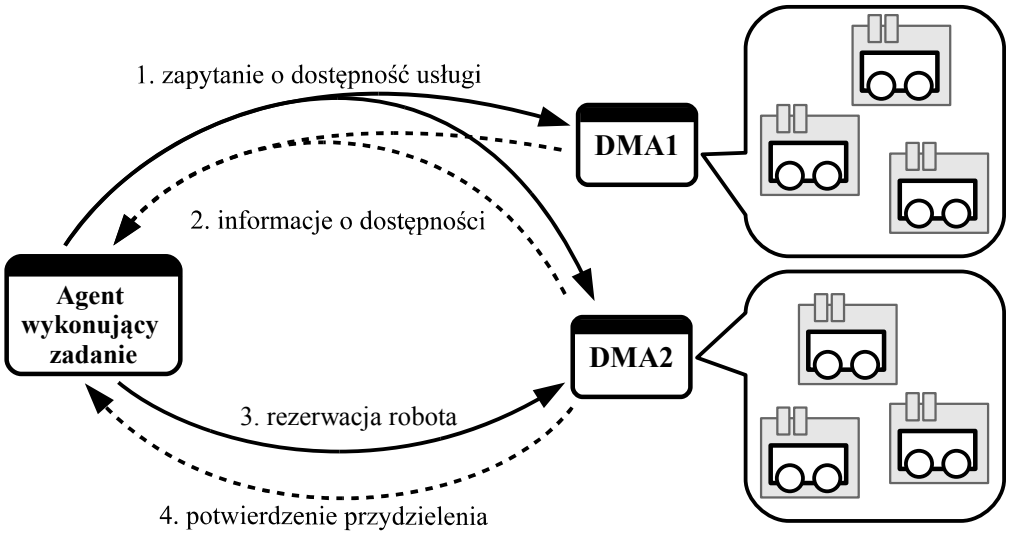
Mechanizm udostępniania informacji o usługach robotów działa na podobnej zasadzie, co znany z technologii Web Services, rejestr UDDI [79]. Usługi udostępnianie przez roboty mobilne oraz inne urządzenia fizyczne, charakteryzują się jednak dodatkowymi ograniczeniami, które nie występują w przypadku Web Services, jak:

- brak możliwości równoległego wykonywania tej samej usługi dla różnych zleceniodawców,

- wykonywanie usługi w określonej lokalizacji w przestrzeni przez relatywnie długi okres czasu.

Te ograniczenia uniemożliwiają bezpośrednie wykorzystanie istniejących standardów (np. wspomniany wcześniej UDDI). W przypadku czysto programowych systemów, opartych na rozproszonych usługach, integracja usług w pożądaną funkcjonalność (zwana aranżacją), może zostać zrealizowana jednokrotnie, podczas implementowania systemu. Rozwiązania wykorzystujące usługi realizowane przez sprzęt wymagają wzięcia pod uwagę zależności czasowych oraz rezerwowania dostawcy usługi na wyłączność. Dlatego też konieczne jest wybieranie urządzeń, które będą realizowały usługę, za każdym razem gdy jest ona potrzebna.

Przydziałem poszczególnych robotów do zadań zajmuje się agent DMA. Odpowiedź na zapytanie o dostępność robotów realizujących określoną usługę zawiera informacje o okresach czasu, w których robot jest dostępny. Komunikat żądania rezerwacji robota musi natomiast zawierać informacje na temat czasu rozpoczęcia i zakończenia zadania. Robot zostaje przydzielony tylko wtedy, gdy termin nie koliduje z istniejącymi już rezerwacjami lub operacjami serwisowymi. Schemat komunikacji w systemie podczas wybierania robota do zadania prezentuje rysunek 7.1.



Rysunek 7.1: Komunikacja pomiędzy agentami w podsystemie zapewnienia dostępności robotów.

W przypadku bardzo dużych systemów, w których występowałyby setki agentów DMA może dochodzić do nadmiernego obciążenia warstwy komunikacyjnej przez protokół zapytań o dostępność usług. Jest on oparty o rozgłaszanie komunikatów przez agenty odpowiedzialne za realizację zadań – odpowiedzi na zapytania wysyłane są przez wszystkie

agenty DMA, co może powodować znaczne obciążenia. Rozwiązaniem może być zastosowanie opisywanego wcześniej wzorca hierarchizacji. Wymaga to wprowadzenia nadrzędnego agenta lub agentów typu DMA, które przejmą obsługę protokołu zapytań od kilku agentów podrzędnych.

Opisane tu podejście do problemu zapewniania dostępności systemu i przydziału robotów do zadań charakteryzuje się pożądanymi cechami pozafunkcjonalnymi. Przede wszystkim praktycznie nie występują tu ograniczenia skali. Dodawanie nowych robotów może być realizowane bez zatrzymywania systemu, poprzez zarejestrowanie ich w istniejących zajezdniach. W przypadku przekroczenia możliwości pojedynczego agenta DMA, część robotów można przenieść do innej (nowej) zajezdni pod opiekę innego DMA. Jeżeli pojedynczy agent DMA jest w stanie dbać średnio o m robotów, a w systemie obecnych jest n robotów, to liczba agentów powinna wynosić $\lceil n * (1 + \frac{1}{m}) \rceil$ agentów, a na pewno będzie mieścić się w przedziale $[n + 1, 2n]$.

Jedyna komunikacja, jaka występuje samoczynnie wewnątrz systemu jest związana z potrzebami serwisowymi robotów. Liczba komunikatów jest zależna liniowo od liczby robotów i częstotliwości wykonywania czynności serwisowych. Komunikacja z pozostałymi podsystemami to głównie protokoły zapytań i rezerwacji robotów. Jeżeli w systemie działa d agentów DMA, to dla pojedynczego zadania wymagającego zarezerwowania k robotów przesłanych może zostać maksymalnie $1 + d + 2k$ komunikatów. Wyjątkiem może być sytuacja odmowy przydzielenia żądanego robota. Gdy d staje się zbyt duże zastosować można wspomnianą wcześniej hierarchię agentów DMA.

Rozszerzalność możliwości systemu o nowe typy robotów również jest zagwarantowana. Jeżeli istnieje konieczność wprowadzenia robota, którym nie jest w stanie opiekować się żaden istniejący DMA, konieczne jest dodanie nowej zajezdni i nowego DMA. Najczęściej jest to spowodowane również nowymi wymaganiami sprzętowymi – istniejąca infrastruktura serwisowa nie jest odpowiednia dla nowego typu robota. Architektura umożliwia również podmianę agenta DMA nową wersją, jednak wymaga to wyrejestrowania wszystkich robotów, co czasowo może zmniejszyć wydajność systemu.

Ukrywanie funkcjonalności robotów za interfejsem programowym daje podobne korzyści, jak stosowanie abstrakcyjnych interfejsów dla komponentów programowych. Umożliwia zamianę elementu wykonującego konkretną czynność bez konieczności modyfikowania oprogramowania zlecającego wykonanie tej czynności. Pozwala to w szczególności na wykorzystywanie różnego typu robotów do realizacji tych samych zadań, co umożliwia ograniczenie ogólnej liczby robotów w systemie, zapewnia większą niezawodność oraz wydajność.

7.2 Przydział i wykonanie zadań

Podsystem przydziału i wykonania zadań realizuje podstawową funkcjonalność systemu – przyjmuje zlecone zadania, wyszukuje sposoby ich wykonania, ustala kolejność realizacji, wybiera roboty i wykonuje przy ich pomocy niezbędne czynności. W większości rzeczywistych systemów będzie najczęściej zmienianym i rozbudowywanym elementem, dlatego jego architektura musi zapewnić wszystkie pozafunkcjonalne właściwości systemu informatycznego.

Agent realizujący pojedyncze zadanie jest nazywany *Task Executor Master Agent* (TExMA) i jest implementacją sposobu wykorzystania usług oferowanych przez roboty. Każdy rodzaj TExMA jest w stanie wykonać jeden rodzaj zadania, wykorzystując określoną liczbę robotów dostarczających określone usługi. Agent TExMA jest tworzony dla każdego zadania z osobna, a kończy życie w chwili zakończenia wykonywania zadania.

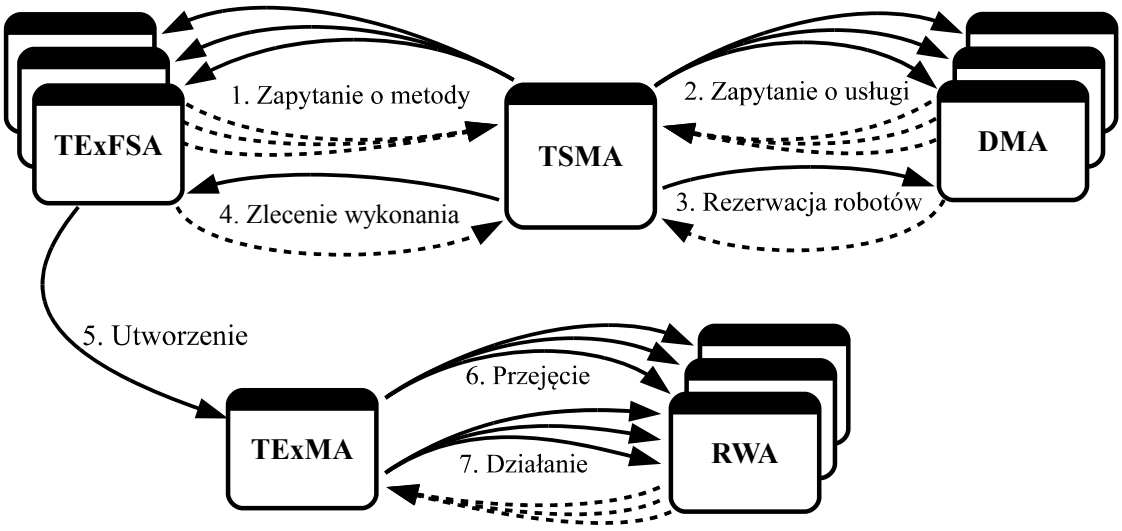
Instancje agenta TExMA są tworzone przez agenta typu *Task Executor Factory Server Agent* (TExFSA). Dla każdego typu agenta TExMA istnieje w systemie tylko jeden agent TExFSA. Poza samym tworzeniem instancji agentów wykonujących zadania, TExFSA pełni również rolę informatora o możliwościach tworzonych agentów. Protokół komunikacji z agentem TExFSA zawiera następujące rodzaje wiadomości:

- zapytanie o możliwość wykonania zadania,
- zlecenie wykonania zadania.

Polecenie wykonania zadania powoduje powołanie do życia nowej instancji agenta TExMA. Jako parametr przyjmuje ono identyfikatory agentów RWA, czyli listę robotów, które będą wykonywały zadanie. TExMA mógłby samodzielnie wybrać roboty komunikując się z agentami DMA, jednak wymagałoby to zawarcia w implementacji każdego TExMA obsługi odpowiedniego protokołu komunikacji. Lepszym rozwiązaniem jest przypisanie robotów do zadań przed utworzeniem agentów TExMA, co pozwala dodatkowo na optymalizację kolejności wykonywania zadań. Przy takim podejściu agent TExMA ma tylko jeden cel: jak najszybciej zakończyć wykonywanie zadania przy pomocy przypisanych do wykonania danego zadania robotów.

Ostatni element w podsystemie przydziału i wykonania zadań to planista, zwany *Task Scheduler Master Agent* (TSMA). Udostępnia on zewnętrzny interfejs dodawania zadań dla systemu oraz zarządza kolejnością wykonania zadań i przydziałem robotów. W systemie może pracować wiele agentów TSMA realizujących różne grupy zadań. Schemat komunikacji pomiędzy agentami w podsystemie przydziału i wykonywania zadań przedstawia rysunek 7.2.

Najważniejszą cechą agenta TSMA jest jego niezależność od typu wykonywanego zadania. TSMA nie analizuje znaczenia zadania, nie zajmuje się sposobem jego realizacji. Nie



Rysunek 7.2: Komunikacja pomiędzy agentami podczas planowania i wykonywania zadań (kolejność komunikatów zaznaczono numerami).

jest więc ograniczony jedynie do pewnego zbioru zadań, które przewiduje jego implementacja. Specyfikacja każdego zadania umożliwia stwierdzenie, które typy agentów TExMA są w stanie to zadanie realizować. Wymagania poszczególnych typów TExMA dotyczą jedynie liczby robotów i dostarczanych przez nie usług. Agent TSMA jest więc implementacją algorytmu aranżacji usług dostarczanych przez agenty RWA i TExMA. Może dodatkowo być wzbogacony o mechanizmy optymalizacji wykorzystania dostępnych zasobów sprzętowych.

Przedstawiona struktura systemu agentowego pozwala na osiągnięcie pożądanych właściwości pozafunkcjonalnych. Można zauważyć, że liczba instancji agentów wykonujących zadania jest równa liczbie zadań wykonywanych w danej chwili czasowej. Zadania, które oczekują na realizację są przechowywane w pamięci agenta planującego przydział robotów do zadań – dopiero w chwili rozpoczęcia wykonywania zadania tworzony jest nowy agent. Liczba komunikatów potrzebnych do rozpoczęcia wykonywania zadania jest uzależniona od liczby różnych metod rozwiązania danego zadania, ponieważ tyle agentów TExFSA odpowie na rozgłoszone zapytanie. Można więc stwierdzić, że zarówno liczba agentów obecnych w systemie jak i obciążenie warstwy komunikacyjnej podczas etapu aranżacji usług nie stanowią ograniczenia skalowalności systemu. Samo wykonanie zadania będzie natomiast wymagało dość częstej komunikacji pomiędzy agentami RWA i agentem TExMA, która jest niezbędna w celu synchronizowania działania robotów.

Podejście agentowe gwarantuje modularność rozwiązania, jednak dopiero odpowiedni podział funkcjonalności pomiędzy poszczególne agenty pozwana na osiągnięcie korzyści

wynikających z modularności. W opisywanym systemie implementacja każdego elementu może zostać podmieniona nową wersją bez konieczności zatrzymywania pozostałych elementów, co w sytuacji współbieżnego wykonywania długotrwałych zadań ma bardzo istotne znaczenie. Jeżeli stwierdzona zostanie usterka programowa jednego z algorytmów realizujących zadania, możliwe jest usunięcie odpowiedniego agenta TExFSA i wprowadzenie nowej wersji, która będzie tworzyła instancje TExMA nowego, poprawionego typu. Istniejące instancje mogą natomiast bez przeszkód dokończyć wykonanie rozpoczętych zadań.

Dodawanie nowych typów zadań wykonywanych przez roboty będzie najprawdopodobniej najczęściej wykonywaną rozbudową funkcjonalności systemu. W prezentowanej architekturze polega ona na dodaniu jednego agenta TExFSA, który jest w stanie tworzyć instancje agentów TExMA implementujących rozwiązanie nowego zadania. Nowe agenty mogą z powodzeniem wykorzystywać istniejące roboty. W przypadku pojawienia się w systemie robotów nowego typu, mogą one zostać wykorzystane przez istniejące wcześniej algorytmy wypełniania zadań. Taka elastyczność pozwala na osiągnięcie wysokiego stopnia wykorzystania sprzętu i zapewnia odporność na zmiany w ilościach zadań poszczególnych typów.

7.3 Planowanie tras i koordynacja ruchu

Jeżeli system wykorzystuje roboty mobilne, to pomiędzy etapem przypisywania robotów do zadań, a rozpoczęciem ich wykonania, musi nastąpić przemieszczenie robotów do miejsca, w którym zadanie ma być wykonane. Zagadnienie przemieszczania robotów mobilnych jest jednym z bardziej złożonych i skomplikowanych problemów, z jakimi poradzić musi sobie system wielorobotowy. Zakładając, że środowisko działania jest znane oraz istnieje metoda lokalizacji robotów w środowisku, do rozwiązania pozostają następujące kwestie:

- sterowanie napędem danego robota w celu pokonania zaplanowanej trajektorii lub osiągnięcia zadanego celu,
- planowanie trasy dla danego robota,
- koordynacja ruchu grupy robotów.

Sposób sterowania napędem jest uzależniony od rodzaju robota. Algorytm sterujący, specyficzny dla każdego typu urządzenia, powinien zostać ukryty za interfejsem usługi agenta RWA – usługa mobilności jest podstawową usługą dostarczaną przez każdego RWA robota mobilnego.

Zagadnienie zaplanowania tras i jednoczesnego przemieszczenia grupy robotów z konfiguracji początkowej do docelowej można postrzegać jako zadanie. Możliwe byłoby stworzenie agenta TExMA, który implementowałby algorytmy planowania trasy koordynacji

ruchu. Musiałby on posiadać kompletną wiedzę o strukturze środowiska (mapie), oraz o planach innych agentów zajmujących się przemieszczaniem robotów. W przypadku rozległych środowisk i licznych grup robotów, takie podejście może znacząco ograniczyć możliwości systemu.

Problem przemieszczania grup robotów występował będzie w każdym systemie wykorzystującym roboty mobilne. Warto więc zastanowić się nad uniwersalnymi algorytmami zapewniającymi niezależność od typów robotów, skalowalność oraz wysoką wydajność, które będzie można wykorzystywać w różnych systemach. W osiągnięciu tych właściwości pomocna będzie opisywana wcześniej wielomodelowa reprezentacja przestrzeni działania robotów.

W rozdziale 4.3.1 zostały wspomniane dwie podstawowe klasy metod sterowania przemieszczaniem robota mobilnego, do których da się zakwalifikować zdecydowaną większość istniejących algorytmów:

- reaktywne metody unikania kolizji,
- metody planowania i wykonania (przebycia) trasy.

Metody reaktywne, podczas trwania nawigacji, cyklicznie obliczają sterowanie, które przybliży robota do ustalonego celu, jednocześnie nie powodując kolizji z przeszkodami. Są one najczęściej bezstanowe, czyli nie zapamiętują poprzednich konfiguracji robota i nie przewidują przyszłych. Nie wymagają wiedzy na temat kształtu całego środowiska, opierają się najczęściej na bieżących odczytach z sensorów. Ich implementacja jest nieskomplikowana, a ilość obliczeń prowadzonych podczas nawigacji jest niewielka, dlatego mogą być z powodzeniem wykonywane przez autonomiczne roboty, które nie posiadają wydajnych jednostek obliczeniowych. Istnieje wiele prac dotyczących tej grupy metod [60, 80]; większość wskazuje podobne zalety oraz wspólne ograniczenia, z których najpoważniejsze to brak niezawodności. Metody nawigacji reaktywnej nie gwarantują osiągnięcia celu, nie umożliwiają stwierdzenia, czy osiągnięcie celu jest możliwe oraz nie pozwalają na dokładne oszacowanie czasu osiągnięcia celu. Działają jednak bardzo dobrze w środowiskach, których kształt można modelować figurą wypukłą.

Druga grupa algorytmów umożliwiających przemieszczenie robota do zadanej lokalizacji, to metody planujące trasę na podstawie mapy, czyli modelu środowiska działania robota [24]. Odpowiednia reprezentacja mapy (najczęściej graf lub siatka zajętości) jest wykorzystywana do znalezienia optymalnej ścieżki. Najczęściej stosowanym kryterium optymalności trasy jest jej długość lub przewidywany czas pokonania, ale może być nim także bezpieczeństwo czy minimalne zużycie energii. Globalne metody planowania, czyli takie, które wykorzystują mapę całego środowiska, pozwalają na stwierdzenie, czy cel jest osiągalny,

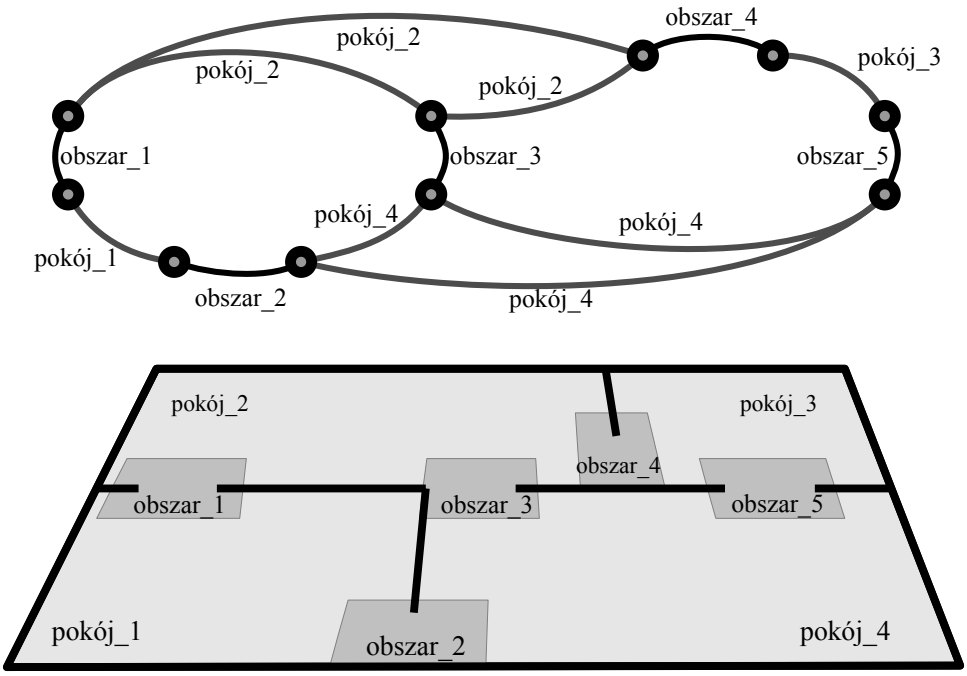
oraz jaki jest przewidywany czas osiągnięcia celu. Należy jednak zaznaczyć, że ich wymagania znacznie przewyższają wymagania metod reaktywnych. Przede wszystkim konieczna jest szczegółowa wiedza na temat środowiska, a ponadto ilość wykonywanych obliczeń jest uzależniona od wielkości środowiska. Taki brak skalowalności uniemożliwia bezpośrednie stosowanie metod globalnego planowania w systemach, które mają spełniać podstawowe wymagania pozafunkcjonalne.

Podobne ograniczenia dotyczą algorytmów koordynacji ruchu. Problem wyznaczania i optymalizacji tras bezkolizyjnych dla wielu robotów jest znacznie bardziej złożony od problemu planowania trasy w środowisku statycznym. Istniejące podejścia i propozycje rozwiązań [16] charakteryzują się bardzo wysoką złożonością i mogą być stosowane w niewielkich środowiskach i dla nielicznych grup robotów.

Rozwiązaniem problemu skalowalności algorytmów planowania tras i koordynacji ruchu może być opisywane wcześniej podejście, oparte na jednoczesnym zastosowaniu wielu modeli środowiska. Warto zauważyć, że planowanie trajektorii i koordynowanie ruchu wielu robotów najczęściej nie musi być stosowane w całym środowisku ich działania. Możliwe jest wyznaczenie w środowisku fragmentów (nazywanych dalej pokojami), w których roboty mogą bezpiecznie i skutecznie przemieszczać się stosując autonomiczne algorytmy reaktywne. Precyzyjne planowanie i koordynacja jest konieczna jedynie w niektórych częściach środowiska (zwanych dalej obszarami), np. w wąskich przejściach czy skrzyżowaniach tras. Dodatkowo, zastosowanie takiej separacji obszarów prowadzi do niezależności procesów zarządzania ruchem w poszczególnych obszarach – mogą one być z powodzeniem realizowane przez osobne agenty programowe, zwane *Area Master Agent* (AMA). Poszczególne agenty AMA mogą implementować różne algorytmy operujące na różnego typu szczegółowych modelach fragmentów środowiska, co daje dużą swobodę rozbudowywania środowiska. Przykłady agentów implementujących różne algorytmy koordynacji ruchu zostaną przedstawione i porównane w następnych rozdziałach.

Opisany podział środowiska tworzy w sposób naturalny abstrakcyjną mapę. Jeżeli wszystkie wydzielone fragmenty środowiska (zarówno pokoje, jak i obszary) przedstawimy jako krawędzie grafu, a połączenia między nimi jako węzły, to otrzymamy grafowy model środowiska. Może on z powodzeniem być wykorzystywany do globalnego planowania tras. Krawędziom reprezentującym pokoje przypisać można wagi proporcjonalne do odległości pomiędzy łączonymi obszarami. Przykład środowiska i odpowiadającej mu mapy grafowej ilustruje rysunek 7.3.

Graf jest przechowywany i wykorzystywany przez agenta *Map Server Agent* (MSA), który implementuje algorytm wyszukiwania najkrótszej ścieżki. Każdy agent, który chce zaplanować trasę dla robota, kontaktuje się z MSA, podając współrzędne punktu początkowego i docelowego. MSA wyszukuje trasę i zwraca listę pokoi i obszarów, które muszą



Rysunek 7.3: Schemat przykładowego środowiska (na dole) oraz graf będący jego abstrakcyjnym modelem. Pokoje to fragmenty terenu, w których roboty mogą przemieszczać się autonomicznie, obszary wymagają stosowania algorytmów koordynacji ruchu.

zostać pokonane. MSA może dodatkowo implementować algorytm równoważenia ruchu, oparty na preferowaniu tras omijających najbardziej obciążone regiony środowiska. Szczegóły działania tego algorytmu zostaną opisane w następnym rozdziale.

Dla niewielkich środowisk, w których graf reprezentujący całe środowisko może być przetwarzany centralnie, tak skonstruowana mapa wielomodelowa jest wystarczająca. Każda trasa jest planowana przez jedynego MSA. Jeżeli jednak środowisko znacząco się powiększy, wykorzystanie scentralizowanego algorytmu wyszukiwania ścieżki może okazać się niedostatecznie wydajne. Należy pamiętać, że złożoność algorytmów wyszukiwania najkrótszej ścieżki w grafie nie jest liniowa.

Najbardziej klasyczny algorytm wyszukiwania najkrótszej ścieżki w grafie – algorytm Dijkstry [27] – charakteryzuje się kwadratową złożonością; dla grafu o V wierzchołkach i E krawędziach wynosi ona $O(|V|^2+|E|)$. Znacznie bardziej wydajne jest zastosowanie metod modyfikujących kolejność odwiedzania węzłów grafu na podstawie oszacowania odległości węzła od celu. Takie oszacowanie, zwane heurystyką, jest łatwe do uzyskania w grafach reprezentujących rzeczywistą przestrzeń, ponieważ każdy węzeł jest skojarzony z konkretną lokalizacją. Pierwszym algorytmem stosującym heurystykę był algorytm A* [25], który w różnych wariantach jest cały czas wykorzystywany. Przy zastosowaniu poprawnej heu-

rystyki, złożoność algorytmu A* nie jest zależna od ilości węzłów i krawędzi w grafie, a jedynie od długości najkrótszej ścieżki.

Zastosowanie oszacowania, obliczanego jako odległość euklidesowa pomiędzy lokalizacją węzła i punktu docelowego, powoduje niestety znaczne problemy w sytuacji gdy środowisko jest bardzo złożone i zawiera liczne i wyraźne lokalne minima odległości pomiędzy punktami. Taka sytuacja jest dość często obecna w rzeczywistości – przykładem może być problem wyznaczenie trasy pomiędzy dwoma pomieszczeniami znajdującymi sie nad sobą, na różnych piętrach budynku. Optymalna trasa wymaga wykorzystania schodów lub windy, która może znajdować się w odległości wielokrotnie większej niż euklidesowa odległość pomiędzy źródłem i celem.

By zilustrować problem wydajności wyznaczania trasy w złożonych środowiskach przeprowadzono pomiary czasu wyszukiwania najkrótszej ścieżki w grafach o różnym stopniu skomplikowania. Wyniki tych badań zostaną zaprezentowane w rozdziale 9.2.

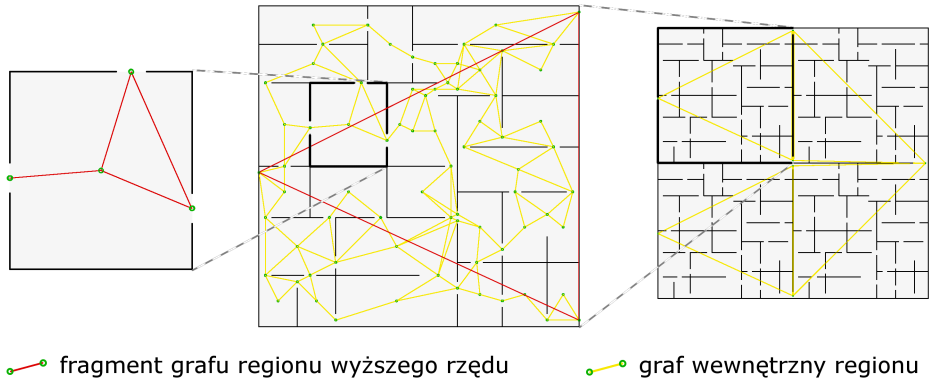
Jednym z możliwych rozwiązań tego problemu może być uproszczenie modelu środowiska i zastosowanie mapy na wyższym poziomie abstrakcji, która reprezentowana będzie przez graf o mniejszej ilości wierzchołków i krawędzi. Pociąga to jednak za sobą poważne konsekwencje, które nie zawsze są akceptowalne:

- Utrata informacji o charakterystyce trasy – odpowiednio szczegółowy graf może być wykorzystany do zapisywania dodatkowych informacji wykorzystywanych podczas planowania, jak szerokość przejść czy charakterystyka łuków. W zależności od typu robota niektóre krawędzie mogą być nieakceptowalne.

- Trudności z reprezentowaniem dynamiki środowiska – zmianę w środowisku trudno odwzorować w modelu, gdy nie istnieją krawędzie odpowiadające każdemu możliwemu przejściu.

Rozwiązaniem, które pozwala na szczegółową reprezentację środowiska oraz jest dostatecznie wydajne, może być wykorzystanie mapy wielomodelowej o wielu poziomach abstrakcji. By opisać algorytm tworzenia i wykorzystywania mapy wielomodelowej o dowolnej liczbie poziomów, konieczne jest zdefiniowanie podstawowej jednostki, z której mapy są konstruowane, nazywanej **regionem**. Region jest modelem fragmentu środowiska, którego szczegółowość zależy od poziomu regionu w hierarchii. Regiony wyższych poziomów modelują większe fragmenty terenu w sposób bardziej uproszczony. Regiony najniższego poziomu to pokoje i obszary; regiony wyższych poziomów modelują fragment środowiska za pomocą grafu. Tworzenie hierarchii jest realizowane poprzez łączenie regionów niższych poziomów w jeden region poziomu wyższego. Każdy podregion tworzy fragment grafu regionu poziomu wyższego poprzez utworzenie wierzchołków w miejscach modelujących połączenia z innymi podregionami, oraz utworzenie krawędzi łączących te wierzchołki. Wagi utworzonych

krawędzi są proporcjonalne do odległości wierzchołków, która w przypadku wyższych poziomów jest rozumiana jako długość najkrótszej ścieżki pomiędzy łączonymi wierzchołkami w grafie podregionu. Utworzony model poziomu wyższego reprezentuje więc możliwość i koszt przejechania robota przez poszczególne podregiony, bez wnikania w szczegóły trasy. Przykład kolejnych poziomów mapy wielomodelowej jest przedstawiony na rysunku 7.4.



Rysunek 7.4: Kolejne poziomy abstrakcji modelu tworzące mapę wielomodelową.

W przypadku stosowania wielu poziomów mapy, algorytm wyszukiwania trasy dla robota jest implementowany i wykonywany przez wiele agentów MSA. Z punktu widzenia agenta żądającego wyznaczenia trasy nie ma znaczenia, czy zadanie realizuje jeden MSA, czy wiele. System agentów MSA połączony jest w hierarchię odpowiadającą hierarchii regionów, przy czym regiony najniższego poziomu (pokoje i obszary) nie posiadają nadzorujących MSA.

Algorytm wyznaczania trasy w n-poziomowej mapie wielomodelowej rozpoczyna się od zlokalizowania najmniejszego regionu, który zawiera zarówno źródło jak i cel poszukiwanej trasy. W tym celu agent MSA najniższego poziomu, do którego przysłane zostało zapytanie o wyznaczenie trasy, sprawdza czy nadzorowany przez niego obszar zawiera punkt docelowy. Jeśli nie, żądanie wyznaczenia trasy jest przekazywane do agenta MSA wyższego poziomu. Ta operacja jest powtarzana do czasu odnalezienia MSA nadzorującego obszar zawierający zarówno początek jak i koniec planowanej trasy (nazywany dalej MSA_p).

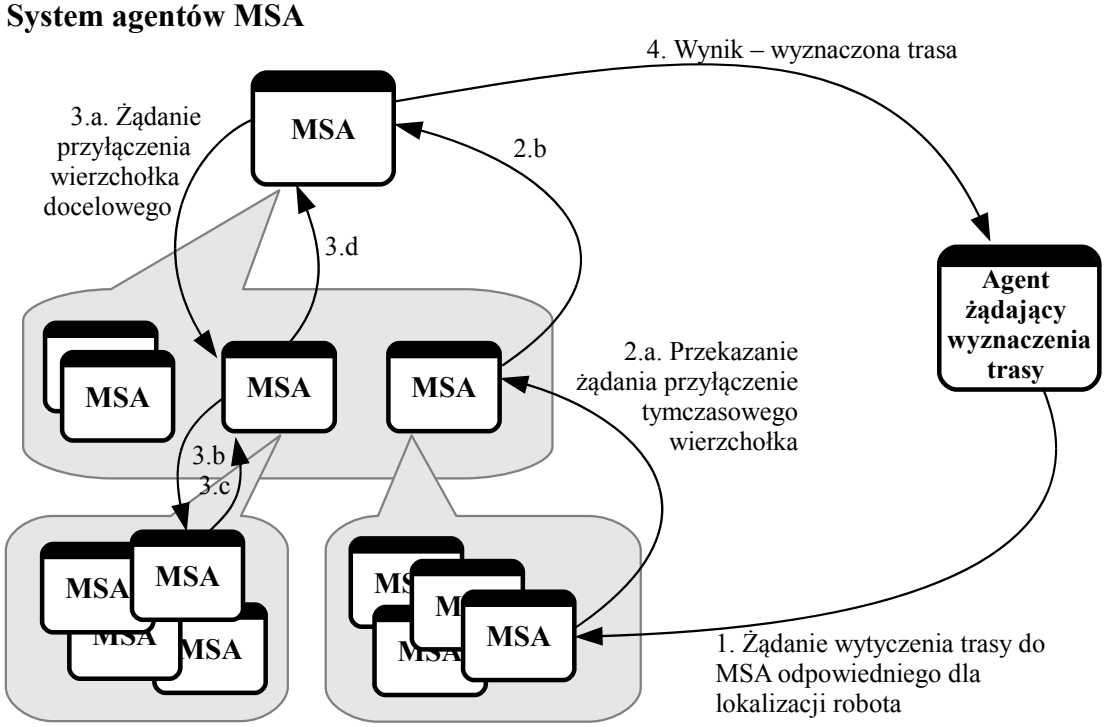
Podczas przekazywania zapytania do kolejnych MSA tworzone są w ich grafach tymczasowe wierzchołki, które reprezentują początkowy punkt trasy. Są one przyłączane do punktów styku grafu podregionu tymczasowymi krawędziami, których wagi są wyliczane przez przekazującego zapytanie MSA.

Gdy odnaleziony zostanie MSA_p, który nadzoruje obszar zawierający zarówno początek jak i koniec trasy, musi on dodatkowo przyłączyć docelowy punkt docelowy trasy do swojego grafu. Wymaga to wykonania procedury odwrotnej do przyłączania punktu po-

czątkowego. Żądanie przyłączenia wierzchołka docelowego do grafu jest przekazywane do MSA coraz niższych poziomów, aż do MSA najniższego poziomu. Przyłączenie tymczasowych wierzchołków wymaga obliczenia maksymalnie $2 * n * m$ najkrótszych ścieżek w grafach regionów na różnych poziomach, gdzie m to średnia ilość punktów styku regionów.

Po zakończeniu tej procedury graf agenta MSA_p zawiera dwa tymczasowe wierzchołki, które reprezentują początek i koniec planowanej trasy. MSA_p uruchamia algorytm A* lub Dijkstry w celu wyznaczenia ścieżki o najniższym koszcie pomiędzy tymi wierzchołkami. Będzie ona reprezentowała optymalną trasę dla robota.

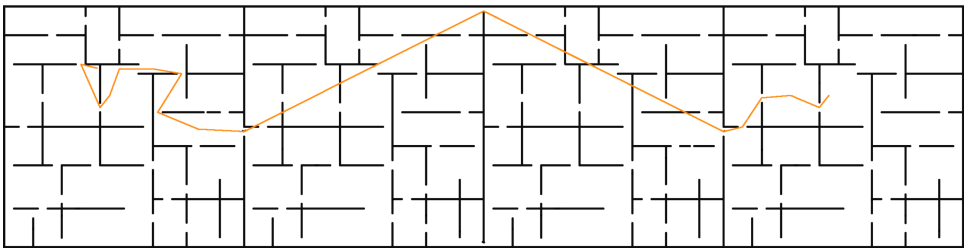
Schemat komunikacji pomiędzy agentami podczas planowania trasy prezentuje rysunek7.5. Można zauważyć, że wyznaczenie trasy może wymagać przesłania maksymalnie $2 + 3 * n$ komunikatów, przy czym warto zauważyć, że liczba komunikatów jest zależna od długości ścieżki. Jeżeli początek i koniec trasy znajdują się w obszarze nadzorowanym przez tego samego MSA, przesłane będą tylko 2 komunikaty.



Rysunek 7.5: Komunikacja w podsystemie nawigacji podczas planowania trasy dla robota.

Zaplanowana trasa jest reprezentowana jako ciąg elementów składających się z par wierzchołków skojarzonych z regionem przez który przechodzą. Jeżeli region jest regionem najniższego poziomu, to para wierzchołków wyznacza odcinek wewnątrz któregoś z pokoi. Jeżeli region jest regionem wyższego rzędu, to trasę szczegółową pomiędzy wierzchołka-

mi trzeba wyznaczyć z wykorzystaniem grafu regionu. Do najniższego poziomu rozbite są jedynie końcowe odcinki zaplanowanej trasy. Fragmenty przechodzące „tranzytem" przez regiony wyższych poziomów są pozostawiane do czasu zaistnienia konieczności opracowania trasy szczegółowej, czyli do czasu osiągnięcia przez robota początkowego wierzchołka danego elementu. Opóźnione wyznaczanie trasy szczegółowej pozwala na zaplanowanie trasy zgodnie z najbardziej aktualnym stanem środowiska, reprezentowanym przez model. Przykładowa trasa wyznaczona przez opisywany algorytm jest pokazana na rysunku 7.6.



Rysunek 7.6: Przykładowa trasa wyznaczona przy użyciu mapy wielomodelowej. Środkowe fragmenty, które przechodzą „tranzytem" przez regiony, zostaną rozplanowane przy pomocy grafu niższego poziomu, gdy zajdzie taka potrzeba.

Z właściwości algorytmów wyznaczania najkrótszej ścieżki w grafie wynika, że znacznie szybciej można wyznaczyć kilka ścieżek w niewielkim grafie, niż jedną w grafie bardo złożonym. Właśnie ta obserwacje jest wykorzystywana przez opisany tu system agentowy do szybkiego i dokładnego wyznaczania tras dla robotów. Porównanie wydajności opracowanej metody z klasycznymi algorytmami wyznaczania najkrótszej ścieżki w grafie zostanie zaprezentowane w rozdziale 9.2.

Opisane podejście do problemu planowania tras i koordynacji ruchu zapewnia pożądane właściwości pozafunkcjonalne. Zastosowanie wielopoziomowej mapy wielomodelowej powoduje, że czas planowania trasy nie jest zależny kwadratowo od wielkości środowiska, ale liniowo od liczby poziomów mapy. Rozproszenie algorytmu koordynacji ruchu pomiędzy niezależne agenty zarządzające poszczególnymi obszarami pozwala na realizowanie obliczeń na wielu komputerach. Zwiększanie skali systemu nie stanowi zatem problemu.

Rozszerzanie środowiska o nowe typy fragmentów, np. udostępnienie windy łączącej dwa poziomy budynku, również nie jest trudne. Fragment można uznać za obszar zarządzany przez nowy typ agenta AMA, który będzie implementował algorytm obsługi nowego urządzenia. Obszar można przyłączyć do odpowiednich regionów poprzez modyfikację ich grafów. Żadne inne elementy systemu nie muszą być modyfikowane, w szczególności algorytmy sterujące robotami (agenty RWA) nie muszą implementować sposobów obsługi nowego urządzenia.

Oprogramowanie implementujące podsystem jest podzielone na autonomiczne moduły: agenty koordynujące ruch w obszarach oraz komponent mapy wielomodelowej. Zarówno algorytm planowania tras jak i algorytmy agentów koordynujących ruch powinny być niezależne od typów robotów, co umożliwia wykorzystywanie implementacji w wielu systemach.

W dalszych rozdziałach opisana zostanie zrealizowana implementacja systemu agentowego zarządzającego ruchem grupy robotów oraz zaprezentowane będą wyniki badań tego systemu.

8 Implementacja systemu agentowego zarządzającego ruchem robotów mobilnych

Zagadnienia związane z przemieszczaniem są jednymi z najciekawszych problemów, jakie stoją przed współczesną robotyką. Ich skuteczne rozwiązanie jest podstawą każdego systemu wykorzystującego roboty mobilne. Dlatego też zostaną użyte jako przykład problemów, które można skutecznie rozwiązać przy pomocy prezentowanego w niniejszej pracy podejścia.

Przedstawiony w rozdziale 4.3 przegląd problemów z dziedziny nawigacji robotów wskazuje, że istniejące rozwiązania mają charakter eksperymentalny. Skupiają się na rozwiązaniu jednego, szczegółowego aspektu, stosują określony rodzaj sprzętu, działają w wybranym środowisku. Każde z nich ma inne właściwości i ograniczenia, zadziała poprawnie w określonych warunkach, ale żadne nie gwarantuje uniwersalności. By możliwe było stworzenie systemu nawigacji robotów, który mógłby zostać zastosowany w praktyce, konieczne jest zagwarantowanie jego wysokiej jakości. System taki musi być niezawodny, elastyczny, rozszerzalny i skalowalny, musi pokrywać wszystkie aspekty nawigacji i przewidywać możliwe kierunki rozwoju.

Opisywana w tym rozdziale implementacja ma na celu wykazanie, że w oparciu o zaprezentowane wcześniej podejście, można skonstruować system nawigacji robotów, który będzie się charakteryzował pożądanymi cechami pozafunkcjonalnymi.

W pierwszej części rozdziału przedstawiona zostanie wykorzystana w implementacji platforma agentowa oraz sposób realizacji poszczególnych typów agentów w ramach platformy. W kolejnych częściach rozdziału zaprezentowane zostaną algorytmy implementowane przez poszczególne agenty oraz sposób współpracy agentów w ramach systemu:

- Algorytm konstruowanie mapy wielomodelowej na podstawie modelu środowiska.
- Metoda planowania trasy z wykorzystaniem utworzonej mapy.
- Algorytm autonomicznej nawigacji reaktywnej dla pojedynczego robota.
- Podstawowa wersja systemu zarządzającego ruchem grupy robotów.
- Algorytmy koordynacji ruchu stosowane w newralgicznych fragmentach środowiska.
- Rozszerzona wersja systemu zarządzającego ruchem, która charakteryzuje się wszystkimi niezbędnymi właściwościami pozafunkcjonalnymi.

Kwestia lokalizacji robota nie jest rozważana, ponieważ spośród wszystkich problemów nawigacji, jest ona najbardziej powiązana z rodzajem stosowanego sprzętu. W zależności od wyposażenia robota (rodzaje sensorów) oraz dostępnej infrastruktury środowiska

(sztuczne znaczniki, gps), sposób realizacji lokalizacji będzie różny. Jednak niezależnie od wykorzystanych urządzeń, wynikiem działania lokalizacji będzie pozycja i orientacja robota w środowisku, charakteryzowana dodatkowo dokładnością tych wartości. Te informacje są uznawane za dane przez prezentowany system.

8.1 Środowisko agentowe, czyli przestrzeń wirtualna

Środowiskiem działania agentów w prezentowanym podejściu jest przestrzeń wirtualna. Z punktu widzenia implementacji jest to platforma agentowa, która tworzy maszynę wirtualną łączącą wiele komputerów. Stanowi ona środowisko uruchomieniowe dla agentów, zapewniając możliwość ich tworzenia, likwidowania oraz dostarczając podstawowe usługi komunikacyjne.

Platforma agentowa wykorzystywana w prezentowanej implementacji jest oparta na środowisku JADE [13, 14], które jest jednym z najbardziej popularnych rozwiązań tego typu. Pozwala ono na tworzenie maszyny wirtualnej łączącej wiele komputerów poprzez uruchamianie tzw. kontenerów na poszczególnych maszynach i łączenie ich z wykorzystaniem sieci. Tworzy w ten sposób środowisko wirtualne, w którym możliwe jest uruchamianie wielu agentów programowych. Agenty mogą być przemieszczane pomiędzy poszczególnymi kontenerami z zachowaniem stanu.

JADE udostępnia agentom metody komunikacji zgodne ze standardem FIPA-ACL [2], z wykorzystaniem wielu różnych protokołów. Pozwala to na realizację bardzo elastycznej komunikacji asynchronicznej między agentami. Dodatkowo udostępnione są dwa mechanizmy (agenty udostępniające usługi) wspomagające działanie platformy:

- **A**gent **M**anagement **S**ystem, który może tworzyć i likwidować inne agenty, likwidować kontenery i zamknąć platformę.
- **D**irectory **F**aciliator, który jest rodzajem katalogu usług oferowanych przez inne agenty. Usługi mogą być w nim rejestrowane i wyszukiwane.

Pomimo bogatej funkcjonalności, platforma JADE nie jest produktem o wysokiej jakości. Sposób implementacji zachowań (ang. behaviours) oraz warstwy komunikacji powoduje problemy ze stabilnością – zdarzają się wycieki pamięci i sporadyczne problemy z odbieraniem komunikatów. Jednak na potrzeby planowanych eksperymentów jej możliwości są w zupełności wystarczające.

Ponieważ duża ilość komponentów, wykorzystywanych w implementacji prototypowego systemu, jest napisana w języku C++ lub innych językach platformy MS .NET, środowisko JADE zostało zaadaptowane do współpracy z tego typu komponentami. Zostało to osiągnięte poprzez skompilowanie źródeł biblioteki JADE Leap (czyli platformy JADE na

urządzenia przenośne) kompilatorem języka J#. Pozwoliło to na uruchamianie kontenerów JADE na platformie .NET oraz na tworzenie i uruchamianie w tych kontenerach agentów napisanych w językach innych Java. Kontenery JADE .NET mogą z powodzeniem łączyć się z kontenerami uruchamianymi w maszynie wirtualnej Javy, dzięki czemu możliwe jest współistnienie i współpraca w ramach jednej platformy agentów pracujących w maszynach wirtualnych Java oraz .NET. Wspólne protokoły komunikacyjne pozwalają na przesyłanie komunikatów pomiędzy takimi agentami. Nie jest natomiast możliwa migracja agentów pomiędzy kontenerami różnych typów.

Oba typy kontenerów pozwalają na dynamiczne ładowanie komponentów w trakcie działania platformy. Możliwe jest załadowanie nowych bibliotek klas, możliwe jest więc również dodanie implementacji nowych typów agentów. Ta właściwość jest kluczowa z punktu widzenia wymagania rozszerzalności systemu.

Sposób implementacji poszczególnych agentów w platformie JADE zależy od typu agenta. Zdefiniowane w rozdziale 6.4 typy charakteryzują się różnymi właściwościami i możliwościami, których zapewnienie wymaga zastosowania różnych wzorców projektowych i technik wspieranych przez JADE.

Stosunkowo najmniej skomplikowana jest realizacja agentów typu Server Agent. Wymaga ona zdefiniowania jednego zachowania agenta typu *CyclicBehaviour*, które będzie przetwarzało odebrane wiadomości. Po odebraniu wiadomości konieczne jest wykonanie algorytmu specyficznego dla danego agenta i odesłanie wyniku operacji żądającemu agentowi. Podczas inicjalizacji agenta SA, jego usługi muszą zostać zarejestrowane w bazie agenta Directory Faciliator. Z formalnego punktu widzenia, agent Directory Faciliator jest również agentem typu SA.

Implementacja agentów typu Worker Agent jest związana z rodzajem lub rodzajami robotów, którymi agent ma sterować. Agent WA musi definiować jedno zachowanie typu *CyclicBehaviour*, które służy do odbierania wiadomości. Mogą one dodawać cele dla agenta, mogą też być odpowiedziami na wysłane zapytania do agentów SA. Agent WA musi stosunkowo często wykonywać obserwację stanu środowiska robotów, obliczać sterowanie i przekazywać je do urządzeń. W tym celu wykorzystuje najczęściej jedno dodatkowe zachowanie typu *SimpleBehaviour*, które jest uruchomionym wątkiem wykonania algorytmu.

Najbardziej złożona jest implementacja agentów typu Master Agent. Agent korzystający z możliwości innych agentów musi kontaktować się z Directory Faciliator w celu zidentyfikowania zarejestrowanych usług. Będzie więc, podobnie jak wszystkie pozostałe typy, implementował zachowanie typu *CyclicBehaviour*, odpowiedzialne za przetwarzanie odebranych wiadomości. Konieczność nadzorowania wykonania zadań wymaga ciągłego obserwowania stanu środowiska rzeczywistego i wirtualnego. Agent MA jest odpowiedzialny za wykrywanie i obsługę błędów podczas realizacji zadań, musi więc kontrolować sposób re-

alizacji celów przekazanych innym agentom. Często będzie to wymagało stosowania więcej niż jednego wątku przetwarzania.

Środowisko agentowe JADE pozwala na stosunkowo prostą implementację wszystkich zdefiniowanych typów agentów. Podstawowe funkcje, takie jak asynchroniczna komunikacja, wielowątkowe, współbieżne przetwarzanie czy mechanizm publikowania informacji o usługach, są zapewnione przez platformę. Dostarcza ona również możliwości nadzorowania wykonania oraz wykrywania błędów, dzięki rozwiązaniom takim jak agent Sniffer czy Inspector. Szczegóły implementacji badanych algorytmów i uzyskane wyniki badań zostaną zaprezentowane w następnym rozdziale.

8.2 Konstruowanie mapy wielomodelowej

Podstawowym elementem systemu nawigacji jest agent MSA, który konstruuje i przetwarza model środowiska działania robotów. Opisany w rozdziale 7.3 algorytm konstruowania mapy wielomodelowej rozpoczyna swoje działanie od łączenia pokoi i obszarów w regiony wyższego rzędu. Pokoje to regiony rzędu najniższego, w których robot może bezpiecznie przemieszczać się z wykorzystaniem reaktywnej nawigacji autonomicznej. Nawigacja reaktywna sprawdza się doskonale w środowiskach, których kształt można modelować figurą wypukłą, więc taką charakterystykę powinny mieć pokoje. Obszary to fragmenty środowiska, w których stosowany powinien być algorytm koordynacji ruchu.

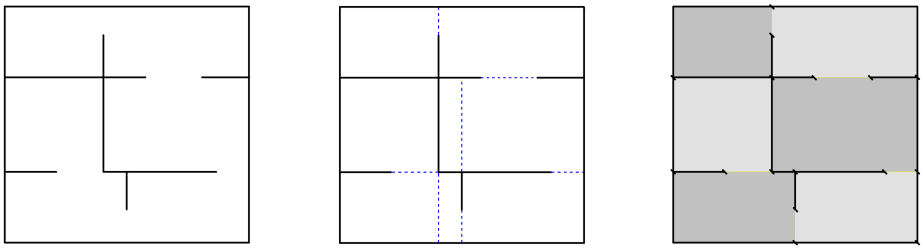
W rozległych środowiskach ręczna realizacja podziału mapy na pokoje nie jest możliwa. Konieczne jest algorytmiczne zrealizowanie tego procesu. Wynikiem działania opisywanego dalej algorytmu jest zbiór pokoi, z których każdy jest definiowany zbiorem odcinków. Odcinki opisują wypukły kształt pokoju, a każdy z nich może być ścianą lub przejściem do innego pokoju. Dane wejściowe algorytmu to zbiór odcinków opisujących wszystkie przeszkody występujące w środowisku robotów.

Na wejściowym zbiorze algorytm wykonuje następujące operacje:

- Utwórz zbiór przedłużeń odcinków opisujących przeszkody. Każdy z wejściowych odcinków jest przedłużany do przecięcia z najbliższym innym odcinkiem. Odcinek przecinany jest dzielony na dwa odcinki w punkcie przecięcia.
- Usuń duplikaty, które mogą powstać w przypadku istnienia odcinków współliniowych, które nie są rozdzielone innym odcinkiem.
- Wyznacz sąsiedztwa wszystkich odcinków, czyli zbiory odcinki zaczynające się lub kończące w końcach każdego z odcinków.
- Zbuduj pokoje. Poczynając od każdego z odcinków dołączaj odcinki sąsiednie, wybierając zawsze odcinek pod najmniejszym kątem do bieżącego. Lista odcinków od-

wiedzonych do czasu powrócenia do odcinka wyjściowego tworzy nowy pokój. Każdy z odcinków może być ścianą pokoju tylko dwa razy.

Przykład działania algorytmu został zilustrowany na rysunku 8.1. Środowisko jest zdefiniowane przez 10 odcinków. Po etapie generowania dodatkowych odcinków zbór zawierał 30 elementów: 9 nowych oraz 21 powstałych na skutek podziału istniejących (sytuacja b.). Po usunięciu duplikatów pozostały 24 elementy, z których zbudowane zostało 6 pokoi (sytuacja c.).



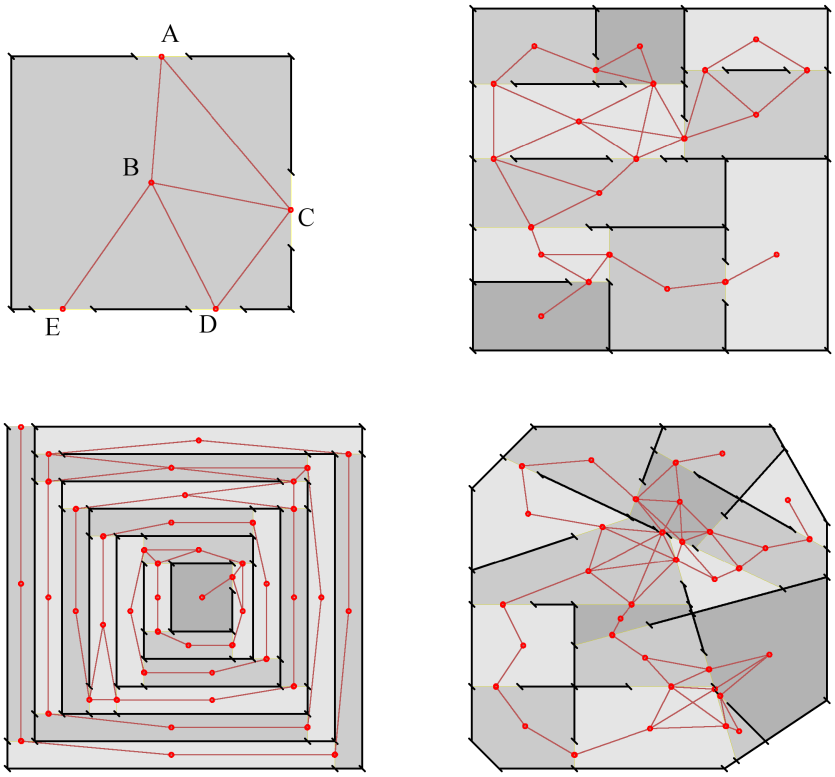
Rysunek 8.1: Kolejne etapy budowania regionów najniższego poziomu na podstawie listy odcinków opisujących przeszkody. Sytuacja **a.** to układ ścian w zadanym środowisku, sytuacja **b.** zawiera potencjalne granice pokojów dodane przez algorytm, sytuacja **c.** to ostateczny podział środowiska.

Każdy z pokoi jest regionem, więc dalsza część algorytmu budowania modelu środowiska przebiega tak samo, jak dla wyższych poziomów modelu. Dla każdego z utworzonych pokoi konstruowany jest graf wewnętrzny regionu, który ma reprezentować możliwe drogi prowadzące przez region. Graf taki ma wierzchołki‘ w punktach, które mogą łączyć region z sąsiednimi regionami. Zrealizowano dwa algorytmy generowania krawędzi łączących n wierzchołków brzegowych:

- poprzez połączenie każdego wierzchołka z każdym – taki graf pełny ma $\frac{n(n-1)}{2}$ krawędzi,
- poprzez utworzenie wierzchołka pośredniego – graf ma wówczas $n+1$ wierzchołków i co najmniej n krawędzi.

Graf utworzony drugim sposobem jest bardziej wydajny podczas wyszukiwania trasy. Podczas tworzenia grafów wewnętrznych pokoi jest stosowana jego modyfikacja, która dodaje dodatkowe krawędzie łączące niektóre pary wierzchołków brzegowych (rys. 8.2 krawędzie AC i CD). Krawędzie nie są dodawane jeżeli wierzchołki znajdują się na współliniowych odcinkach definiujących pokój (rys. 8.2 wierzchołki D i E) lub łącząca je krawędź ma

długość zbliżoną do innej najkrótszej ścieżki między wierzchołkami (rys. 8.2 wierzchołki A i E). Centralny wierzchołek jest umieszczany w środku masy wielokąta definiującego pokój, który dla figur wypukłych zawsze znajduje się wewnątrz figury. Wagi krawędzi są wprost proporcjonalne do euklidesowej odległości wierzchołków, co jest poprawnym oszacowaniem dla regionów najniższego poziomu.



Rysunek 8.2: Przykłady różnych kształtów środowiska i efektów zastosowania algorytmu podziału na pokoje.

Algorytm budujący pokoje ma złożoność kwadratową z względu na liczbę odcinków opisujących przeszkody. Nie można go więc stosować bezpośrednio dla bardzo rozległych i skomplikowanych środowisk. Nie ma natomiast przeciwwskazań do uruchomienia go wielokrotnie dla różnych fragmentów środowiska, czyli dla fragmentów zbioru odcinków opisujących przeszkody, które są zlokalizowane blisko siebie. Wygeneruje on wtedy kilka zbiorów pokoi, które będą miały niepustą część wspólną – wykrycie i usunięcie duplikatów nie stanowi problemu.

Opisane w rozdziale 7.3 podejście przewiduje możliwość tworzenia regionów wyższego rzędu, które modelują większe fragmenty środowiska na wyższym poziomie abstrakcji. Region taki łączy pewną liczbę regionów niższego rzędu, których połączone grafy wewnętrzne

tworzą graf spójny. Analogicznie do opisanej wyżej metody tworzenia grafu wewnętrznego dla pokoju, region wyższego rzędu również konstruuje graf spójny, który łączy wszystkie wierzchołki brzegowe regionu. W tym jednak przypadku nie jest możliwe wykorzystanie wierzchołka pośredniego, z którym połączone byłyby wszystkie wierzchołki brzegowe, ponieważ wyznaczenie wag krawędzi nie byłoby możliwe. Graf wewnętrzny jest więc grafem pełnym, w którym wszystkie pary wierzchołków brzegowych są połączone krawędziami. Wagi utworzonych krawędzi są wyznaczane poprzez obliczenie długości najkrótszej ścieżki pomiędzy łączonymi wierzchołkami w grafie podregionów. Utworzona krawędź jest więc abstrakcyjnym modelem ścieżki w grafie niższego rzędu.

Sposób grupowania podregionów w regiony powinien minimalizować liczbę krawędzi w grafach wyższych poziomów, czyli minimalizować liczbę wierzchołków brzegowych. W typowych środowiskach działania grup robotów, takich jak wielopiętrowe budynki, łatwo wskazać odpowiedni podział – przykładem może być zgrupowanie wszystkich pomieszczeń na jednym piętrze, które jest połączone z innymi poziomami jedną lub dwoma windami. Tworzenie regionów wyższego rzędu jest czynnością wykonywaną jednokrotnie, jedynie w sytuacji bardzo znacznego rozszerzania środowiska działania robotów. Decyzje o optymalnym podziale podejmować może administrator systemu.

Zmiana właściwości środowiska czy też jego rozszerzenie będzie z pewnością dość częstym zjawiskiem w złożonych systemach wielorobotowych. Powodować ją może udostępnienie dla robotów nowego piętra budynku, ale także zamknięcie lub otwarcie drzwi. Hierarchiczna reprezentacja grafowa modelu środowiska jest bardzo dobrze przystosowana do odwzorowywania tego typu zmian. Nie wymagają one zatrzymywania działania modułu planowania trasy, mogą być wprowadzane w biegu.

Zmiany podzielić można na trzy grupy:

- powiększające model – dodanie nowej przestrzeni działania robotów lub nowych połączeń pomiędzy istniejącymi pokojami,
- zmniejszające model permanentnie – usunięcie pokoi lub przejść pomiędzy pokojami,
- zmniejszające model tymczasowo – czasowe usunięcie połączeń pomiędzy pokojami, które może skutkować uniemożliwieniem przemieszczeniem robota do niektórych pokoi.

Pierwsze dwie modyfikacje powodują konieczność zmiany modelu. Trzeci rodzaj modyfikacji powoduje oznaczenie wierzchołków lub krawędzi jako niedostępne. Przywrócenie zablokowanych połączeń jest operacją trywialną.

Rozszerzenie środowiska wymaga utworzenia nowych pokoi oraz ich grafów wewnętrznych. Następnie są one przyłączane do istniejących wcześniej, sąsiadujących pokoi oraz

włączane do hierarchii. Ich utworzenie nigdy nie wymaga usuwania krawędzi z grafu wyższego rzędu – co najwyżej konieczne jest dodanie nowych krawędzi lub zmiana wag krawędzi istniejących. Operacja rozszerzania modelu nie wpływa na zaplanowane wcześniej trasy robotów – są one nadal poprawne, choć mogą przestać być optymalne.

Podczas zawężania środowiska konieczne jest wyznaczenie zmian na wszystkich poziomach abstrakcji. Usunięcie pokoju może mieć wpływ na graf wyższego poziomu, jeżeli krawędź tego grafu była abstrakcją ścieżki zawierającej krawędzie przechodzące przez usuwany pokój. W takiej sytuacji konieczne jest ponowne wyznaczenie kosztu modyfikowanej krawędzi. Jeżeli usuwany pokój zawierał wierzchołek będący jednocześnie wierzchołkiem grafu poziomu wyższego, konieczne jest usunięcie krawędzi z tego grafu. Podczas zawężania środowiska tworzona jest lista wszystkich krawędzi grafów (na wszystkich poziomach abstrakcji), które zostały wyłączone bądź permanentnie usunięte. Następnie wszystkie zaplanowane trasy robotów są przeszukiwane pod kątem zawierania elementów tej listy – jeżeli trasa zawiera usunięte krawędzie, jest w całości planowana od bieżącej pozycji robota.

Tak zrealizowany algorytm konstruowania mapy pozwala agentowi MSA na zbudowanie i przetwarzanie modelu dowolnie skomplikowanych środowisk dwuwymiarowych. W dalszej części rozdziału opisana zostanie implementacja algorytmu planowania trasy z wykorzystaniem zbudowanej mapy oraz inne zastosowania modelu.

8.3 Planowanie i wykonanie tras robotów

Algorytm planowania trasy w mapie wielomodelowej został opisany szczegółowo w rozdziale 7.3. Zrealizowana w testowym systemie implementacja jest realizacją przedstawionej tam metody.

Do wyznaczania najkrótszej ścieżki w grafach wewnętrznych regionów wykorzystywany jest jeden z dwóch algorytmów:

- algorytm Dijkstry, zmodyfikowany tak, by kończył działanie po znalezieniu najkrótsze ścieżki do zadanego wierzchołka,
- algorytm A*, szacujący jakość wierzchołka jako jego odległość Euklidesową od celu.

Poza wyszukiwaniem trasy robota w mapie wielomodelowej, zrealizowana implementacja posiada dwie dodatkowe funkcje. Pierwsza z nich to możliwość opisania każdej krawędzi grafu ograniczeniami wielkości robota. Trasa planowana dla robota charakteryzowanego szerokością s nigdy nie będzie zawierała krawędzi o dopuszczalnej szerokości $e_s < s$. Drugą dodatkową funkcją grafów wykorzystywanych do planowania trasy jest równoważenie obciążenia. Szczegóły zostaną opisane w rozdziale 8.6.

Wynikiem działania algorytmu planowania trasy jest lista kolejnych wierzchołków, które robot powinien osiągnąć. Mogą one należeć do grafów na różnych poziomach abstrakcji. Gdy następna zaplanowana krawędź jest krawędzią grafu jednego z wyższych poziomów, jest ona rozbijana na ścieżkę grafu poziomu niższego. Takie podejście gwarantuje, że następny punkt, który ma zostać osiągnięty przez robota, znajduje się zawsze w tym samym pokoju, co robot. Dzięki temu robot jest w stanie pokonać dowolnie skomplikowaną trasę wykorzystując jedynie algorytm autonomicznej nawigacji reaktywnej.

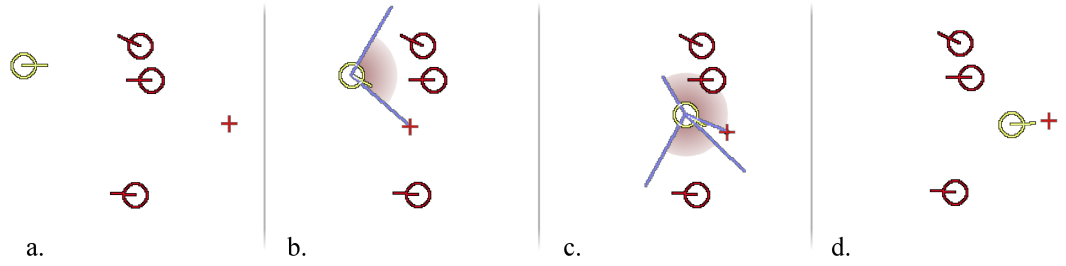
Podczas działania systemu, dowolne fragmenty środowiska robotów mogą zostać przekształcone w obszary, czyli fragmenty terenu, w których roboty są sterowane przez zewnętrzny algorytm implementowany przez agenta AMA. W chwili gdy robot wjedzie w obszar, sterowanie jego ruchami zostaje przejęte przez AMA. W rozdziale 8.7 opisano i porównano możliwości i właściwości różnych zrealizowanych algorytmów koordynacji ruchu.

8.4 Algorytm autonomicznej nawigacji reaktywnej

Podczas pokonywania trasy każdy robot (konkretnie agent RWA) domyślnie wykorzystuje algorytm autonomicznej nawigacji reaktywnej. Algorytm ten jest bezstanowy. W każdym cyklu, na podstawie informacji o otaczających robota przeszkodach wyliczane jest sterowanie, jakie powinno zostać podane na koła, by robot zbliżał się do zadanej pozycji docelowej, unikając jednocześnie kolizji. Wyznaczanie sterowania jest realizowane na podstawie wartości kąta skierowanego α pomiędzy wektorem orientacji robota a wektorem łączącym bieżącą lokalizację robota z punktem docelowym. Im większa jest wartość bezwzględna kąta α, tym mniejszy powinien być promień łuku, po którym robot będzie się przemieszczał. Dla $\alpha = 0$ promień powinien zmierzać do nieskończoności, czyli robot powinien przemieszczać się po prostej.

Zastosowana metoda omijania przeszkód bazuje na wyznaczaniu kątów zakazanych dla robota. Dla każdej przeszkody znajdującej się w odległości mniejszej od ustalonego zasięgu wyznaczany jest zakres kątów, w kierunku których robot nie może się przemieszczać. Jeżeli kierunek prowadzący do celu jest zawarty w zakresie zakazanym, robot czasowo zmienia cel ruchu na punkt pośredni. Punkt ten jest umieszczony na granicy kąta zakazanego w odległości równej odległości od omijanej przeszkody. Spośród dwóch granic kąta zakazanego wybierana jest ta, dla której kąt pomiędzy nią, a wektorem łączącym lokalizację robota z punktem docelowym, jest mniejszy. Zasadę działania algorytmu ilustruje rysunek 8.3, na którym pokazane jest zachowanie robota w okolicy przeszkód.

Robot po lewej stronie pierwszej sytuacji ma osiągnąć pozycję oznaczoną znakiem '+', omijając pozostałe trzy roboty. W chwili wykrycia przeszkody w zasięgu (sytuacja b.), algorytm wyznacza kąt zakazany oraz cel tymczasowy na jednej z granic tego kąta. Wykrycie następnej przeszkody (sytuacja c.) nie wpływa na sterowanie, ponieważ nowy kąt zakazany nie zawiera kierunku wektora łączącego lokalizację robota z bieżącym punktem



Rysunek 8.3: Przykład działania algorytmu autonomicznej nawigacji reaktywnej. Robot z lewej strony sytuacji **a.** ma dojechać do punktu oznaczonego symbolem '+' omijając pozostałe roboty. W sytuacjach **b.** i **c.** algorytm modyfikuje jego cel, by ominąć przeszkody. W sytuacji **d.** robot minął przeszkody i podąża do pierwotnego celu.

docelowym. Po minięciu przeszkód przywracany jest pierwotny cel robota (sytuacja d.).

Dodatkowy mechanizm, stosowane podczas wyliczania sterowania, umożliwia osiąganie celów znajdujących się bardzo blisko przeszkód. Przeszkody znajdujące się w większej odległości od robota niż punkt docelowy są ignorowane.

Prezentowany algorytm jest bardzo skuteczny w nieskomplikowanych środowiskach, których kształt może być modelowany figurą wypukłą, a liczba innych robotów jest niewielka. Jak wszystkie algorytmy reaktywne, może jednak całkowicie zawieść w przypadku umieszczenia robota w lokalnym minimum odległości od celu.

8.5 Podstawowy system zarządzający ruchem robotów

Zaprezentowane do tej pory algorytmy pozwalają na skonstruowanie podstawowej wersji systemu agentowego zarządzającego ruchem robotów. Współpracować w nim będą następujące rodzaje agentów:

- Agent Map Server Agent (MSA), czyli implementacja usługi planowania trasy w środowisku.
- Agent Robot Worker Agent (RWA), który realizuje cele dwóch typów, czyli udostępnia dwie usługi:
  - podstawowe funkcje sterujące silnikami robota: jazda po prostej lub po łuku z zadaną prędkością i przez zadany czas,
  - autonomiczna nawigacja reaktywna, czyli metoda przemieszczania robota do określonego punktu z zastosowaniem algorytmu unikania kolizji.

  Zrealizowana implementacja RWA pozwala na kolejkowanie celów agenta. Dzięki temu możliwe jest zdefiniowanie sekwencji ruchów lub listy punktów, które robot po-

winien osiągnąć.

- Agent Task Executor Master Agent (TExMA), który zarządza ruchem od jednego do n robotów.

  Dla każdego z robotów generowany jest losowy punkt docelowy. TExMA wysyła do MSA żądanie zaplanowania trasy pomiędzy bieżącą lokalizacją i wybranym punktem docelowym. Następnie przekazuje agentowi RWA kolejne cele do osiągnięcia. Jeżeli RWA nie wykona kolejnego zaplanowanego celu (dojedzie do kolejnego punktu pośredniego na trasie) w przewidywanym czasie, TExMA czyści jego listę celów i ponownie wysyła żądanie zaplanowania trasy do MSA. Taki mechanizm pozwala na poprawne obsłużenie dynamicznej zmiany środowiska. Po osiągnięciu przez robota punktu docelowego generowany jest kolejny.

Tak skonstruowany system pozwala na bezpieczne i skuteczne przemieszczanie robotów w środowisku. Algorytm autonomicznej nawigacji reaktywnej gwarantuje bezpieczeństwo przemieszczania poprzez omijanie wszystkich napotkanych przeszkód. Własności trasy zaplanowanej przez MSA gwarantują, że odcinek łączący robota z następnym punktem do osiągnięcia nigdy nie przecina ściany, dzięki czemu robot jest w stanie do niego dojechać.

Można się jednak spodziewać, że wraz ze wzrostem liczby robotów skuteczność systemu będzie wyraźnie spadać. Duża liczba autonomicznie sterowanych robotów operujących na niewielkiej powierzchni powodować będzie konieczność częstego omijania ruchomych przeszkód, co może prowadzić do powstawania zatorów w wąskich przejściach.

Wyniki badań wydajności działania podstawowej wersji systemu zostały zaprezentowane w rozdziale 8.5.

8.6 Algorytm równoważenia obciążenia krawędzi grafu

Skuteczność zaprezentowanej, podstawowej wersji systemu spada gwałtownie wraz ze wzrostem liczby robotów. Jest to spowodowane powstawaniem zatorów w okolicach wąskich przejść w centralnej części środowiska. Przyczyną zatorów jest fakt, że większość tras pomiędzy losowymi punktami w środowisku prowadzi przez centralną część środowiska.

Pewnym rozwiązaniem zaistniałego problemu może być algorytm równoważenia obciążenia krawędzi grafu, implementowany przez MSA. Algorytm mierzy liczbę robotów n pokonujących każdą z krawędzi grafu w jednostce czasu i modyfikuje wagi krawędzi według wzoru: $w'_{AB} = w_{AB} * (1 + g * n)$, gdzie $g > 0$ to współczynnik skalujący. Takie zwiększenie wag krawędzi, które są najbardziej obciążone powoduje wyszukiwanie alternatywnych tras dla kolejnych robotów. MSA czerpie informacje o obciążeniu krawędzi z kierowanych do niego poleceń wyznaczenia trasy. Nie więc ma konieczności modyfikowania żadnych innych elementów systemu.

Wyniki badań nad wpływem algorytmu równoważenia obciążenia na skuteczność działania systemu zaprezentowano w rozdziale 9.4. Sprawdza się on bardzo dobrze w zastosowaniu do środowisk, w których istnieje wiele różnych ścieżek pomiędzy każdą parą punktów w środowisku. Jeżeli jednak w środowisku występuje obszar, przez który musi przejeżdżać duża liczba robotów, algorytm równoważenia obciążenia nie będzie przydatny. W takiej sytuacji jedyną metodą jest zastosowanie algorytmu koordynacji ruchu.

8.7 Algorytmy koordynacji ruchu

Podstawowy system nawigacji, który został zaprezentowany w poprzednich rozdziałach, działa wydajnie jedynie dla mało licznych grup robotów poruszających się w stosunkowo rozległym środowisku. Zwiększanie liczby robotów powoduje powstawanie zatorów w wąskich przejściach w centralnej części środowiska. Rozwiązaniem tego problemu jest dodanie agentów AMA zarządzających ruchem robotów we fragmentach środowiska.

Algorytmy implementowane przez agenty AMA są kluczowym elementem prezentowanego systemu nawigacji robotów. Zostały zaimplementowane dwie metody koordynacji ruchu:

- metoda zarządzania kolejnością przemieszczania robotów,
- metoda scentralizowanego planowania trajektorii i sterowania ruchem robotów.

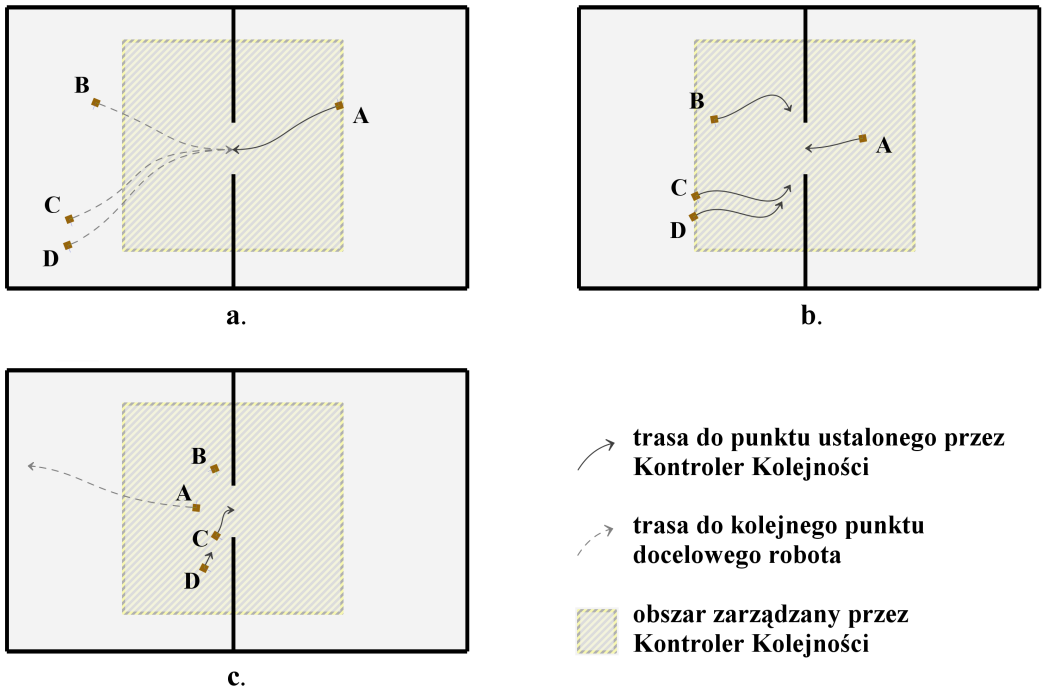
W dalszej części tego rozdziału zostaną one szczegółowo zaprezentowane. W rozdziale 9.5 przedstawione zostaną badania ich własności oraz porównanie skuteczności w różnych sytuacjach.

8.7.1 Kontroler kolejności dostępu i przejazdu

Algorytm reaktywnej nawigacji daje bardzo dobre efekty, gdy robot nie napotyka na dużą liczbę ruchomych przeszkód, w szczególności gdy będzie jedynym robotem pokonującym obszar. Wspomniana właściwość jest podstawą działania algorytmu kontrolera kolejności.

Kontroler kolejności nie przejmuje sterowania nad efektorami robota. Każdy z robotów nadal wykorzystuje autonomiczny algorytm nawigacji reaktywnej. Algorytm kontrolera kolejności ma za zadanie zagwarantować, że robot przemieszczający się przez obszar nie napotka innych robotów. W tym celu modyfikuje tymczasowo lokalizacje docelowe i prędkości przemieszczania każdego z robotów, ustawiając je w kolejkach. Kolejki robotów oczekujących na przejazd są rozmieszczone tak, by nie przeszkadzały w przemieszczaniu wybranego robota, który otrzymuje zezwolenie na przejazd. Zezwolenie jest realizowane poprzez przywrócenie oryginalnego punktu docelowego.

Schemat działania algorytmu prezentuje rysunek 8.4, na którym ciągłymi liniami zaznaczono trajektorię ustaloną przez kontroler kolejności, a przerywanymi trajektori robota



Rysunek 8.4: Działanie algorytmu kontrolera kolejności zastosowanego do problemu wąskiego przejścia. W sytuacji **a.** robot **A** otrzymuje zezwolenie na przejazd. W sytuacji **b.** pozostałe roboty są kierowane do kolejek. W sytuacji **c.** robot **A** pokonał przejście, więc robot **C** otrzymuje zezwolenie na przejazd.

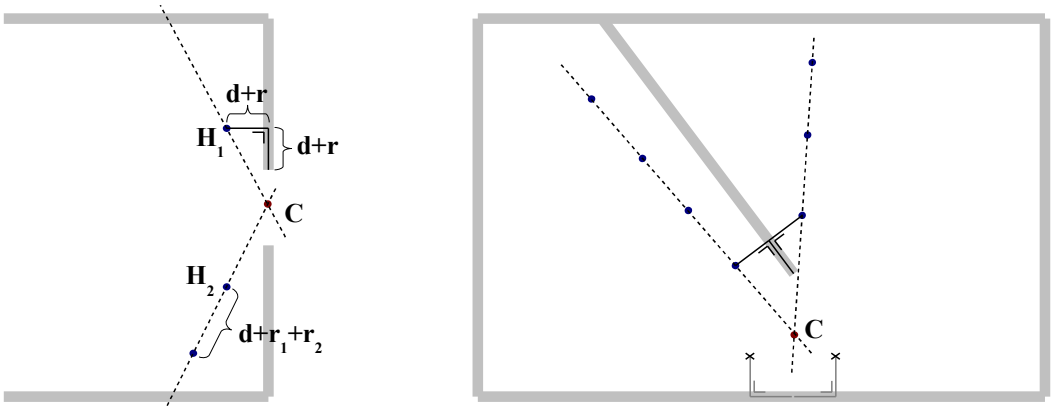
sterowanego autonomicznie. Po wjechaniu w zarządzany obszar (rys. 8.4 a.), robot **A** otrzymuje natychmiast zezwolenie na przejazd i kontynuuje ruch w kierunku bieżącego celu – algorytm nie modyfikuje jego punktu docelowego ani prędkości. Przerywaną linią zaznaczone są aktualne plany robotów **B**, **C** oraz **D**, które również zmierzają do środka wąskiego przejścia. Gdy roboty z lewej strony przejścia wjadą w obszar (rys. 8.4 b.), algorytm Kontrolera zmieni ich punkty docelowe i skieruje je do dwóch kolejek rozmieszczonych po obu stronach przejścia. Roboty **B** i **C** zajmą pierwsze pozycje w kolejkach, robot **D** będzie umieszczony w dolnej kolejce, za robotem **C**.

W chwili, gry robot **A** minie wąskie przejście (rys. 8.4 c.), zezwolenie na przejazd otrzyma robot **C**, ponieważ jest on pierwszym robotem w najdłuższej kolejce. Robot **D** zostanie wtedy przemieszczony na pierwsze miejsce w dolnej kolejce.

Algorytm kontrolera kolejności modyfikuje dodatkowo prędkości poruszania się robotów, które zostają skierowane do kolejki oczekujących. Prędkość robota zmierzającego do kolejki to: $v_k = \frac{v*f}{n}$, gdzie v to standardowa prędkość tego robota, $0 < f < 1$ to współczynnik redukcji, a n to pozycja robota w kolejce, do której został przypisany. Znaczna redukcja

prędkości robotów umieszczonych na odległych pozycjach w swoich kolejkach, zapobiega chaotycznym ruchom wielu robotów, które z dużą prędkością próbują osiągnąć pozycję w kolejce, i wzajemnie sobie przeszkadzają. Umożliwia też znacznie łatwiejsze opuszczenie obszaru przez robota przejeżdżającego z drugiej strony wąskiego przejścia.

Sposób rozmieszczenia kolejek w obszarze musi spełniać kilka kryteriów. Przede wszystkim kolejki nie mogą kolidować z przeszkodami, nie mogą przecinać ścian. Roboty oczekujące w kolejkach nie powinny wpływać na ruch robota pokonującego przejście, więc fragmenty kolejek nie powinny znajdować się naprzeciwko przejścia. Dodatkowo, pierwszy z robotów w każdej z kolejek powinien być w stanie bez przeszkód i w minimalnym czasie dojechać do przejścia. Algorytm wyznacza linie proste, na których znajdować się będą lokalizacje kolejnych robotów w kolejkach. Linie te przechodzą przez środek przejścia C oraz przez jeden z punktów H_1 lub H_2 (rysunek 8.5 a.), będący początkiem kolejki. Kolejne punkty na prostej, w których umieszczane są roboty umieszczane są w odległości $r_1 + r_2 + d$ od poprzednich punktów, gdzie r_1 i r_2 to promienie okręgów opisujących sąsiadujące w kolejce roboty, a d to stały, bezpieczny dystans pomiędzy robotami, będący ustalonym parametrem algorytmu.



Rysunek 8.5: Działanie algorytmu pozycjonowania robotów oczekujących na przejazd w różnych przypadkach rozmieszczenia przeszkód w środowisku. W sytuacji **a.** możliwe jest wyznaczenie dwóch kolejek z każdej strony przejścia, w sytuacji **b.** z każdej strony zostanie wyznaczona tylko jedna kolejka.

Kolejne roboty są umieszczane w kolejce, jeżeli wyznaczony dla robota punkt znajduje się wewnątrz obszaru, a jego odległość od każdej z przeszkód jest większa lub równa $d+r$. Stosowane jest też zabezpieczenie przeciwko umieszczaniu kolejek naprzeciwko przejścia(rysunek 8.5 b. u dołu), które wyraża się wzorem: $|CH| >= \sqrt{[(d+r)\sqrt{2}]^2 + (\frac{c}{2})^2}$, gdzie c to szerokość przejścia.

Robot jest zawsze kierowany do tej kolejki, w której następne wolne miejsce jest bliżej aktualnej pozycji robota. Jeżeli w żadnej z kolejek nie ma wolnych miejsc, maksymalna prędkość robota jest ustawiana na 0.

8.7.2 Scentralizowane planowanie i wykonanie trajektorii bezkolizyjnych

Opisany wcześniej algorytm kontrolera kolejności nie przejmuje sterowania efektorami (silnikami) robota. Wykorzystuje jedynie właściwości algorytmu autonomicznej nawigacji reaktywnej. Takie podejście jest skuteczne i obliczeniowo tanie, ale nie daje możliwości uzyskania optymalnego rozwiązania problemu koordynacji. Nie pozwala także na oszacowanie czasu trwania ruchu n robotów, ponieważ każdy z robotów jest de facto sterowany autonomicznie i może zachowywać się nieprzewidywanie.

Metoda centralnego planowania i wykonania trajektorii, która bezpośrednio kontroluje efektory napędowe koordynowanych robotów, może wyznaczyć znacznie bardziej optymalne sterowanie. Może ona wykorzystać wszystkie możliwości sprzętu i nie jest podatna na wady algorytmu autonomicznego, który nie korzysta z wiedzy o planowanych ruchach innych robotów. Implementacja takiej metody dziali problem na etapy planowania i wykonania, czego efektem jest powstawanie wiedzy o przebiegu ruchu wszystkich robotów zanim ten ruch się rozpocznie. Dzięki temu można precyzyjnie wyznaczyć czas zakończenia ruchu każdego z robotów.

Opracowana metoda jest algorytmem koordynacji ruchu robotów, działającym w czasie rzeczywistym. Oznacza to, że nie wymaga on dostarczenia informacji o planach każdego z robotów przed rozpoczęciem działania – mogą one pojawiać się lub zmieniać w trakcie działania systemu. Tylko taki schemat działania nadaje się do zastosowania w rzeczywistych systemach koordynacji ruchu. W odróżnieniu od innych metod znanych z literatury, opisywane tu rozwiązanie bierze pod uwagę dynamikę robota zarówno w fazie planowania jak i wykonania trajektorii. Nie ogranicza się też ono wyłącznie do robotów holonomicznych, może obsługiwać roboty niezdolne do zmiany orientacji bez zmiany pozycji.

Zrealizowana implementacja przyjmuje dwa uproszczenia:

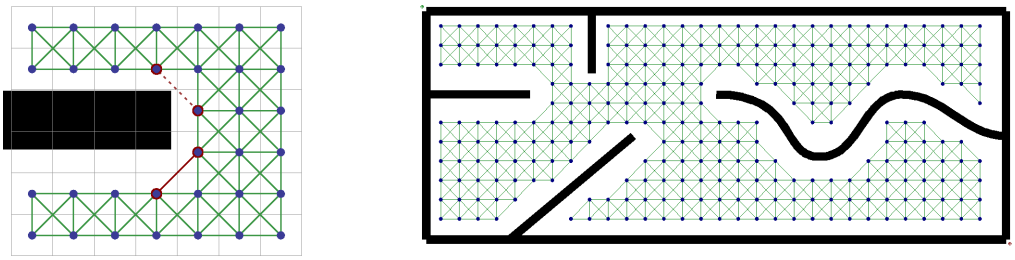
- wszystkie roboty mają tą samą wielkość,
- zmiany w środowisku (np. pojawienie się nowe) nie są obsługiwane.

Są to jednak wyłącznie ograniczenia testowej implementacji. Bardziej zaawansowana implementacja metody może być z powodzeniem stosowana do koordynowania ruchu grup robotów heterogenicznych w dynamicznym środowisku.

Model środowiska, wykorzystywany przez agenta AMA implementującego opisywany algorytm, jest grafem pokrywającym dostępne fragmenty obszaru. Minimalna odległość

pomiędzy wierzchołkami grafu wynosi $d = 2r + \sigma^2$, czyli jest równa średnicy robota powiększonej o błąd lokalizacji. Przy takim założeniu nie powinno dojść do kolizji dwóch robotów zlokalizowanych w dwóch różnych wierzchołkach grafu.

Graf jest konstruowany z wykorzystaniem siatki zajętości. Obszar jest pokrywany siatką o boku d. Wierzchołki grafu są umieszczane w środkach każdej komórki siatki, w której nie ma żadnych fragmentów przeszkód. Przykład działania algorytmu tworzenia grafu prezentuje rysunek 8.6; w części **a.** siatkę zajętości oznaczono kolorem szarym. Każdy z wierzchołków może zostać połączony krawędziami z ośmioma najbliższymi wierzchołkami – cztery z nich znajdują się w odległości d, cztery w odległości $d\sqrt{2}$. Krawędzie o długości d są tworzone zawsze, gdy sąsiednie wierzchołki istnieją. Krawędzie o długości $d\sqrt{2}$ są tworzone pod warunkiem, że w kwadratowym fragmencie terenu, którego narożnikami są łączone wierzchołki grafu, nie ma żadnych przeszkód. Taki kwadratowy fragment nie jest komórką siatki, ale składa się z czterech ćwiartek sąsiadujących komórek. Na rysunku 8.6a. wyróżniono cztery wierzchołki, z których dolne dwa zostaną połączone krawędzią, natomiast dwa górne nie (linia przerywana).



a. b.

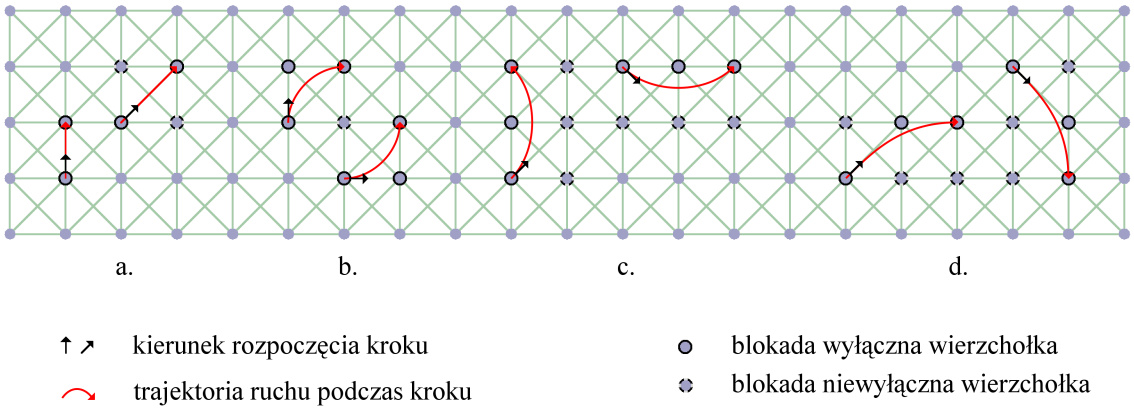
Rysunek 8.6: Konstruowanie modelu środowiska wykorzystywanego przez algorytm koordynacji. Kolorem czarnym oznaczono przeszkody, wierzchołki i krawędzie grafu oznaczono kolorami niebieskim i zielonym.

Implementowany przez opisywanego AMA algorytm koordynacji polega na planowaniu u wykonywaniu bezkolizyjnych trajektorii dla każdego z koordynowanych robotów. Każda nowa trajektoria jest planowana tak, że nie powoduje ona kolizji z wcześniejszymi trajektoriami, które są już wykonywane. Algorytm planowania trajektorii wykorzystuje opisywany graf, jednak nie jest to typowy algorytm wyszukiwania ścieżki w grafie. Wyznaczona przez algorytm trajektoria robota składa się z listy tzw. **kroków**. Krok jest strukturą danych zawierającą informacje o:

- poprzednim kroku,
- wierzchołku grafu, w którym bieżący krok się kończy,

- liniowej prędkości robota pod koniec kroku,
- orientacji robota pod koniec kroku, która jest zawsze jednym z ośmiu kątów, pod jakimi krawędź może wychodzić z wierzchołka,
- szacowany czas zakończenia kroku (ETA).

Algorytm planowania czasooptymalnej trajektorii dla pojedynczego robota wykorzystuje listę kroków posortowanych według wartości ETA, która zawiera kroki do przetworzenia. Inicjalizacja algorytmu polega na dodaniu do listy jednego kroku, który reprezentuje początkowy stan robota (pozycję i orientację) oraz czas ETA równy planowanemu początkowi ruchu. Dla każdego kolejnego kroku z listy tworzone są kolejne kroki – ta operacja jest wykonywana przez komponenty, implementujące wspólny interfejs, z których każdy jest w stanie tworzyć kroki określonego typu. Przykładowe typy kroków są widoczne na rysunku 8.7.



Rysunek 8.7: Trajektorie ruchu planowane przez różne algorytmy konstruowania kroku. Kroki w sytuacji **a.** to ruch po prostej; sytuacje **b.**, **c.** i **d.** to ruch po łukach o różnych promieniach.

Najprostszym rodzajem kroku jest obrót w miejscu – pozycja robota nie ulega zmianie, modyfikowana jest jego końcowa orientacja, a końcowa prędkość wynosi 0. Oczywiście jest on brany pod uwagę jedynie podczas planowania trajektorii robotów, które są w stanie zmieniać orientację bez zmiany pozycji. Kroki na rysunku 8.7 a. reprezentują ruch po prostej; występują tylko dwie długości tego typu kroków, d oraz $d\sqrt{2}$. Pozostałe typy kroków (rys. 8.7 b., c. oraz d.) odpowiadają za ruch po łuku o różnym promieniu.

Oczywiście możliwe jest zdefiniowanie algorytmów wyznaczających inne rodzaje kroków, na przykład dłuższych łuków o większym promieniu, których wykorzystanie pozwoliłoby na tworzenie trajektorii o łagodniejszych zakrętach. Jednak zrealizowany mecha-

nizm gwarantowania bezkolizyjności planowanych trajektorii działa bardziej optymalnie dla krótszych kroków – szczegóły zostaną opisane w dalszej części.

Wykorzystywany podczas planowania i wykonania trajektorii model robota mobilnego opisuje charakterystykę robota następującymi parametrami:

- maksymalna prędkość liniowa,
- maksymalne przyspieszenie liniowe,
- maksymalne opóźnienie liniowe – najczęściej większe od przyspieszenia,
- maksymalne akceptowalne przyspieszenie dośrodkowe,
- minimalny promień skrętu.

Maksymalna prędkość liniowa oraz akceptowalne przyspieszenie dośrodkowe są limitami gwarantującymi bezpieczeństwo i mogą zostać ustalone na dowolne nieujemne wartości. Wartości przyspieszenia i opóźnienia liniowego oraz minimalnego promienia skrętu powinny być dokładnie zmierzone dla każdego robota. Są one wykorzystywane do wyznaczania sterowania podczas pokonywania trajektorii.

Przyjętym kryterium jakości trajektorii jest czas trwania ruchu. W związku z tym prędkość liniowa każdego z robotów jest przez algorytmy planujące trajektorię zwiększana, aż do osiągnięcia któregoś z limitów. Trzeba jednak pamiętać, że niektóre rodzaje kroków wymagają określonej prędkości liniowej w chwili rozpoczęcia – na przykład krok, polegający na obróceniu robota w miejscu, wymaga by prędkość liniowa była równa 0. Dlatego też każdy z algorytmów wyznaczania musi być w stanie przekształcić wygenerowany krok do wersji, w której końcowa prędkość liniowa jest równa zadanej wartości. Taka operacja jest zawsze wykonalna, przy założeniu, że algorytmy generujące poprzednie kroki również ją wspierają (niezbędna utrata prędkości może nie być możliwa w ciągu jednego kroku).

Utworzone kroki są dodawane do listy, pod warunkiem, że nie istnieje jeszcze krok kończący się w tym samym wierzchołku i w tej samej orientacji, ale z mniejszym ETA. Tylko najszybsza metoda dotarcia do danego z wierzchołka pod określonym kątem jest dalej przetwarzana i może stać się źródłem nowych kroków. Procedura tworzenia nowych kroków jest kontynuowana, aż do czasu, gdy:

- lista kroków do przetworzenia zostanie opróżniona – oznacza to, że robot nie jest w stanie osiągnąć pożądanej lokalizacji, lub
- skonstruowany zostanie krok kończący się w docelowym wierzchołku, a na liście kroków do przetworzenia nie będzie kroków o ETA mniejszym od tego kroku.

Wyznaczona trajektoria jest definiowana ostatnim krokiem oraz listą jego kroków poprzednich. Ta trajektoria jest najszybszą metodą osiągnięcia docelowego wierzchołka przez robota, przy zachowaniu ograniczeń narzuconych przez algorytmy konstruowania kroków.

Opisany algorytm planowania trajektorii jest obliczeniowo kosztowny. Poszukuje rozwiązania w każdym możliwym kierunku, zbierając informacje o możliwości najszybszego osiągnięcia każdego z wierzchołków w grafie, aż do odszukania wierzchołka docelowego. Złożoność jest wprost proporcjonalna do liczby wierzchołków oraz do liczby rodzajów generowanych kroków. Wydajność algorytmu mogłaby zostać znacząco poprawiona poprzez zastosowanie metody zmieniającej kolejność analizowania kroków, na przykład heurystyki szacującej czas potrzebny na osiągnięcie celu rozpoczynając od danego kroku.

Planowanie trajektorii w sytuacji, gdy inne roboty wykonują już wcześniej zaplanowanie trasy, wymaga sprawdzenia, czy nowa trasa nie koliduje z wcześniejszą. Operacja ta jest wykonywana w trakcie planowania kolejnych kroków.

Każdy krok zaakceptowanej do wykonania trajektorii generuje w grafie czasowe rezerwacje wierzchołków. Okres trwania rezerwacji to czas pomiędzy ETA kroku poprzedniego i kroku bieżącego. Wszystkie wierzchołki rezerwowane przez krok są zarezerwowane na ten sam okres czasu, dlatego też algorytmy generowania kroków powinny wykorzystywać minimalną liczbę wierzchołków. Dla robotów, które nie mają żadnych zadań i pozostają w spoczynku, tworzona jest rezerwacja jednego wierzchołka z nieskończonym czasem zakończenia.

Istnieją dwa rodzaje rezerwacji wierzchołków: wyłączna i niewyłączna. Wierzchołki z rezerwacją niewyłączną mogą być w nachodzącym okresie czasu zarezerwowane niewyłączeni przez inne kroki; wierzchołki z rezerwacją wyłączną nie mogą być wykorzystane przez inne kroki w żaden sposób. Sposób rezerwowania wierzchołków przez różne algorytmy generowania kroków są widoczne na rysunku 8.7. Rezerwacje są tworzone dla wierzchołków odległych od trasy robota o mniej niż d. Niewyłączne rezerwacje są tworzone w tych wierzchołkach, od których planowana trasa jest odległa o $\frac{d}{2}\sqrt{2}$ lub więcej.

Podczas planowania kroku algorytm sprawdza, czy żadna z istniejących rezerwacji nie została naruszona. Jeżeli naruszenie zostanie wykryte, algorytm generujący poprzedni krok jest proszony o dostarczenie wersji kroku, która ma wartość ETA taką, jak czas zakończenia kolidującej rezerwacji. Operacja opóźnienia czasu zakończenia kroku może zostać zrealizowana na wiele sposobów, ponieważ nadmiarowy czas może zostać stracony w różnych momentach. Zastosowany algorytm zawsze maksymalnie opóźnia wytracenie czasu, by zminimalizować liczbę kroków, które muszą zostać zmodyfikowane. Opóźnianie kroków może spowodować powstanie innych naruszeń rezerwacji, które muszą zostać rozwiązane. Może to spowodować, że opóźniona wersja kroku nie będzie się kończyła dokładnie w żądanym czasie. Dlatego też operacja opóźnienia ma zwrócić krok z ETA nie mniejszym niż zadana

wartość. Tak zdefiniowana operacja jest zawsze wykonalna, choć zwrócony krok może mieć nieskończoną wartość ETA, jeżeli naruszona zostanie rezerwacja nieskończona.

Teoretycznie, jeżeli trajektoria ruchu została zaplanowana z uwzględnieniem dynamicznych właściwości robota, powinna być wystarczająca do wyznaczenia ciągu sterowań, które muszą zostać podane na efektory w celu jej pokonania. W rzeczywistości występuje jednak wiele zjawisk, które nie są brane pod uwagę w wykorzystywanym modelu, a powodują niedokładności w działaniu urządzenia mechanicznego. Te drobne błędy nawarstwiają się i z czasem powodują znaczne odstępstwa od przewidywań. Dlatego też zaplanowana trajektoria ruchu musi być przez algorytm sterujący robotem traktowana jako cel, a nie jako podstawa do wyliczania sterowania.

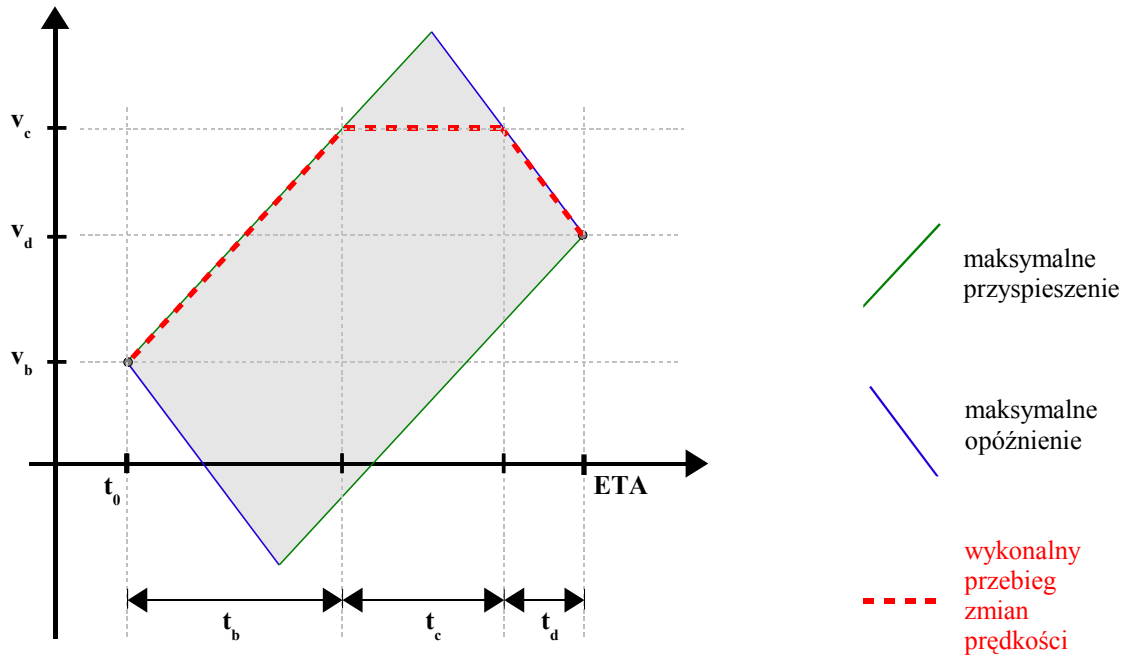
Zastosowany algorytm wykonania kroku rozdziela obliczanie prędkości liniowej i kątowej robota. Dzięki temu możliwe jest zastosowanie tych samych równań obliczania pożądanej prędkości liniowej dla wszystkich rodzajów kroków. Pierwszym etapem jest wyznaczenie długości krzywej, jaka pozostała do pokonania w ramach kroku. Krzywa jest przybliżana odcinkiem i fragmentem okręgu, które są styczne siebie, do kierunku robota i do docelowego kierunku kroku.

Obliczony dystans, pożądana prędkość i czas osiągnięcia docelowej lokalizacji kroku są ograniczeniami, które muszą zostać zachowane, by robot poprawnie podążał trajektorią. Do dyspozycji algorytmu pozostaje manipulowanie prędkością liniową robota. Rysunek 8.8 pokazuje zależność pomiędzy czasem a prędkością robota. Ciemniejszy region oznacza prędkości, jakie robot może rozwinąć w czasie trwania kroku, by ukończyć krok z pożądaną prędkością v_d.

Algorytm sterujący zakłada, że prędkość robota może rosnąć z maksymalnym przyspieszeniem, maleć z maksymalnym opóźnieniem lub pozostać stała. Jest to dość typowe działanie kołowych robotów mobilnych napędzanych elektrycznie. Dystans pokonany przez robota wyraża się wzorem:

$$d = \int_{t_0}^{ETA} v(t)dt = \frac{(v_b + v_c)}{2} * t_b + v_c * t_c + \frac{(v_c + v_d)}{2} * t_d \tag{8.1}$$

gdzie:
d – pozostały dystans,
t_0 – aktualny czas
v_b – prędkość początkowa,
v_c – prędkość stała, którą robot powinien osiągnąć podczas kroku,
v_d – pożądana prędkość w chwili zakończenia kroku,
$t_b + t_c + t_d = ETA - t_0 = t$.



Rysunek 8.8: Trajektorie ruchu planowane przez różne algorytmy konstruowania kroku.

Wartości t_b, t_c oraz t_d można zapisać jako:

$$t_b = \frac{v_c - v_b}{a_1}, \quad t_d = \frac{v_d - v_c}{a_2}, \quad t_c = t - (t_b + t_c), \tag{8.2}$$

gdzie każda z wartości a_1 oraz a_2 może byś przyspieszeniem lub opóźnieniem, w zależności od sytuacji. . Równanie 8.1 może więc zostać wyrażone jako:

$$v_c^2(\frac{1}{2 * a_2} - \frac{1}{2 * a_1}) + v_c(t + \frac{v_b}{a_1} - \frac{v_d}{a_2}) + (\frac{v_d^2}{2 * a_2} - \frac{v_b^2}{2 * a_1} - d) = 0. \tag{8.3}$$

Zakładając, że wartości t_b, t_c oraz t_d są nieujemne, to kwadratowe równanie ma jedno lub zero rozwiązań. Jeżeli rozwiązanie nie istnieje, przynajmniej jedno ograniczenie zostanie naruszone przez robota.

Obliczona wartość v_c jest prędkością liniową, jaka powinna zostać podana jako sterowanie robota. W zależności od typu robota należy ją odpowiednio zintegrować z wartością drugiej pochodnej krzywej, po której robot powinien się przemieszczać. W przypadku robotów o napędzie różnicowym równania są następujące:

$$\begin{aligned} v_r &= v_c * (1 - \frac{w}{r}) \\ v_l &= v_c * (1 + \frac{w}{r}) \end{aligned} \tag{8.4}$$

gdzie:
v_r, v_l – prędkość odpowiednio prawych i lewych kół robota,
w – połowa odległości między prawymi a lewymi kołami robota,
r – promień okręgu po jakim w danej chwili powinien podążać robot.

Jeżeli wartości przyspieszenia i opóźnienia robota zostały zmierzone dokładnie, ustawianie wartości v_r i v_l w enkoderach silników powinno spowodować, że robot będzie podążał dokładnie wzdłuż pożądanej trajektorii.

Złożoność obliczeniowa algorytmu kontrolera kolejności jest znacznie mniejsza niż algorytmu planowania i wykonania trajektorii. Abstrahując od wydajności obu metod, pamiętać należy jednak, że kontroler kolejności nie może być stosowany w obszarach o dowolnym kształcie. Nie zawsze możliwe jest odpowiednie rozłożenie kolejek robotów oczekujących na przejazd przez obszar. Algorytm planowania i wykonania trajektorii jest w stanie skutecznie zarządzać ruchem robotów w dowolnie skomplikowanym obszarze.

Badania skuteczności działania obu metod oraz porównanie uzyskanych wyników z metodą autonomiczną zostanie zaprezentowane w rozdziale 9.5.

8.8 Zintegrowany system nawigacji robotów mobilnych

Kompletny system nawigacji robotów mobilnych jest rozbudową wersji podstawowej, zaprezentowanej w rozdziale 8.5. Po wzbogaceniu go o możliwość tworzenia i konfigurowania agentów AMA charakteryzuje się wszystkimi pożądanymi cechami wysokiej jakości systemu informatycznego. Agenty współpracujące w ramach systemu to:

- Agenty typu Map Server Agent (MSA), które tworzą mapę wielomodelową i realizują usługę planowania tras dla robotów.
- Agenty typu Robot Worker Agent (RWA), które sterują efektorami poszczególnych robotów i implementują algorytm autonomicznej nawigacji reaktywnej.
- Agenty typu Task Executor Master Agent (TExMA), które zarządzają ruchem przydzielonych robotów, nadając im losowe cele do osiągnięcia.
- Agenty typu Area Master Agent (AMA), które zarządzają ruchem w podległych im obszarach.

Agenty MSA, RWA oraz TExMA działają analogicznie, jak w przypadku podstawowej wersji systemu, opisanej w rozdziale 8.5. Rozbudowa systemu o agenty zarządzające ruchem w niektórych obszarach nie wymaga modyfikacji ani nawet zatrzymywania działania pozostałych elementów systemu.

Dodane w kompletnej wersji agetny AMA można tworzy i usuwać w dowolnym momencie pracy systemu. W celu utworzenia nowego AMA konieczne jest zdefiniowanie zarządzanego obszaru i wybranie rodzaju algorytmu zarządzania ruchem. Podczas działania każdy AMA obserwuje podległy mu obszar i komunikuje się z agentami RWA, których roboty w ten obszar wjeżdżają. Po wykryciu nowego robota w obszarze, AMA przejmuje nad kontrolę nad odpowiednim agentem RWA, a po udanym pokonaniu obszaru informuje RWA o powrocie pod kontrolę odpowiedniego agenta TExMA. W zależności od stosowanej metody koordynacji, AMA wykorzystują algorytm autonomicznej nawigacji RWA lub jego niskopoziomowe metody kontroli nad efektorami robota.

8.9 Własności zintegrowanego systemu nawigacji robotów mobilnych

W rozdziale 3 wskazano najważniejsze cechy, jakimi powinien charakteryzować się system informatyczny zarządzający działaniem robotów mobilnych. Są to:

- skalowalność,
- rozszerzalność,
- wysoka dostępność, odporność na awarie,
- łatwość ponownego wykorzystywania elementów.

Możliwość poszerzania środowiska działania działania jest w prezentowanym systemie praktycznie nieograniczona. Dodanie nowego fragmentu terenu realizuje się poprzez utworzenie nowego agenta MSA, który konstruuje model terenu. Następnie MSA musi zostać włączony w hierarchię mapy wielomodelowej poprzez przypisanie go do odpowiedniego agenta MSA wyższego rzędu.

Dodawanie nowych robotów również jest zadaniem trywialnym. Wymaga jedynie utworzenia nowego agenta RWA, który od momentu uruchomienia może być wykorzystywany przez agenty TExMA. Badania wydajności systemu nawigacji podczas obsługiwania licznych grup robotów zostaną zaprezentowane w rozdziale 9.6.

Podstawowe kierunki rozszerzania systemu o nowe możliwości to dodawanie robotów nowego typu, powiększanie środowiska o nowego rodzaju elementy (np. windy, automatyczne drzwi itp.) czy dodawanie nowych zadań, które będą realizowane z wykorzystaniem robotów. W prezentowanym systemie dodanie robota nowego rodzaju wymaga jedynie zaimplementowania agenta RWA. Po jego uruchomieniu robot będzie mógł rozpocząć działania wraz z pozostałymi robotami obecnymi już w systemie.

Rozszerzenie środowiska działania robotów o obszar nowego rodzaju wymagać będzie zaimplementowania agenta AMA, który będzie w stanie poprawnie kontrolować ruch w

obszarze. Przykładem może być obsługa windy, która może zostać zrealizowana w sposób podobny do kontroli kolejności przejazdu. AMA może ustawiać roboty oczekujące na windę w kolejki i w odpowiedni sposób umieszczać je w kabinie.

Za realizację zadań odpowiadają agenty TExMA. Zmiana funkcjonalności systemu wymaga więc dodania nowego rodzaju agenta TExMA, który przejmie kontrolę nad odpowiednimi robotami i zarządza ich działaniem.

Najważniejszą własnością prezentowanego systemu w kontekście rozszerzalności jest brak konieczności zatrzymywania systemu podczas wprowadzania zmian. Program nowych agentów można załadować do kontenera JADE „w biegu", jako bibliotekę klas. Następnie można uruchamiać nowe agenty i wykorzystywać ich funkcje. Pozostałe agenty pracują podczas całej operacji bez zakłóceń.

W systemie wykorzystującym roboty mobilne można spodziewać się występowania awarii dwóch rodzajów: sprzętu i oprogramowania. Separacja przestrzeni działania agentów i robotów, która jest podstawą prezentowanego w pracy podejścia, pozwala na łatwe wykrycie rodzaju awarii. W przypadku wadliwego działania jednego z robotów, agent RWA ma za zadanie wykrycie sytuacji i przekazanie informacji o wyjątku do agenta TExMA. Sposób obsługi tego typu awarii nie był rozważany w prezentowanej implementacji.

Awaria oprogramowania, czyli błąd w działaniu jednego z agentów, również może być łatwy wykryty i obsłużony. Za detekcję awarii agenta RWA odpowiedzialny jest agent agent TExMA. Awaria jednego z agentów AMA nie spowoduje wstrzymania pracy systemu. Jej efektem będzie zanik kontrolki nad ruchem w podległym obszarze. Agenty RWA będą stosowały autonomiczny algorytm reaktywny, co prawdopodobnie negatywnie wpłynie na skuteczność przemieszczania, ale nie wstrzyma ruchu robotów.

Awaria agenta MSA jest bardzo mało prawdopodobne, ponieważ jego algorytm jest niezbyt skomplikowany. Wykrycie tego typu błędu również nie jest skomplikowane, a objawiać się będzie brakiem odpowiedzi na żądanie wyznaczenia trasy. Błąd w działaniu agenta TExMA jest zdecydowanie najtrudniejszy do wykrycia, ponieważ TExMA nie jest zależny ani bezpośrednio wykorzystywany przez inne agenty. Stanowi on jednak w prezentowanym systemie implementację funkcjonalności, więc kontrolą poprawności jego działania może zajmować się operator systemu.

Przeprowadzone badania zachowania systemu po awarii poszczególnych elementów zostaną opisane w rozdziale 9.6.

Podział funkcjonalności pomiędzy poszczególne agenty, jaki został zastosowany w prezentowanym systemie, zapewnia duże możliwości w zakresie wielokrotnego wykorzystywania kodu. Ścisłe powiązanie kodu programu agenta z rodzajem wykorzystywanego sprzętu występuje jedynie w przypadku agenta RWA, który musi być implementowany dla każdego rodzaju robota z osobna. Implementacja RWA dla określonego typu robota może być

z powodzeniem wykorzystywana w innych systemach, w których ten typ robota będzie występował.

Agent MSA, czyli implementacja podsystemu konstruowania mapy hierarchicznej i planowania tras, jest całkowicie niezależna od rodzaju robotów czy zadań przez nie wykonywanych. Może być bez zmian wykorzystywana w różnego typu systemach wykorzystujących roboty mobilne.

Agenty typu AMA są w założeniu również niezależne od sprzętu i pozostałych elementów systemu. Mogą zarządzać ruchem dowolnych robotów, przy założeniu udostępniania przez RWA odpowiednich usług. Zrealizowana na potrzeby prototypu implementacja algorytmu planowania i wykonania trajektorii przyjmuje pewne dodatkowe założenia dotyczące własności robotów, Stwarza to trudności z możliwością wykorzystania tej konkretnej implementacji w innych systemach. Implementacja kontrolera kolejności jest pod tym kątem bardziej uniwersalna i przenośna, ponieważ wymaga jedynie usługi autonomicznej nawigacji reaktywnej, nie przyjmując żadnych dodatkowych założeń.

Agent TExMA, który generuje punkty docelowe dla poszczególnych robotów, również nie jest zależny od rodzaju robotów ani od usług dostarczanych przez poszczególne RWA. Może być z powodzeniem wykorzystywany w innych systemach, choć realizowana przez jego algorytm funkcjonalność nie jest zbyt użyteczna.

Analiza własności zaprezentowanego systemu pozwala stwierdzić, że cechuje się pożądanymi cechami pozafunkcjonalnymi. W dalszej części pracy przedstawione zostaną badania i testy potwierdzające tą tezę.

9 Badania własności zrealizowanego systemu

Analiza cech stworzonego systemu zarządzającego ruchem robotów mobilnych pozwala sądzić, że będzie się on charakteryzował pożądanymi cechami pozafunkcjonalnymi. W tym rozdziale opisane zostaną wszystkie badania i testy poszczególnych elementów składowych systemu. Zaprezentowane zostaną porównania skuteczności różnych metod rozwiązywania poszczególnych problemów nawigacji. Przedstawione też zostaną wyniki badań i testów kompletnego systemu, potwierdzające prawdziwość tezy.

W pierwszej części rozdziału przedstawiona zostanie platforma testowa, która została wykorzystana do przeprowadzenia badań. W testach stosowano rzeczywiste roboty oraz środowisko do symulacji robotów, które pozwoliło na sprawdzenie zachowania systemu podczas zarządzania bardzo licznymi grupami robotów.

W dalszej części rozdziału zostaną opisane przeprowadzone badania oraz uzyskane wyniki testów:

- Porównanie wydajności algorytmów planowania tras dla robotów (rozdział 9.2). Jak wspomniano w rozdziale 7.3, zastosowanie typowych algorytmów wyszukiwania najkrótszej ścieżki w grafie do planowania tras dla licznych grup robotów w rozległym środowisku może nie być dostatecznie wydajne. Porównanie ma na celu eksperymentalne zademonstrowanie tego problemu. Wyniki testów algorytmów Dijkstry oraz A* zostaną porównane z wynikami metody wielomodelowej, która została zaproponowana w niniejszej pracy i jest stosowana w prezentowanym systemie.

- Badania własności podstawowej wersji systemu zarządzającego ruchem robotów (rozdział 9.3). W wersji podstawowej systemu współpracują agenty typu MSA, RWA oraz TExMA. Tak skonfigurowany system zapewnia możliwość bezpiecznej i skutecznej nawigacji robotów, jednak nie przewiduje możliwości koordynacji ruchu w wybranych obszarach. Konfiguracja zostanie przetestowana pod kątem wydajności nawigacji w zależności od liczby robotów jednocześnie znajdujących się w środowisku.

- Rozszerzenie wersji podstawowej systemu o algorytm równoważenia obciążenia (rozdział 9.4). Dodanie algorytmu równoważenia obciążenia powoduje omijanie przez roboty fragmentów środowiska, w których znajduje się najwięcej robotów. Zaprezentowane zostaną wyniki testów wydajności systemu w zależności od parametrów algorytmu równoważenia obciążenia.

- Porównanie skuteczności algorytmów koordynacji ruchu (rozdział 9.5). Możliwość dodania komponentów zarządzających ruchem robotów w najbardziej obciążonych fragmentach środowiska jest kluczową cechą kompletnej wersji prezentowanego systemu. W celu dobrania odpowiednich metod zarządzania ruchem konieczne jest przebadanie

ich własności i wydajności. Zaprezentowane zostaną wyniki testów porównawczych różnych metod koordynacji w zastosowaniu do dwóch typowych problemów: wąskiego przejścia i dostępu do zasobu.

- Badania zintegrowanego systemu nawigacji robotów mobilnych (rozdział 9.6). Kompletna wersja systemu, wzbogacona o agenty AMA koordynujące ruch w wybranych obszarach powinna cechować się skalowalnością, rozszerzalnością i odpornością na awarie. Przedstawione zostaną wyniki badań systemu po kątem możliwości zwiększania liczebności grupy robotów oraz rozbudowywania środowiska, jak również sposób obsługiwania przez system sytuacji wyjątkowych. Wykazane zostanie, że stworzony system charakteryzuje się pożądanymi cechami pozafunkcjonalnymi, które zostały zdefiniowane w rozdziale 3.

9.1 Roboty i środowisko ich działania, czyli przestrzeń rzeczywista

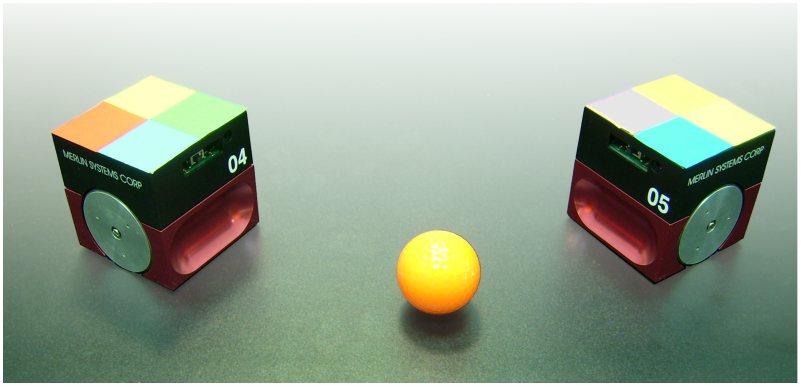
Zrealizowany system agentowy działa w przestrzeni wirtualnej, tworzonej przez platformę agentową. Za zadanie ma sterowanie robotami mobilnymi, które przebywają i działają w przestrzeni rzeczywistej. W celu zbadania własności systemu wykorzystane zostaną dwa sposoby realizacji przestrzeni rzeczywistej:

- platforma sprzętowa, składająca się z grupy robotów typu Miabot działających w specjalnie przygotowanym, niewielkim środowisku,
- system symulacji robotów mobilnych, który pozwala na realistyczną symulację zachowania bardzo licznych grup robotów w rozległych środowiskach.

9.1.1 Platforma sprzętowa

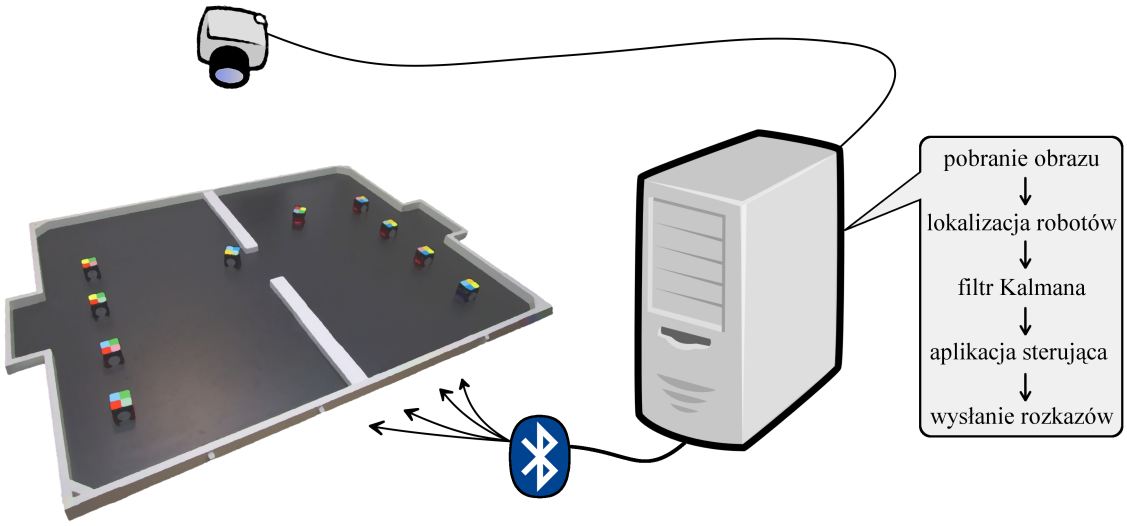
Do przeprowadzenia wszystkich opisywanych w tej pracy testów, w których wykorzystywane były rzeczywiste roboty, użyto niewielkich robotów mobilnych typu Miabot. Są one zgodne ze standardem ligi FIRA MiroSot [35], a zostały wyprodukowane przez firmę Merlin Robotics[59]. Roboty mają kształt sześcianu o boku długości 7,5 cm (rysunek 9.1). Są wyposażone w dwa, niezależnie napędzane koła, mogą rozwijać prędkość do 3,5 m/s. Funkcję jednostki sterującej pełni mikrokontroler programowalny Atmel, który nie pozwala na prowadzenie zbyt złożonych obliczeń. Każdy z robotów jest wyposażony w moduł komunikacji bezprzewodowej Bluetooth. W podstawowej wersji, która była wykorzystywana w testach, roboty nie są wyposażone w żadne sensory. Szczegółowa specyfikacja parametrów robota Miabot została przedstawiona w dodatku A.2.

Środowiskiem działania robotów jest płaski, prostokątny stół o wymiarach 220x180 cm, otoczony bandą. W środowisku ustawiane są przeszkody różnych kształtów, których roboty nie są w stanie uszkodzić ani przemieścić. Ponieważ roboty nie posiadają żadnych



Rysunek 9.1: Roboty Miabot, zgodne ze specyfikacją ligi FIRA MiroSot.

sensorów, konieczne jest zastosowanie zewnętrznego systemu lokalizacji, który jest opary o wykorzystanie kamery umieszczonej nad stołem. Każdy z robotów jest oznaczony unikatowym zestawem czterech barwnych znaczników, które pozwalają na zidentyfikowanie robota, ustalenie jego pozycji oraz orientacji. Schemat działania systemu jest widoczny na rysunku 9.2.



Rysunek 9.2: Schemat systemu sterującego grupą robotów.

Wykorzystana kamera przemysłowa dostarcza około 30 klatek na sekundę, z których każda musi być przetworzona przez algorytm rozpoznawania obrazu w celu zlokalizowania robotów. Wyznaczone pozycje są następnie przetwarzane przez filtr Kalmana [48] w celu zminimalizowania błędów lokalizacji. Błędy są spowodowane szumem sensora kamery, ograniczeniami rozdzielczości oraz własnościami algorytmu rozpoznawania obrazu.

Warto zaznaczyć, że algorytm rozpoznawania zwraca wyniki sprzed około 1/15 sekundy, podczas których robot jadący z prędkością 1 m/s pokona około 6,7 cm. Zastosowany filtr wykorzystuje historię odczytów, podane sterowanie oraz model ruchu robota w celu wyznaczenia najbardziej prawdopodobnej, aktualnej pozycji robotów. Ostateczne wyniki działania algorytmu lokalizacji są przekazywane do aplikacji sterującej, która oblicza sterowanie dla wszystkich robotów i wysyła decyzję (sterowanie prędkościami obrotowymi każdego z kół) do robotów.

9.1.2 System symulacji robotów

Dostępna platforma sprzętowa pozwala na prowadzenie eksperymentów z wykorzystaniem maksymalnie dziesięciu robotów. Służyła ona do zweryfikowania poprawności różnych podejść i algorytmów w zastosowaniu do rzeczywistych robotów. Wszystkie opisywane algorytmy nawigacji i koordynacji zostały z powodzeniem uruchomione i przetestowane na opisanej platformie.

Do przeprowadzenia testów z większą liczbą robotów wykorzystano system symulacyjny **RoBOSS2** [78], który pozwala na modelowanie i symulowanie działania różnego typu robotów, takich jak ramiona przemysłowe, roboty kołowe czy kroczące. Definicja modelu symulacyjnego składa się z części opisujących statyczne środowisko oraz działające w nim roboty. Każdy robot jest modelowany jako układ połączonych brył sztywnych, wyposażony w efektory i sensory. Definicja robota składa się z:

- Zbioru brył sztywnych w przestrzeni trójwymiarowej. Każda bryła jest charakteryzowana przez:
  - masę, opisywaną wartością, wielkością oraz kształtem (kula, prostopadłościan, walec),
  - kształt, służący do wykrywania kolizji, który może być złożeniem brył podstawowych (kula, prostopadłościan, walec) lub siatkątrójkontów,
  - współczynnik tarcia powierzchni,
  - współczynnik sprężystości.
- Połączeń pomiędzy bryłami, które ograniczają liczbę stopni swobody ruchu brył względem siebie:
  - zawias – pozostawia możliwość obrotu wokół jednej osi,
  - podwójny zawias – pozostawia możliwość obrotu wokół dwóch osi,
  - kula-gniazdo – pozostawia możliwość obrotu wokół ustalonego punktu,
  - siłownik – umożliwia przemieszczanie obiektów wzdłuż jednej osi.

- Efektorów, które przykładają siłę lub moment siły do połączonych brył. Tym sposobem można definiować wszelkiego rodzaju silniki, serwomechanizmy i siłowniki.
- Sensorow mierzących własności stanu symulowanego modelu:
  - dalmierz wykrywający najbliższą przeszkodę w zadanym kierunku,
  - odometr, mierzący drogę pokonaną przez robota kołowego,
  - gps, który zwraca zakłóconą pozycję robota,
  - żyrokompas dostarczający informacji o orientacji robota w przestrzeni.

System RoBOSS wykorzystuje bibliotekę Open Dynamics Engine [71, 72], która pozwala na symulowanie układów brył sztywnych w przestrzeni trójwymiarowej. Głównymi zaletami biblioteki ODE są:

- licencja typu open-source,
- bogaty zbiór obsługiwanych kształtów podstawowych i połączeń,
- dostępność dwóch różnych algorytmów obliczania zmian stanu modelu – dokładny o złożoności kwadratowej oraz przybliżony o złożoności liniowej,
- możliwość wpływania na dokładność obliczeń poprzez modyfikację długości kroku czasowego.

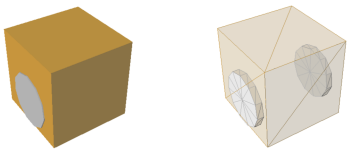
RoBOSS udostępnia bibliotekę komunikacyjną, która pozwala na szybkie tworzenie aplikacji sterującymi pojedynczymi robotami lub grupami robotów. Definiuje ona dwa interfejsy programowe, które pozwalają na sterowanie procesem symulacji i działaniem robotów. Programy sterujące symulowanymi robotami można tworzyć w językach C#, Managed C++, J# lub Visual Basic, czyli w dowolnym z języków platformy .NET 1.1.

Jedną z ważniejszych właściwości systemu RoBOSS jest możliwość zrównoleglania obliczeń symulacyjnych. Ponieważ RoBOSS jest systemem rozproszonym, działającym w sieci, poszczególne jego komponenty mogą być uruchamiane na różnych komputerach. System potrafi wykorzystać wiele jednostek obliczeniowych realizujących symulację dynamiki brył, dzięki czemu pozwala na bardzo wydajną i dokładną symulację. Jest to bardzo ważna własność, kluczowa z punktu widzenia badań nad sterowaniem licznymi grupami robotów.

Wykorzystywany w badaniach model robota typu Miabot został w systemie RoBOSS odwzorowany szczególnie dokładnie. Wzięto pod uwagę właściwości fizyczne oraz dynamiczne, wymiary, masę, przyspieszenia, opóźnienia, maksymalne prędkości, współczynniki tarcia kół oraz korpusu ślizgającego się po podłożu.

Interfejs programowy systemu RoBOSS umożliwia sterowanie wszystkimi rodzajami efektorów, jakie system modeluje i symuluje. W celu ułatwienia prowadzenia testów, został on wzbogacony o warstwę dedykowaną do sterowania robotami typu FIRA Mirosot. Dzięki temu możliwe jest uruchamianie dokładnie tych samych programów sterujących, zarówno na rzeczywistych robotach, jak i na symulowanym modelu.

Model symulacyjny robotów stosowanych w testach jest widoczny na rysunku 9.3.



Rysunek 9.3: Wizualizacja modelu symulacyjnego dwóch robotów typu FIRA Miabot. Po lewej: obraz nieprzezroczysty; po prawej: półprzezroczysta wizualizacja siatek trójkątów modelujących poszczególne części robota.

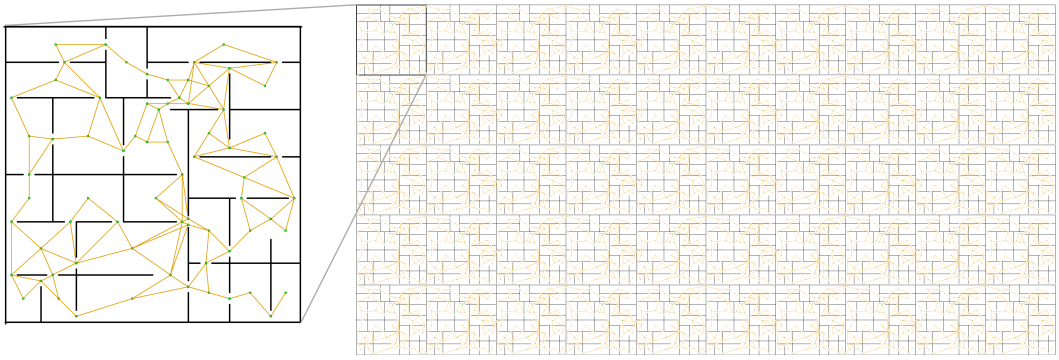
W badaniach wykorzystano wiele różnych modeli środowisk działania robotów. Najczęściej były to różnego rodzaju labirynty, różniące się wielkością i rozmieszczeniem ścian. W następnym rozdziale zaprezentowane zostaną szczegóły przeprowadzonych eksperymentów oraz uzyskane wyniki.

9.2 Wydajność algorytmu planowania tras dla robotów

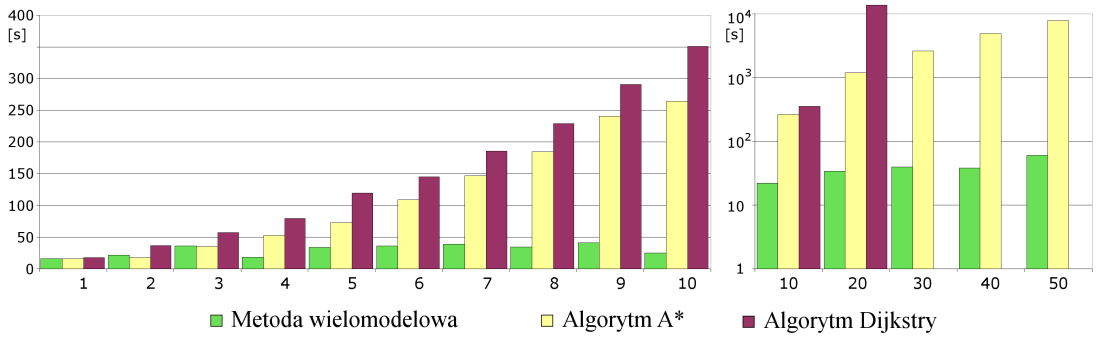
Zilustrowanie problemu wydajności wyznaczania trasy w złożonych środowiskach wymagało przeprowadzenia pomiarów czasu wyszukiwania najkrótszej ścieżki w grafach o różnym stopniu skomplikowania. Podstawowy graf, widoczny po lewej stronie rysunku 9.4, reprezentował sieć połączeń pomiędzy kilkoma przykładowymi pomieszczeniami i składał się ze 116 wierzchołków i 332 krawędzi. Testy przeprowadzono dla środowisk złożonych z 1 do 50 takich grafów, łącząc sąsiadujące segmenty jednym lub dwoma przejściami. Największy graf składał się z 5798 wierzchołków i 16812 krawędzi.

Dla każdego rozważanego grafu przeprowadzono pomiar czasu wyznaczania najkrótszej ścieżki pomiędzy tysiącem par losowo wybranych wierzchołków. Zastosowano algorytm A* z heurystyką opartą na euklidesowej odległości lokalizacji wierzchołków oraz algorytm Dijkstry zmodyfikowany tak, by kończył działanie po zlokalizowaniu najkrótszej ścieżki do wierzchołka docelowego (oryginalna wersja wyznacza najkrótsze ścieżki do wszystkich wierzchołków w grafie). Obliczenia przeprowadzono z wykorzystaniem współczesnego komputera klasy PC (dodatek A.1), a wyniki prezentujące wartości średnie z dziesięciu przeprowadzonych testów, są na rysunku 9.5.

W przypadku wyznaczania trasy dla jednego robota w niewielkim grafie zarówno al-



Rysunek 9.4: Graf wykorzystany podczas testów wydajności różnych algorytmów wyznaczania najkrótszej ścieżki.



Rysunek 9.5: Średni czas wyznaczania 1000 losowych ścieżek w zależności od wielkości grafu. Dla grafów o liczbie segmentów większej niż 20 nie udało się ukończyć obliczeń algorytmem Dijkstra.

gorytm Dijkstry jak i algorytm A* dają wyniki w zadowalającym czasie – kilka do kilkudziesięciu milisekund jest wartością akceptowalną w większości zastosowań. W większych środowiskach wyznaczenie długich tras za pomocą algorytmu Dijkstry może zajmować bardzo dużo czasu, nawet kilkadziesiąt sekund. Dla grafów większych niż 20 segmentów nie zakończono eksperymentu z tysiącem ścieżek. Zastosowanie algorytmu A* w przypadku scentralizowanego wyznaczania tras dla wielu robotów również będzie niewystarczająco wydajne. Dla największego testowanego środowiska średni czas działania algorytmu A* był o dwa rzędy wielkości większy od czasu działania metody wielomodelowej.

9.3 Badania własności podstawowego systemu zarządzającego ruchem robotów

Podstawowa wersja systemu zarządzającego ruchem składa się z agentów RWA sterujących poszczególnymi robotami, agentów MSA udostępniających usługę planowania trasy oraz

agentów TExMA przydzielających robotom kolejne punkty docelowe. Podczas działania systemu agent TExMA wyznacza losowe cele dla każdego robota, który skończył poprzednie zadanie przemieszczania. W tym celu prosi MSA o wytyczenie trasy i przekazuje jej kolejne punkty pośrednie do odpowiedniego RWA. Jednocześnie obserwuje postępy w wykonaniu tras przez wszystkie podległe RWA i mierzy skuteczność działania podległych robotów.

Miarą skuteczności działania każdego z robotów (s_i dla i-tego robota) jest stosunek długości pokonanej trasy do maksymalnego dystansu, jaki robot teoretycznie mógłby pokonać w jednostce czasu. Sposób obliczania skuteczności działania robota prezentuje równanie 9.1.

$$s_i = \frac{d_i}{V_{max} * t} \tag{9.1}$$

gdzie:
t – czas trwania pomiaru,
V_{max} – maksymalna prędkość robota,
d_i – dystans pokonany przez robota i w czasie t.

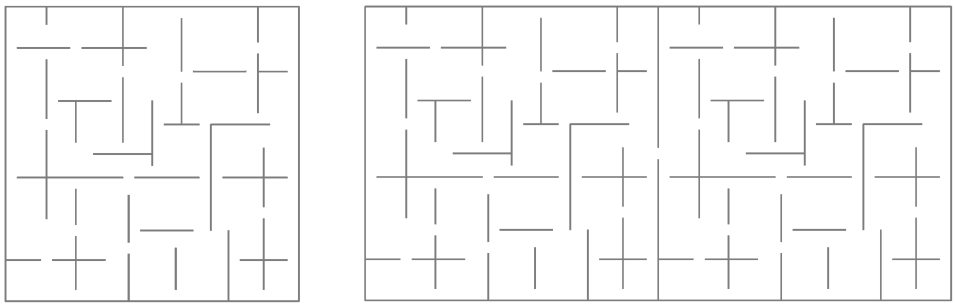
Skuteczność działania całego systemu S, złożonego za n robotów, jest mierzona jako średnia skuteczność działania wszystkich robotów (równanie 9.2).

$$S = \frac{\sum_{i=1}^{n} d_i}{V_{max} * t * n} \tag{9.2}$$

Tak zdefiniowana wartość jest zawsze liczbą rzeczywistą z przedziału $[0, 1]$. Oczywiście nigdy nie osiągnie ona wartości równej 1, ponieważ poruszający się w skomplikowanym środowisku robot nie jest w stanie utrzymać maksymalnej prędkości przez cały czas przemieszczania. Jeżeli robot musi omijać liczne przeszkody lub utknie w grupie innych robotów, skuteczność jego działania będzie gwałtownie spadać.

Wartość skuteczności, widoczna w prezentowanych dalej wynikach, została obliczona dla okresu czasu długości 600 sekund, poczynając od setnej sekundy trwania eksperymentu. Opóźnienie w rozpoczęciu pomiaru jest konieczne, by ograniczyć wpływ początkowej, arbitralnie zadanej konfiguracji robotów w środowisku.

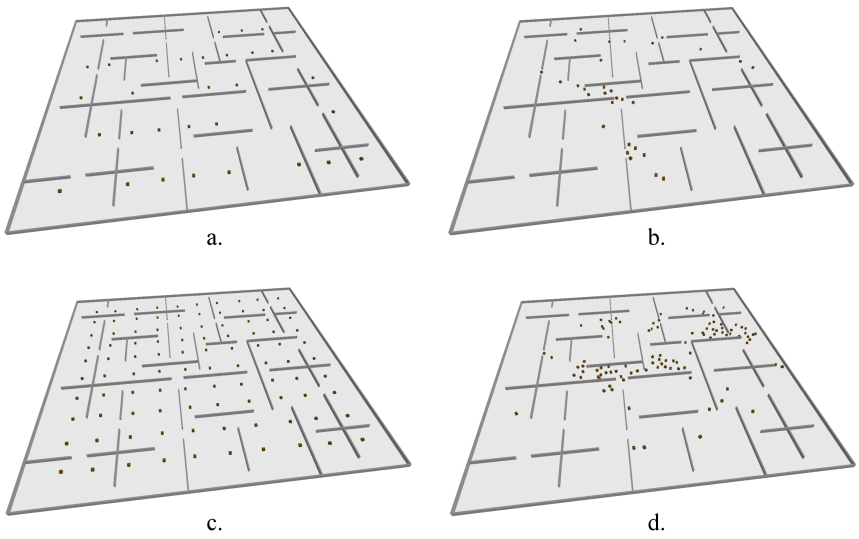
W celu zbadania skuteczności działania podstawowej wersji systemu, przeprowadzono eksperyment z wykorzystaniem narzędzia symulacyjnego RoBOSS. Badania wymagały wykorzystania dwóch różnych środowisk działania robotów. Pierwsze środowisko, którego schemat jest widoczny na rysunku 9.6a, zostało przez algorytm budowania mapy podzielone na 30 pokoi. Do każdego z pokoi prowadzi kilka wejść, więc pomiędzy każdą parą pokoi istnieje wiele różnych tras. Drugie wykorzystane w badaniach środowisko, widoczne na rysunku 9.6b, zostało skonstruowane poprzez połączenie dwóch kopii pierwszego środowiska pojedynczym przejściem. Wszystkie trasy pomiędzy lewą i prawą częścią tego środowiska muszą prowadzić przez to przejście.



Rysunek 9.6: Schemat układu pomieszczeń w środowiskach wykorzystanych podczas badania własności podstawowej wersji systemu.

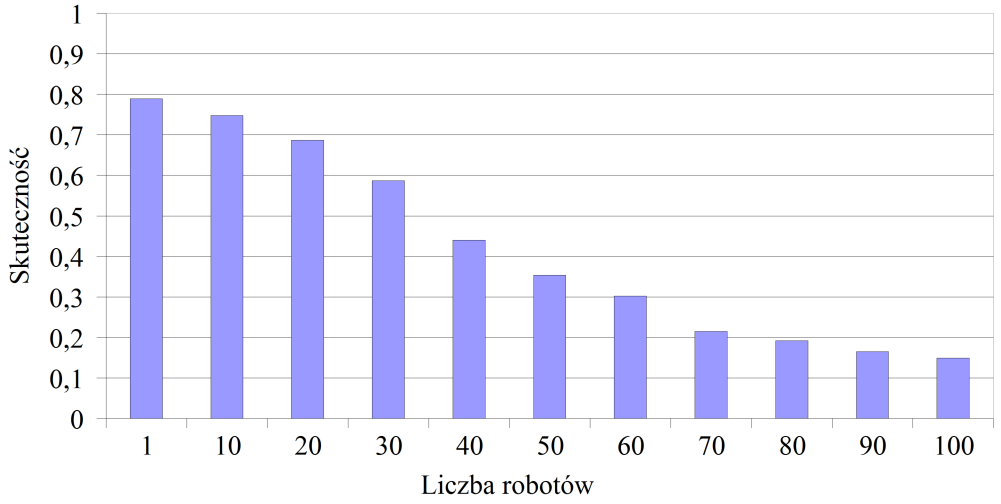
Przeprowadzone testy miały za zadanie zbadanie zależności pomiędzy liczbą robotów poruszających się w środowisku, a skutecznością działania systemu. W tym celu przeprowadzono serie pomiarów skuteczności dla każdego z przedstawionych środowisk. Każda seria składała się z jedenastu testów dla grup robotów o różnej liczebności.

W każdym ze środowisk przeprowadzono pomiar skuteczności działania jednego robota. Miało to na celu wyznaczenie maksymalnej, możliwej do osiągnięcia skuteczności działania. Pojedynczy robot nie napotka na żadne ruchome przeszkody, więc będzie się przemieszczał z maksymalną, możliwą do osiągnięcia w danym środowisku, prędkością średnią.



Rysunek 9.7: Wizualizacja układu robotów w środowisku złożonym z 30 pokoi dla grupy 30 (a. i b.) oraz 100 (c. i d.) robotów. Po lewej: stan tuż po uruchomieniu testów; po prawej: stan po stu sekundach działania systemu.

Na rysunku 9.7 widoczne są wizualizacje układu robotów podczas testów z wykorzystaniem pierwszego środowiska. Sytuacje **a.** i **b.** prezentują test grupy 30 robotów, a sytuacje **c.** i **d.** grupy 100 robotów. Po lewej stronie widoczne są stany początkowe systemu, a po prawej stany po 100 sekundach trwania eksperymentu. Łatwo można zauważyć tendencję do gromadzenia się robotów w okolicy wąskich przejść, szczególnie w centralnej części mapy. Roboty, które tworzą zatory, tracą czas na omijanie licznych ruchomych przeszkód i nie pokonują dostatecznie szybko zaplanowanej trasy. To zjawisko powoduje bardzo znaczący spadek skuteczności działania systemu wraz ze wzrostem liczby robotów.



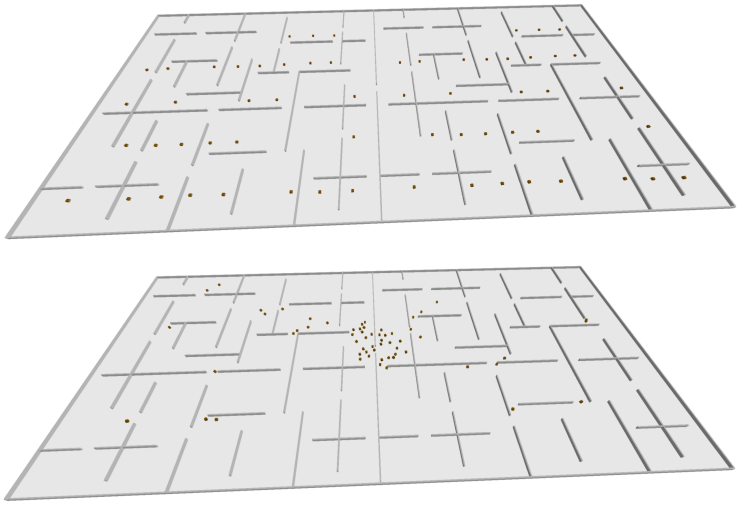
Rysunek 9.8: Skuteczność działania podstawowej wersji systemu w zależności od liczby robotów w środowisku złożonym z 30 pokoi.

Zestawienie wyników badań skuteczności działania systemu, w zależności od liczby robotów, jest widoczne na rysunku 9.8.

Maksymalna wartość skuteczności, zmierzona dla pojedynczego robota, wyniosła 0,789. Można zauważyć, że dla niezbyt licznych grup robotów, skuteczność jest bliska wartości maksymalnej, czyli każdy z robotów prawie nieprzerwanie podąża do celu. Wraz ze wzrostem liczby robotów ponad 30, skuteczność gwałtownie spada, co jest spowodowane gromadzeniem się robotów w okolicy wąskich przejść w centralnej części mapy. Powstający zator robotów, które wzajemnie sobie przeszkadzają, bardzo znacząco ogranicza skuteczność. Warto jednak zaznaczyć, że nawet dla bardzo licznych grup robotów, skuteczność ciągle jest większa od zera, czyli działanie systemu nie zostało całkowicie wstrzymane.

Analogiczne testy zostały przeprowadzone dla drugiego rozważanego środowiska, którego schemat jest widoczny na rysunku 9.6b. Ponieważ badane środowisko jest dwukrotnie większe od poprzedniego, podczas testów stosowano dwukrotnie bardziej liczne grupy robo-

tów. Wizualizacje układu robotów podczas przeprowadzonych eksperymentów są widoczne na rysunku 9.9.



Rysunek 9.9: Wizualizacja układu robotów w środowisku złożonym z 60 pokoi dla grupy 60 robotów. Na górze: stan tuż po uruchomieniu testów; na dole: stan po stu sekundach działania systemu.

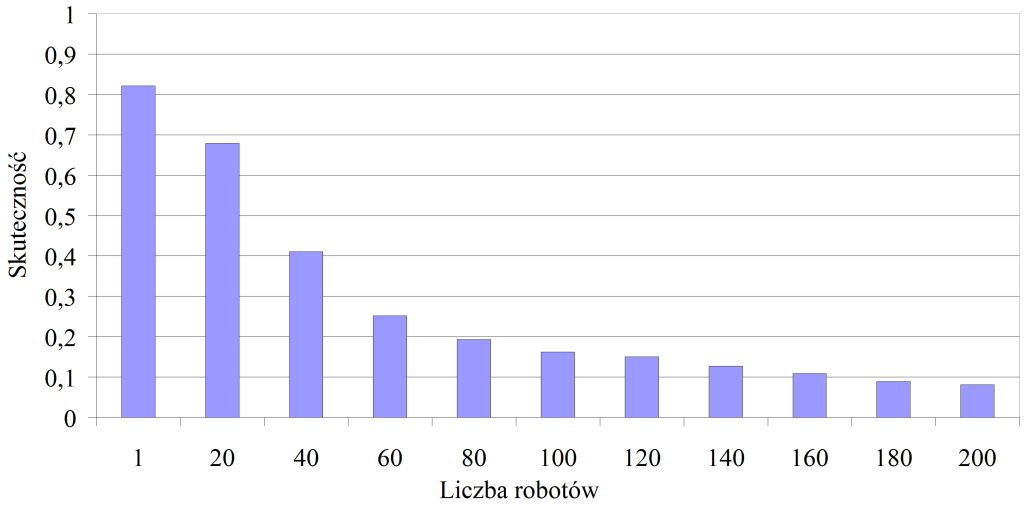
Wąskie przejście, znajdujące się w centralnej części tego środowiska, jest jedynym połączeniem pomiędzy prawą i lewą częścią środowiska. Wszystkie roboty, które mają za zadanie przemieścić się pomiędzy częściami, będą musiały pokonać to przejście. Z tego powodu bardzo szybko dochodzi do nagromadzenia robotów w rejonie tego przejścia. Nawet niezbyt liczna grupa 60 robotów szybko tworzy zator.

Zestawienie wyników badań skuteczności działania systemu, w zależności od liczby robotów, jest widoczne na rysunku 9.10.

Maksymalna wartość skuteczności, zmierzona dla pojedynczego robota działającego w tym środowisku, wyniosła 0,821. Nieznaczny wzrost skuteczności względem testów mniejszego środowiska jest spowodowany tym, że statystyczna trasa robota jest w tym przypadku znacznie dłuższa, więc rzadziej występuje konieczność jej planowania. Robot poświęca nieco więcej czasu na przemieszczanie i nie marnuje go w oczekiwaniu na plan trasy.

Umieszczenie w testowanym środowisku grupy kilkudziesięciu robotów powoduje bardzo wyraźny spadek skuteczności. Pomimo dwukrotnie większej przestrzeni działania robotów, wyniki dla grup o liczebności do 100 robotów są bardzo podobne do uzyskanych w mniejszym środowisku.

Zaprezentowane badania podstawowej wersji systemu pozwalają stwierdzić, że pozwala on na bezpieczne i skuteczne zarządzanie ruchem niezbyt licznych grup robotów w zło-



Rysunek 9.10: Skuteczność działania podstawowej wersji systemu w zależności od liczby robotów w środowisku złożonym z 60 pokoi.

żonych środowiskach. Charakteryzuje się on skalowalnością ze względu na rozmiar środowiska, ponieważ umożliwia łatwe zwiększanie przestrzeni działania robotów. Skuteczność działania systemu w powiększonym środowisku jest zbliżona do skuteczności działania w środowisku wyjściowym.

Podstawowa wersja systemu nie jest natomiast skalowalna ze względu na liczbę robotów działających w środowisku. Zwiększanie liczby robotów powoduje bardzo wyraźny spadek skuteczności działania systemu, co nie jest własnością akceptowalną.

9.4 Wpływ algorytmu równoważenia obciążenia krawędzi grafu na skuteczność systemu

Główną przyczyną gwałtownego spadku skuteczności działania systemu wraz ze wzrostem liczby robotów jest zjawisko gromadzenia się wielu robotów w okolicy wąskich przejść centralnej części środowiska. Algorytm planowania tras zawsze znajduje najkrótszą ścieżkę pomiędzy zadanymi punktami, więc największa liczba tras przebiega przez środkową część mapy. Roboty zgromadzone na małym obszarze wzajemnie uniemożliwiają sobie dotarcie do celu, ponieważ każdy z nich stosuje jedynie nawigację reaktywną.

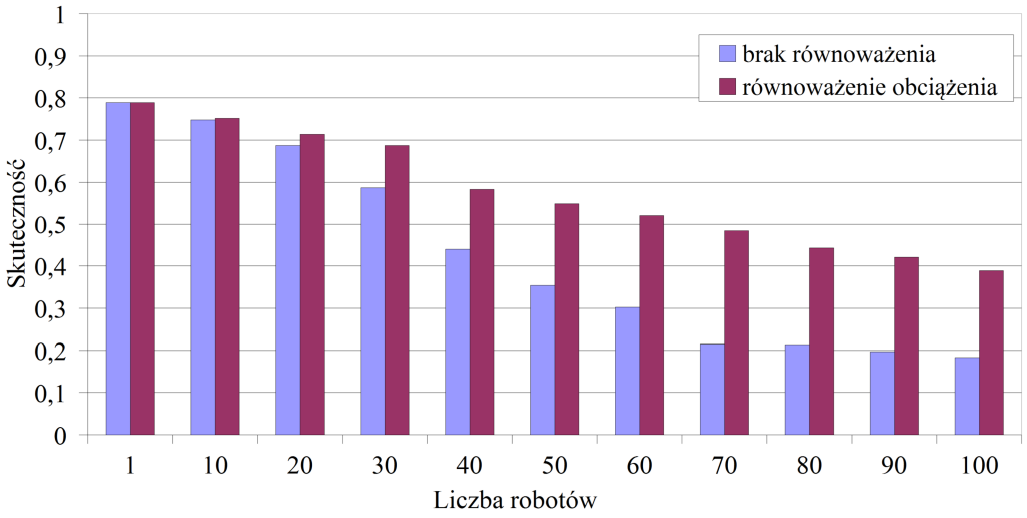
Rozwiązaniem tego problemu może być algorytm równoważenia obciążenia, który został opisany w rozdziale 8.6. Odpowiednie modyfikowanie wag krawędzi grafu powinno powodować przekierowywanie robotów na alternatywne drogi dotarcia do celu.

W celu zbadania wpływu algorytmu równoważenia obciążenia na skuteczność działania systemu przeprowadzono serię eksperymentów analogicznych do opisanych w poprzednim

podrozdziale. Wykorzystano te same środowiska i tak samo liczne grupy robotów, sterowane za pomocą tego samego algorytmu. Jedyną wprowadzoną modyfikacją był sposób wyznaczania tras przez agenta MSA.

Algorytm równoważenia obciążenia wymaga zdefiniowania współczynnika skalującego, który jest wykorzystywany podczas modyfikowania wag krawędzi grafu. Testy wykazały, że konkretna wartość współczynnika nie ma kluczowego znaczenia. Dla różnych wartości $0,05 < g < 1$ efekty były bardzo podobne.

Pierwsza grupa testów została wykonana z wykorzystaniem środowiska składającego się z 30 pokoi (model widoczny na rysunku 9.6a). Porównanie skuteczności działania testowanego systemu bez i z zastosowaniem algorytmu równoważenia obciążenia prezentuje rysunek 9.11.



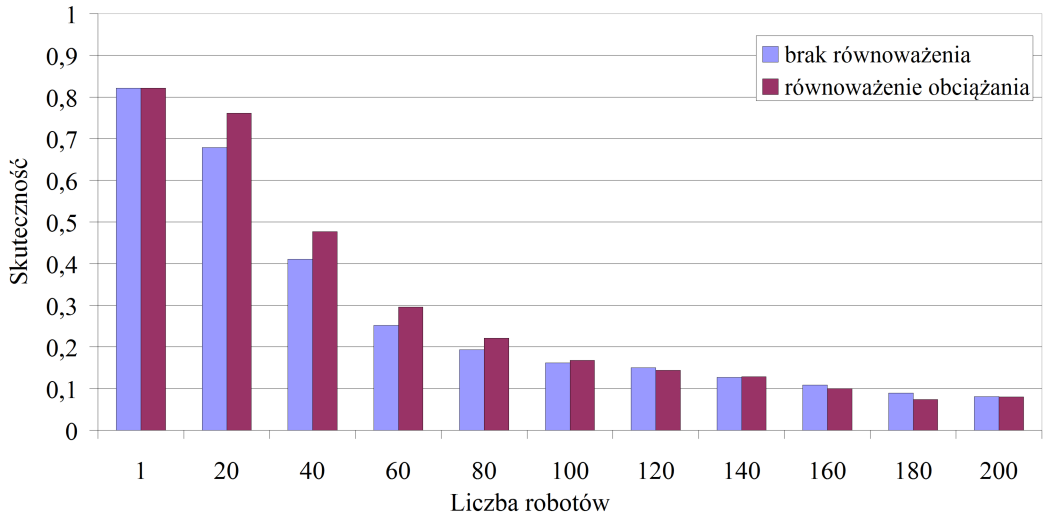
Rysunek 9.11: Porównanie skuteczności działania systemu z uruchomionym algorytmem równoważenia obciążenia i bez niego dla środowiska składającego się z 30 pokoi.

Pozytywny wpływ algorytmu równoważenia obciążenia na skuteczność działania systemu jest widoczny już dla bardzo małych grup. Dla grup składających się z ponad 70 robotów poprawa skuteczności była ponad dwukrotna. Zastosowanie algorytmu pozwoliło na utrzymanie płynności ruchu w całym środowisku i prawie całkowicie zlikwidowało zjawisko powstawania zatorów w okolicach wąskich przejść.

Należy pamiętać, że algorytm równoważenia obciążenia jest w stanie poprawić skuteczność działania systemu jedynie w sytuacji, gdy w środowisku istnieje wiele alternatywnych ścieżek pomiędzy różnymi punktami. W przypadku, gdy grafowy model środowiska jest 1-spójny, czyli występuje w nim jeden wierzchołek, którego usunięcie dzieli graf na dwie niepołączone części, zmiana wag krawędzi w zależności od rozmieszczenia robotów nie da pożądanych efektów. W takim grafie wszystkie trasy pomiędzy dwoma częściami muszą

przechodzić przez ten sam wierzchołek, niezależnie od wag krawędzi.

Dobrą ilustracją tej własności są wyniki pomiarów skuteczności działania badanego systemu w zastosowaniu do środowiska złożonego z 60 pokoi, którego model jest widoczny na rysunku 9.6b. Przez wąskie przejście, znajdujące się w centralnej części środowiska, przechodzić będą wszystkie ścieżki łączące lewi prawą cześć środowiska. Rysunek 9.12 prezentuje porównanie skuteczności bez i z zastosowaniem algorytmu równoważenia obciążenia.



Rysunek 9.12: Porównanie skuteczności działania systemu z uruchomionym algorytmem równoważenia obciążenia i bez niego dla środowiska składającego się z 60 pokoi.

Skuteczność działania systemu w tym środowisku nie ulega poprawie w przypadku zastosowania algorytmu równoważenia obciążenia. Zator w środkowej części środowiska powoduje gwałtowny spadek skuteczności niezależnie od tego, czy algorytm równoważący jest uruchomiony, czy też nie.

Można więc stwierdzić, że dla pewnych rodzajów środowisk podstawowa wersja systemu, wzbogacona o algorytm równoważenia obciążenia, jest skalowalna ze względu na liczbę robotów. Będą to środowiska, których model grafowy charakteryzuje się wysokim stopniem spójności wierzchołkowej.

Osiągnięcie poprawy skuteczności działania systemu dla pozostałych środowisk będzie wymagało zastosowania algorytmów koordynacji ruchu, które usprawnią przemieszczanie się robotów przez najbardziej obciążone fragmenty środowiska.

9.5 Porównanie zrealizowanych algorytmów koordynacji ruchu

Algorytmy implementowane przez agenty AMA, które zarządzają ruchem w poszczególnych obszarach, są kluczowym elementem zrealizowanego systemu nawigacji robotów. W

tym rozdziale przedstawione zostanie porównanie właściwości i wydajności zaimplementowanych metod koordynacji, które zostały szczegółowo opisane w rozdziale 8.7:

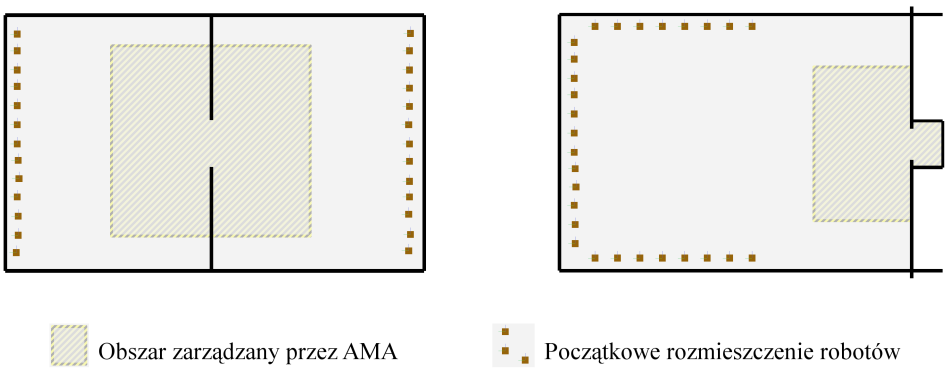
- metoda zarządzania kolejnością przemieszczania robotów,
- metoda zdalnego, centralnego planowania i sterowania ruchem robotów.

Jako punkt odniesienia dla metod koordynujących ruch, wykorzystane zostanie rozwiązanie, w którym **nie** występuje agent obszaru, nie jest stosowana żadna metoda koordynująca zachowania robotów. Podczas pokonywania obszaru każdy z agentów sterujących poszczególnymi robotami wykorzystuje algorytm autonomicznej nawigacji reaktywnej, który został przedstawiony w rozdziale 8.4.

W celu wykonania badań wydajności wykorzystane zostaną dwa problemy, wymagające koordynacji ruchu robotów:

- kontrola ruchu w wąskim przejściu,
- zarządzanie dostępem robotów do określonej lokalizacji.

Są to typowe problemy występujące w większości środowisk działania robotów mobilnych. Wąskie przejście jest abstrakcyjnym modelem wszelkiego rodzaju połączeń pomiędzy pomieszczeniami, do których roboty mogą jednocześnie zbliżać się z obu stron. Zagadnienie dostępu do określonej lokalizacji może występować w miejscach pobierania lub oddawania transportowanych przedmiotów, przy wejściu do windy czy punkcie wymiany akumulatorów. Schematy środowisk wykorzystanych podczas testów widoczne są na rysunku 9.13.



Rysunek 9.13: Schemat kształtów środowiska i początkowego rozmieszczenia robotów wykorzystywanych w testach metod koordynacji ruchu. Środowisko po lewej stronie modeluje problem wąskiego przejścia, po prawej problem dostępu do określonej lokalizacji.

W przypadku problemu wąskiego przejścia (model środowiska na rysunku 9.13 po lewej) zadaniem każdego z robotów było osiągnięcie zadanej pozycji po przeciwległej stronie przejścia. Po osiągnięciu tej pozycji każdy z robotów miał powrócić do pozycji wyjściowej, czyli w ramach jednego testu musiał pokonać wąskie przejście dwukrotnie. Wszystkie roboty rozpoczynały ruch jednocześnie.

Celem każdego z robotów w problemie dostępu do określonej lokalizacji (model środowiska na rysunku 9.13 po prawej) było wjechanie do niewielkiego pomieszczenia, znajdującego się na środku prawej granicy środowiska. Po osiągnięciu celu, każdy z robotów musiał powrócić do początkowej lokalizacji. Również w tym przepadku wszystkie roboty, biorące udział w teście, rozpoczynały ruch jednocześnie.

Przyjętą miarą jakości metody koordynacji jest czas wykonania całego testu, tzn. czas od rozpoczęcia ruchu przez pierwszego robota do powrotu wszystkich robotów na wyjściową pozycję. Warunkiem poprawności wykonania testu jest brak jakichkolwiek kolizji pomiędzy robotami.

Testy poprawności wszystkich metod koordynacji przeprowadzono zarówno na rzeczywistych robotach, jak i w symulatorze RoBOSS. Prezentowane w dalszej części tego rozdziału wyniki zostały uzyskane w symulacji, ponieważ liczba dostępnych robotów była niewystarczająca.

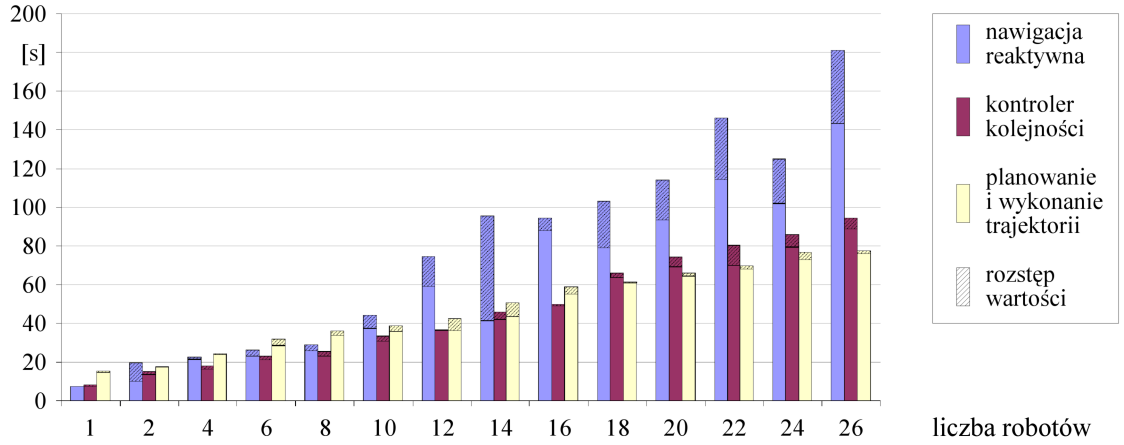
9.5.1 Problem organizacji ruchu w wąskim przejściu

Testy poszczególnych metod zostały przeprowadzone dla grup liczących od 1 do 26 robotów, umieszczonych symetrycznie po obu stronach wąskiego przejścia. Zadaniem każdego z robotów było przemieszczenie się na przeciwległą stronę przejścia i powrót do pozycji wyjściowej. Wszystkie roboty były uruchamiane jednocześnie, a maksymalna prędkość wynosiła 0,5 m/s. Pozostałe parametry wszystkich algorytmów zostały dobrane tak, by zminimalizować czas wykonania testu przy jednoczesnym zagwarantowaniu braku występowania kolizji. Każdy z testów został wykonany pięciokrotnie.

Rysunek 9.14 prezentuje zestawienie czasów wykonania zadania koordynacji w zależności od liczby robotów. Zgodnie z przewidywaniami, metoda autonomicznej nawigacji reaktywnej sprawdza się dobrze tylko dla niezbyt licznych grup robotów. Gdy liczebność grupy przekracza 10, czas wykonania zadania szybko rośnie i staje się nieprzewidywalny. Algorytmy kontrolera kolejności oraz planowania i wykonania trajektorii dają zbliżone wyniki i dobrze się skalują wraz ze wzrostem liczby robotów. Poniżej prezentowany jest szczegółowy opis działania poszczególnych metod.

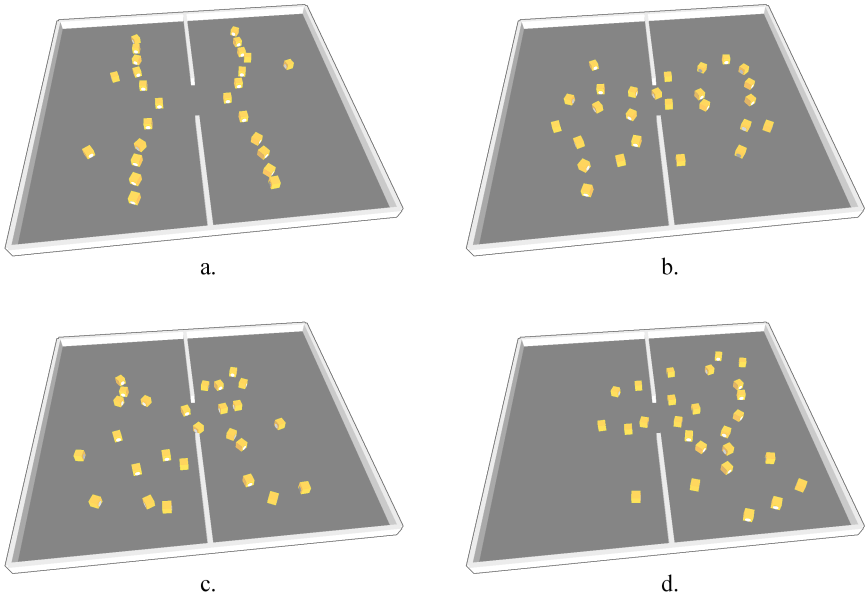
Algorytm autonomicznej nawigacji reaktywnej

Rysunek 9.15 prezentuje stan systemu w 2, 5, 15 i 30 sekundzie eksperymentu, w którym każdy z robotów był sterowany za pomocą algorytmu autonomicznej nawigacji reaktywnej.



Rysunek 9.14: Zestawienie wyników pomiarów czasu wykonania zadania koordynacji ruchu w wąskim przejściu przy pomocy różnych algorytmów.

By zminimalizować wpływ cech algorytmu reaktywnego, trasa każdego z robotów została wzbogacona o punkt pośredni znajdujący się w środku wąskiego przejścia.



Rysunek 9.15: Kolejne stany systemu, w którym każdy z robotów stosuje wyłącznie algorytm autonomicznej nawigacji reaktywnej. Sytuacja **a.**: 2 sekunda, **b.**: 5 sekunda, **c.**: 15 sekunda, **d.**: 30 sekunda eksperymentu.

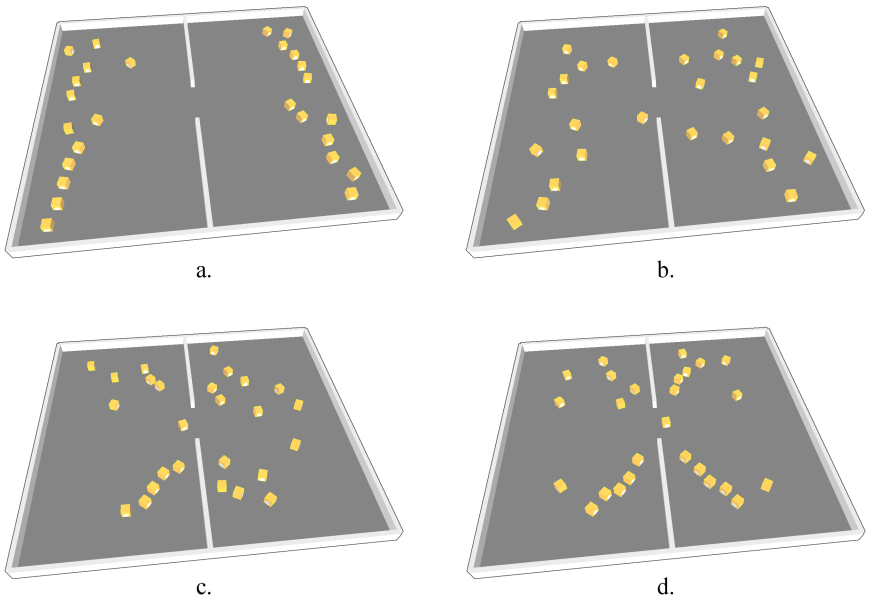
W sytuacji **a.** prawie każdy robot podąża prosto do przejścia – wzajemne ustawienie nie powoduje jeszcze wykrycia przeszkód na trasie. Jedynie cztery roboty (2 z każdej strony)

wykryły już innego robota na swojej trasie i zmodyfikowały swój cel, by uniknąć kolizji. W późniejszych sytuacjach (**b.**, **c.** oraz **d.**) ruch robotów staje się chaotyczny. Każdy robot próbuje przedostać się przez przejście, co może udać się jedynie w sytuacji, gdy z drugiej strony przejścia nie będzie przez chwilę żadnego robota. Obserwując takie zachowanie robotów wydawać by się mogło, że wykonanie zadania zajmie bardzo dużo czasu. Okazuje się jednak, że stosunkowo szybkie i dynamiczne ruchy pozwalają poszczególnym robotom na pokonanie przejścia. W miarę ubywania robotów krążących w centralnej części środowiska, coraz łatwiej jest wykonać zadanie.

Największą wadą takiego podejścia jest brak możliwości oszacowania czasu wykonania zadania. Rozstęp wartości czasu wykonania zadania rośnie znacząco dla grup większych niż 10 robotów. Średni czas wykonania również jest znacznie gorszy od metod zarządzających współdziałaniem licznej grupy. Warto jednak pamiętać, że jest to podejście bardzo łatwe w realizacji i nie wymagające komunikacji pomiędzy robotami. Dla nielicznych grup daje wyniki w czasie bardzo podobnym, co metody zarządzające.

Algorytm kontrolera kolejności przejazdu

Algorytm kontrolera kolejności, zastosowany do problemu wąskiego przejścia, definiuje cztery kolejki, po dwie z każdej strony przejścia. Na rysunku 9.16 widoczne są stany systemu po 2, 5, 15 i 30 sekundach eksperymentu.



Rysunek 9.16: Kolejne stany systemu zarządzanego przez algorytm kontrolera kolejności. Sytuacja **a.**: 2 sekunda, **b.**: 5 sekunda, **c.**: 15 sekunda, **d.**: 30 sekunda eksperymentu.

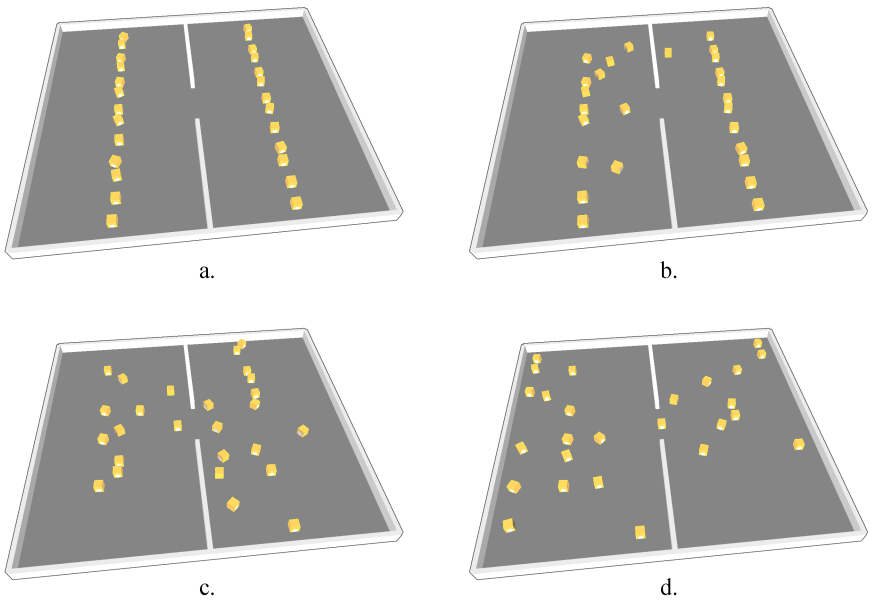
W sytuacji **a.** większość robotów porusza się bardzo wolno, ponieważ zostały przypisane

na odległe miejsca w kolejkach. Robot, który jako pierwszy będzie pokonywał przejście, znajduje się w lewej-górnej ćwiartce środowiska, najbliżej przejścia. W kolejnych sytuacjach zaobserwować można coraz bardziej wyraźne formowanie się kolejek robotów oczekujących na przejazd. W sytuacji **d.** kilka robotów zmierza już na docelowe pozycje, jeden pokonuje przejście, wszystkie pozostałe znajdują się w czterech kolejkach.

Metoda kontroli kolejności skutecznie wykonuje zadanie koordynacji w wąskim przejściu, dając przewidywalne wyniki. Jest przy tym nieskomplikowana w realizacji, ponieważ wykorzystuje algorytm nawigacji reaktywnej. Pozwala osiągnąć bardzo dobre rezultaty bez konieczności prowadzenia kosztownych obliczeń, dzięki czemu może być stosowana w wielu miejscach w złożonym środowisku, nie zwiększając znacząco wymagań sprzętowych.

Algorytm planowania i wykonania trajektorii

Na rysunku 9.17 widoczne są stany systemu zarządzanego przez algorytm planowania i wykonania trajektorii, w 2, 5, 15 i 30 sekundzie eksperymentu.



Rysunek 9.17: Kolejne stany systemu zarządzanego przez algorytm planownia i wykonania trajektorii robotów. Sytuacja **a.**: 2 sekunda, **b.**: 5 sekunda, **c.**: 15 sekunda, **d.**: 30 sekunda eksperymentu.

W sytuacji **a.** wszystkie roboty osiągnęły granicę zarządzanego obszaru – zostały umieszczone w skrajnych wierzchołkach siatki. Roboty z lewej strony znalazły się w obszarze nieco szybciej i dlatego ich trajektorie zostały zaplanowane w pierwszej kolejności. W sytuacji **b.** widać, że większość przemieszczających się robotów zbliża się do przejścia z lewej strony. W kolejnej sytuacji poszczególne roboty, ze sporą prędkością, pokonują przej-

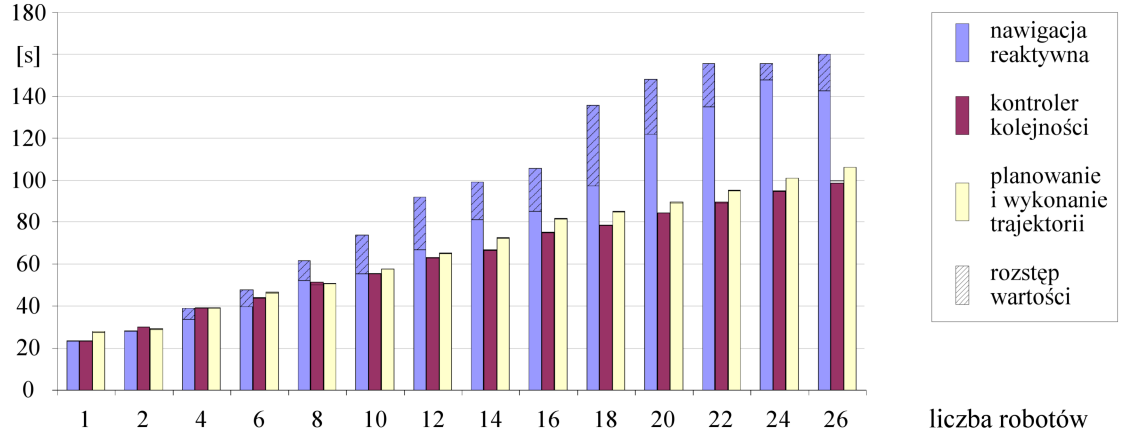
ście i zmierzają do swoich celów. W sytuacji **d.** kilka robotów zakończyło wykonywanie zadań – tu również widoczna jest przewaga robotów z lewej strony przejścia.

Zgodnie z przewidywaniami, algorytm planowania i wykonania trajektorii dał najlepsze i najbardziej stabilne wyniki dla najbardziej licznych grup robotów (rysunek 9.14). Jest on najbardziej odpowiednim rozwiązaniem bardzo trudnych zadań koordynacji, charakteryzuje się jednak również największą złożonością obliczeniową.

9.5.2 Problem zarządzania dostępem robotów do określonej lokalizacji

W podobnych warunkach przeprowadzono testy algorytmów koordynacji w zastosowaniu do problemu kontroli dostępu. W systemie symulacyjnym zdefiniowano środowisko, w którym wyróżniono niewielki obszar reprezentujący kontrolowany zasób. Zadaniem każdego z robotów było przemieszczenie się do wyznaczonej lokalizacji i powrót na wyjściową pozycję. Testy przeprowadzono dla grup od 1 do 26 robotów. Wszystkie roboty były uruchamiane jednocześnie, a ich maksymalna prędkość wynosiła 0,5 m/s. Podobnie, jak w testach problemu wąskiego przejścia, pozostałe parametry wszystkich algorytmów zostały dobrane tak, by zminimalizować czas wykonania testu przy jednoczesnym zagwarantowaniu braku występowania kolizji. Każdy z testów został wykonany pięciokrotnie.

Wyniki pomiaru czasu wykonania zadania prezentuje rysunek 9.18.

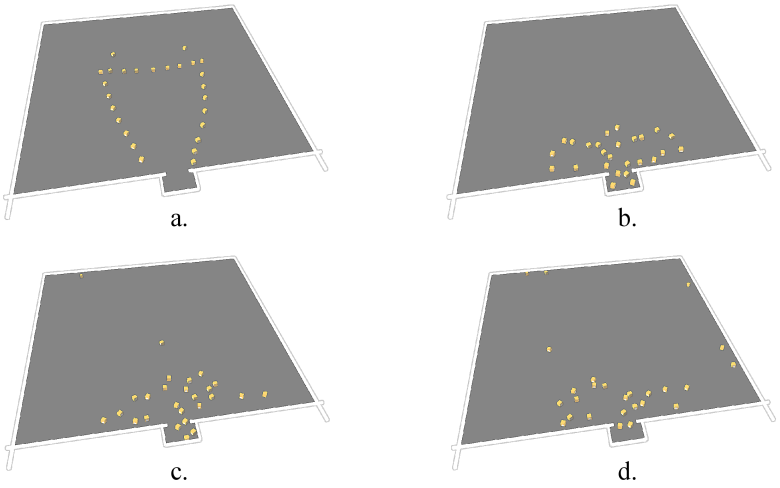


Rysunek 9.18: Zestawienie wyników pomiarów czasu wykonania zadania kontroli dostępu do zasobu przy pomocy różnych algorytmów.

Algorytm autonomicznej nawigacji reaktywnej

Podobnie, jak w przypadku problemu wąskiego przejścia, można zauważyć, że metoda autonomicznej nawigacji reaktywnej daje dobre wyniki jedynie dla nielicznych grup robotów – w przypadku opisywanego testu były to grupy mniejsze niż 10 robotów. Na rysunku

9.19 widoczne są stany systemu w 5, 10, 20 i 40 sekundzie trwania eksperymentu.



Rysunek 9.19: Kolejne stany systemu, w któym każdy z robotów stosuje wyłącznie algorytm autonomicznej nawigacji reaktywnej. Sytuacja **a.**: 5 sekunda, **b.**: 10 sekunda, **c.**: 20 sekunda, **d.**: 40 sekunda eksperymentu.

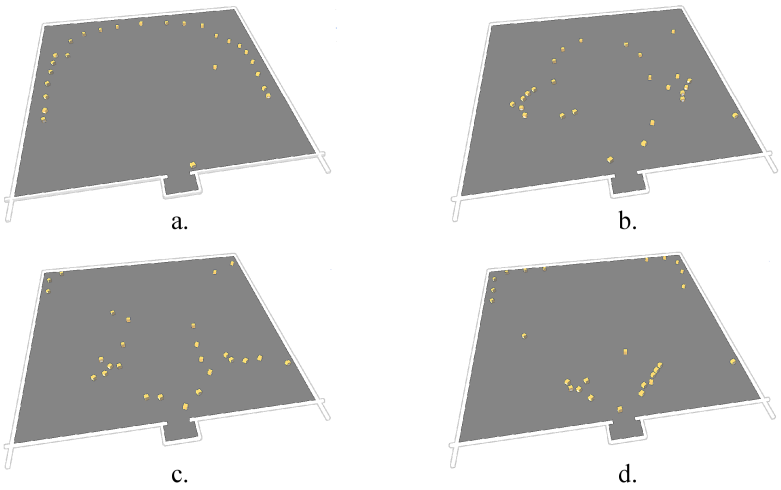
W sytuacji **a.** roboty zmierzają prosto do niewielkiego pomieszczenia, które modeluje pożądany zasób. W kolejnych sytuacjach roboty krążą w okolicy wejścia do pomieszczenia. Częstą sytuacją było długie pozostawanie tych samych robotów w pomieszczeniu, spowodowane blokowaniem wejścia przez inne roboty, które próbowały do pomieszczenia wjechać. W sytuacji **d.** można zauważyć trzy roboty, które zakończyły wykonywanie zadania. Chaotyczne ruchy robotów w okolicy wejścia do pomieszczenia powodowały znaczne rozbieżności w czasie wykonania zadania przy poszczególnych uruchomieniach eksperymentu. Jest to zdecydowanie największa wada tej metody.

Algorytm kontrolera kolejności przejazdu

Algorytm kontrolera kolejności tworzył dwie kolejki robotów oczekujących na wjazd do pomieszczenia. Rysunek 9.20 prezentuje stany systemu w 5, 10, 20 i 40 sekundzie trwania eksperymentu.

W sytuacji **a.** widoczny jest wyraźny wpływ metody redukowania prędkości robotów, które zajmować mają odległe miejsca w kolejkach. Jeden z robotów znajduje się już bardzo blisko docelowego obszaru, a pozostałe ze zredukowaną prędkością zmierzają na swoje miejsca w kolejkach.

Już po 10 sekundach eksperymentu (sytuacja **b.**) pierwszy robot powrócił na wyjściową pozycję. Kolejne roboty zaczynają w tym czasie formować zgrupowania po dwóch stronach wejścia do pomieszczenia.



Rysunek 9.20: Kolejne stany systemu zarządzanego przez algorytm kontrolera kolejności. Sytuacja **a.**: 5 sekunda, **b.**: 10 sekunda, **c.**: 20 sekunda, **d.**: 40 sekunda eksperymentu.

Znaczna odległość początkowej lokalizacji niektórych robotów od pomieszczenia, spowodowała opóźnienie w uformowaniu wyraźnych kolejek, które najlepiej widoczne są dopiero w sytuacji **d.**.

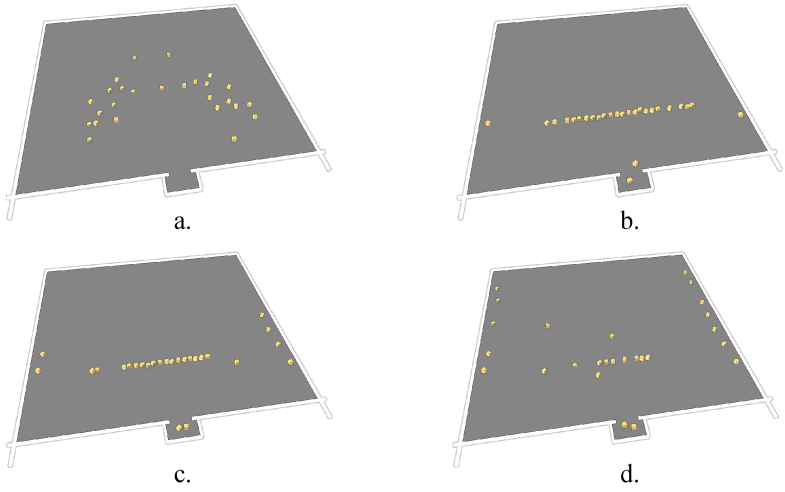
Algorytm kontrolera kolejności, w zastosowaniu do problemu kontroli dostępu, dał wyniki nieco lepsze niż algorytm planowania i wykonania trajektorii. Należy pamiętać, że charakteryzuje się on przy tym znacznie niższą złożonością obliczeniową. Rozstęp zmierzonych wartości dla obu metod był minimalny.

Algorytm planowania i wykonania trajektorii

Na rysunku 9.21 widoczne są kolejne stany systemu zarządzanego przez algorytm zdalnego planowania i wykonania trajektorii. Podobnie, jak w poprzednich przypadkach zostały one uchwycone w 5, 10, 20 i 40 sekundzie trwania eksperymentu.

Również w tej metodzie, po dziesięciu sekundach pierwsze roboty powróciły na wyjściowe pozycje. Jednak czas obsługiwania kolejnych robotów był w tej metodzie znacznie dłuższy, niż w przypadku kontrolera kolejności. Każdy z robotów miał za zadanie osiągnięcie jednego z dwóch wierzchołków, które znajdowały się wewnątrz pomieszczenia. Każda zaplanowana trajektoria tworzyła rezerwację wyłączną w węźle docelowym, która była aktywna do czasu planowania trajektorii powrotnej. Blokada uniemożliwiała planowanie i wykonywanie trajektorii dla innych robotów, co zwiększyło czas wykonania zadania.

Podsumowując przedstawione porównanie metod koordynacji ruchu należy stwierdzić, że zastosowanie metod koordynacji w wybranych fragmentach środowiska powoduje zwiększenie płynności ruchu, skrócenie średniego czasu pokonywania trasy, umożliwia oszacowa-



Rysunek 9.21: Kolejne stany systemu zarządzanego przez algorytm planownia i wykonania trajektorii robotów. Sytuacja **a.**: 5 sekunda, **b.**: 10 sekunda, **c.**: 20 sekunda, **d.**: 40 sekunda eksperymentu.

nie czasu trwania przemieszczania robota oraz ułatwia osiągnięcie bezpieczeństwa ruchu robotów. Należy jednak zaznaczyć, że stosowanie algorytmu koordynacji jest uzasadnione dopiero wtedy, gdy liczba robotów w obszarze jest odpowiednio duża. Dla pojedynczych robotów zastosowanie autonomicznej metody reaktywnej daje bardzo dobre efekty.

Przeprowadzone testy wskazują, że w najbardziej typowych obszarach, jakimi są wąskie przejścia pomiędzy pokojami, zastosowanie metody kontrolera kolejności jest w zupełności wystarczające. Ponieważ korzysta on z algorytmu nawigacji reaktywnej i jedynie modyfikuje punty docelowe i prędkości, jest łatwy w implementacji i ma niską złożoność obliczeniową.

Trzeba jednak zaznaczyć, że stosowanie kontrolera kolejności jest możliwe jedynie we fragmentach środowiska o określonych właściwościach. Algorytmiczne wyznaczenie odpowiedniego rozłożenia kolejek w dowolnie skomplikowanych obszarach nie jest łatwe, a czasem może nie być możliwe. W takich sytuacjach zastosować można metodę planowania i wykonania trajektorii, która jest uniwersalna i pozwala na koordynowanie ruchu w dowolnie skomplikowanych obszarach.

9.6 Badania własności zintegrowanego systemu nawigacji robotów mobilnych

Zintegrowany system nawigacji robotów mobilnych jest rozbudową podstawowego systemu o algorytm równoważenia obciążenia oraz o możliwość dodawania agentów AMA. Agenty AMA zarządzają ruchem we fragmentach środowiska, stosując jeden z algorytmów koor-

dynacji. Dokładny opis systemu i sposobu jego działania został przedstawiony w rozdziale 8.8.

Prezentowane w dalszej części badania miały na celu wykazanie, że zrealizowany system charakteryzuje się pożądanymi własnościami pozafunkcjonalnymi. W tym celu zaprezentowane zostaną wyniki przeprowadzonych pomiarów skuteczności działania systemu w zastosowaniu do środowisk różnej wielkości oraz grup robotów o różnej liczebności. Ponadto opisane zostaną metody rozbudowywania systemu o nowe typy robotów oraz nowe fragmenty środowiska. Przedstawione będą też wyniki badań odporności systemu na awarie poszczególnych elementów.

9.6.1 Skalowalność systemu

Skalowalność systemu jest rozumiana jako zdolność do przetwarzania większej ilości zadań ustalonego rodzaju. W przypadku badanego systemu zarządzającego ruchem robotów wyróżnić można dwa podstawowe kierunki zwiększania skali:

- zwiększanie rozmiarów środowiska działania robotów,
- zwiększanie liczby robotów.

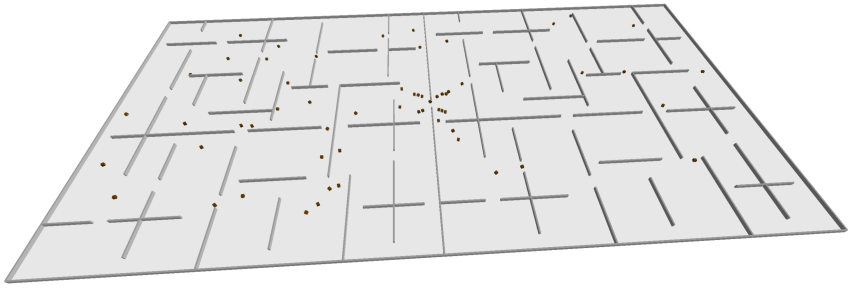
Zrealizowany system pozwala na rozbudowę środowiska o nowe w trakcie działania systemu, bez konieczności zatrzymywania żadnych jego elementów. Możliwość zwiększania rozmiarów środowiska jest praktycznie nieograniczona. Wynika to ze sposobu realizacji podsystemu planowania trasy, który jest realizowany przez grupę agentów MSA. Może być z powodzeniem rozpraszany pomiędzy wiele komputerów, dzięki czemu nie wprowadza ograniczenia rozmiarów obsługiwanego środowiska. Pozostałe komponenty systemu są praktycznie niezależne od rozmiarów środowiska. W przeprowadzonych badaniach wykorzystano środowiska składające się z 30, 60 oraz 120 pokoi. Wyniki pomiarów nie wykazały zależności pomiędzy rozmiarem środowiska, a poprawnością czy skutecznością działania systemu.

Przyjętą miarą skalowalności systemu, ze względu na liczbę robotów działających jednocześnie w środowisku, jest skuteczność działania systemu, która została zdefiniowana w rozdziale 9.3. Wraz ze wzrostem liczby robotów skuteczność będzie się zmniejszać, ponieważ roboty działające we wspólnym środowisku wzajemnie wpływają na swoje trajektorie. Ważne jednak, by zależność skuteczności od liczby robotów była liniowa. Zakładając, że danym środowisku średnia liczba robotów w pokoju nie jest większa od 3, wartość skuteczności powinna być równa co najmniej połowie maksymalnej osiągalnej skuteczności.

Opisane w rozdziałach 9.3 oraz 9.4 badania wykazały, że podstawowa wersja systemu wraz z algorytmem równoważenia obciążenia może nie być dostatecznie skalowalna w pewnych typach środowisk. Zwiększanie liczby robotów w środowiskach, w których duże

fragmenty połączone są pojedynczym przejściem, powoduje gwałtowny, nieliniowy spadek skuteczności systemu.

W celu poprawienia skuteczności działania systemu w zastosowaniu do środowiska widocznego na rysunku 9.6b, dodano agenta AMA, który zarządzał ruchem w przejściu łączącym lewą i prawą cześć środowiska. Własności zrealizowanego systemu pozwoliły na dodanie agenta AMA w trakcie działania systemu, bez konieczności zatrzymywania pozostałych elementów. Dodany agent AMA stosował algorytm kontroli kolejności i ustawiał roboty w cztery kolejki, po dwie z każdej strony przejścia.



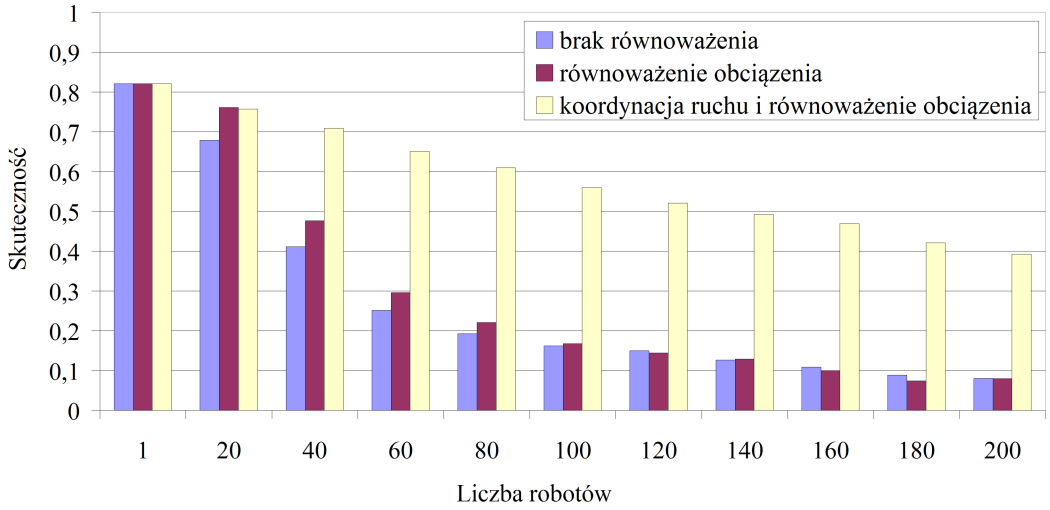
Rysunek 9.22: Wizualizacja układu robotów w środowisku złożonym z 60 pokoi dla grupy 60 robotów. Stan po stu sekundach działania systemu.

Rysunek 9.22 prezentuje układ grupy 60 robotów po 100 sekundach od utworzenia agenta AMA. W porównaniu do układu widocznego na rysunku 9.9 widać znacznie lepszą organizację działania robotów. W okolicy kluczowego przejścia zgromadzonych jest jedynie kilka robotów, podczas gdy pozostałe przemieszczają się w innych fragmentach środowiska. Zastosowanie agenta AMA zarządzającego ruchem w przejściu pomiędzy częściami środowiska znacznie zwiększyło liczbę robotów pokonujących przejście w jednostce czasu. Dzięki temu ruch wszystkich robotów był znacznie bardziej płynny.

Wyniki pomiarów skuteczności tak skonfigurowanego systemu prezentuje rysunek 9.23. Dla porównania na tym samym wykresie zaprezentowano również wyniki uzyskane podczas przeprowadzania analogicznego eksperymentu bez zastosowania agenta AMA. Zależność skuteczności systemu od liczby robotów jest bliska liniowej, a jej wartość jest zadowalająca, nawet dla licznych grup robotów.

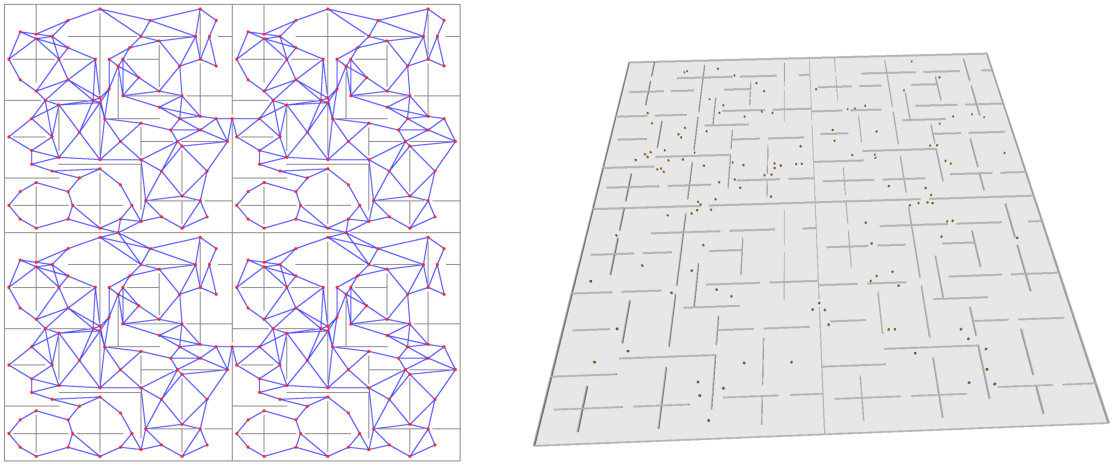
W celu potwierdzenia możliwości skalowalności zrealizowanego systemu przeprowadzono testy z wykorzystaniem dwukrotnie większego środowiska, złożonego ze 120 pokoi. Działały w nim grupy robotów o liczebności od 40 do 400. Układu pokoi w środowisku oraz skonstruowany grafowy model są widoczne po lewej stronie rysunku 9.24.

Przeprowadzone zostały trzy serie testów, podczas których mierzono skuteczność działania systemu. Podobnie jak w poprzednich testach, w każdej serii badano zależność pomiędzy liczbą robotów a skutecznością dla:



Rysunek 9.23: Skuteczność działania zintegrowanego systemu nawigacji, w zależności od liczby robotów, w środowisku złożonym z 60 pokoi.

- podstawowej wersji systemu,
- podstawowej wersji z uruchomionym algorytmem równoważenia obciążenia,
- wersji zintegrowanej z czterema agentami AMA, które zarządzały ruchem w okolicach połączeń pomiędzy segmentami środowiska.

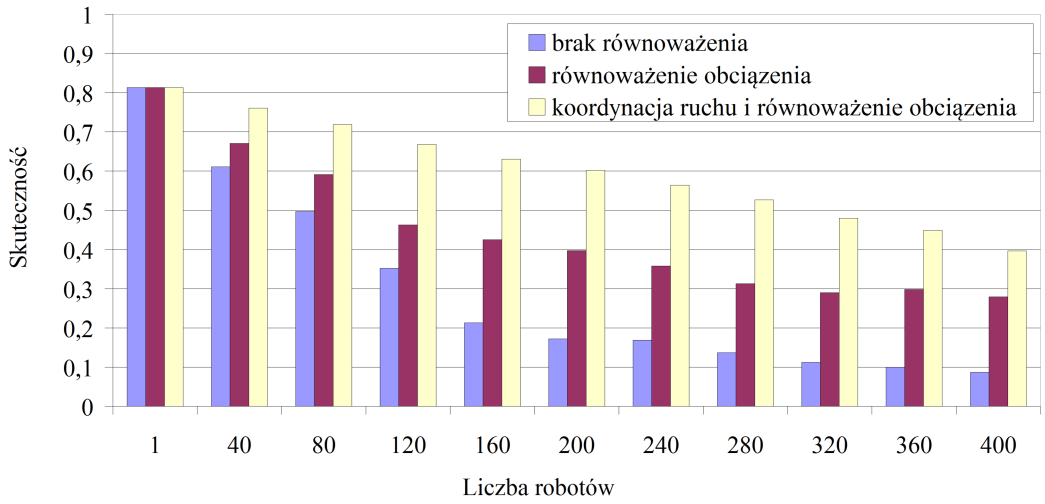


Rysunek 9.24: Po lewej: schemat środowiska złożonego ze 120 pokoi oraz jego grafowy model; po prawej: układ grupy 120 robotów w środowisku po 100 sekundach testu zintegrowanego systemu zarządzania ruchem.

W celu poprawienia wydajności oprogramowania symulacyjnego, symulację przestrzeni rzeczywistej prowadzono z wykorzystaniem 4 komputerów klasy PC (specyfikacja w dodatku A.1). Pozwoliło to na prowadzenie dostatecznie dokładnych obliczeń w czasie rzeczywistym – upływ czasu w symulowanym środowisku był taki jak w rzeczywistości.

System agentowy zarządzający ruchem robotów uruchomiono na dwóch komputerach. Na jednym pracowały agenty AMA oraz MSA, na drugim agenty TExMA oraz RWA.

Na rysunku 9.24 po prawej stronie widoczny jest układ 120 robotów po 100 sekundach trwania eksperymentu. Zauważyć można, że w okolicach wąskich przejść gromadzą się jedynie nieliczne roboty, a ruch pozostałych jest płynny. Wyniki pomiarów skuteczności działania systemu prezentuje rysunek 9.25



Rysunek 9.25: Porównanie zaleznosci pomiędzy liczebnością grupy robotów a skutecznością działania różnych konfiguracji systemu zarządzającego ruchem.

Wyniki uzyskane podczas testowania podstawowej wersji systemu są dalekie od zadowalających. Skuteczność jest bardzo niska już dla stosunkowo niewielkiego zagęszczenia robotów w środowisku. Jest to spowodowane powstawaniem zatoru w okolicy jednego z przejść łączących segmenty środowiska. W poszczególnych eksperymentach różne przejścia były przyczyną powstawania zatoru, jednak zawsze jeden zator gromadził większość robotów działających w środowisku.

Algorytm równoważenia obciążenia sprawdza się w przypadku testowanego środowiska znacznie lepiej, niż w przypadku środowiska złożonego z dwóch segmentów połączonych jednym przejściem. W testowanym środowisku występują cztery przejścia pomiędzy parami segmentów, dzięki czemu algorytm równoważenia może wytyczy trasę omijającą największy zator. Wpływa to bardzo wyraźnie na skuteczność systemu.

Po uruchomieniu czterech agentów AMA udało się uzyskać pożądaną charakterystykę zależności skuteczności od liczby robotów. Jest ona bliska liniowej, a wartości skuteczności są dostatecznie duże.

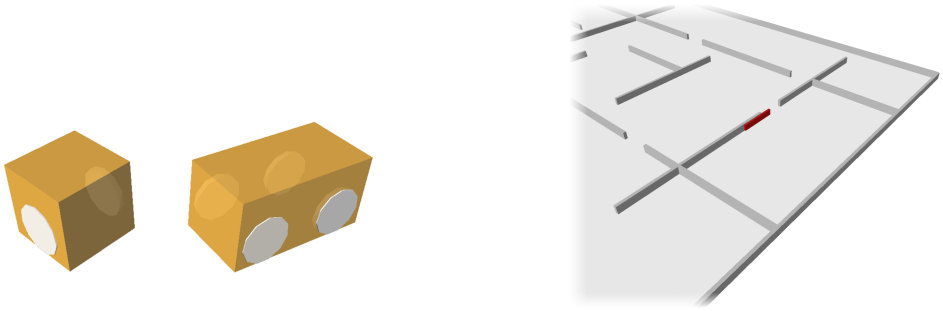
9.6.2 Rozszerzalność systemu

Architektura systemu umożliwia bardzo łatwe rozbudowywanie jego funkcjonalności. Bardzo ważną cechą zastosowanej platformy agentowej jest możliwość dodawania nowych komponentów programowych bez jej zatrzymywania ani bez konieczności zatrzymywania działających w niej agentów. Pewne kierunki rozszerzania systemu zostały już zaprezentowane we wcześniejszych rozdziałach. Były to:

- Modyfikacja programu agenta MSA, która dodała algorytm równoważenia obciążenia. Wdrożenie modyfikacji zostało zrealizowane poprzez zniszczenie agentów MSA, dodanie programu nowego rodzaju agenta MSA do platformy oraz odtworzenie agentów MSA wykorzystujących nową wersję algorytmu.
- Dodanie agentów AMA, które zarządzały ruchem w przejściach. Ta operacja również została wykonana bez zatrzymywania pozostałych elementów platformy.

Inne prawdopodobne kierunki rozszerzania systemu nawigacji robotów to na przykład:

- dodanie robota nowego typu,
- dodanie fragmentu środowiska nowego rodzaju.



Rysunek 9.26: Po lewej: porównanie modeli robota FIRA i nowego, większego robota czterokołowego; po prawej: fragment środowiska z drzwiami ograniczającymi dostęp do pomieszczenia.

Podczas przeprowadzonych eksperymentów symulacyjnych udało się dodać do systemu nowy model robota, bez konieczności modyfikowania lub zatrzymywania któregokolwiek z

pozostałych elementów systemu. Nowy robot był dwukrotnie dłuższy od robotów FIRA i był napędzany czterema kołami (rysunek 9.26 po lewej). Uruchomienie robota wymagało dodanie nowego agenta RWA, który potrafił sterować odmiennym modelem. Nowy RWA implementował te same usługi co RWA dla robotów FIRA, więc z punktu widzenia pozostałych elementów systemu prawie niczym się nie różnił. Po dodaniu robota, system funkcjonował bez zakłóceń. Jedyną zaobserwowaną różnicą było zachowanie agenta AMA, który zwiększył odległość między robotami w kolejce, w której znajdował się nowy, większy robot.

owego rodzaju fragmentu środowiska, który był dodawany w trakcie działania systemu, były drzwi przesuwne (rysunek 9.26 po prawej). W początkowej fazie eksperymentu, drzwi były cały czas otwarte. Następnie zmieniono wymagania w taki sposób, że system miał za zadanie zamykać drzwi, gdy tylko było to możliwe. W tym celu dodano do systemu agenta typu Master Agent, podobnego do AMA, który jednak nie wpływał w żaden sposób na działanie robotów. Obserwował jedynie okolice drzwi i otwierał je w chwili. gdy jakikolwiek robot znajdował się w niewielkiej odległości. Takie rozwiązanie nie wymagało zatrzymywania żadnego innego elementu systemu i nie wpłynęło na jego ogólną skuteczność.

9.6.3 Odporność na awarie

W celu zbadania zachowania systemu w sytuacjach awaryjnych przeprowadzono cztery eksperymenty, które symulowały uszkodzenie poszczególnych elementów. Były to:

- awaria robota,
- zniszczenie agenta RWA,
- zniszczenie agenta TExMA,
- zniszczenie agenta MSA.

Awaria robota została zrealizowana poprzez odwrócenie symulowanego robota kołami do góry. Nadal odpowiadał on na polecenia sterującego nim RWA, ale nie przemieszczał się. Agent RWA wykrył problem po około 10 sekundach prób przemieszczenia robota w różnych kierunkach. Brak zmiany lokalizacji robota został uznany za błąd sprzętu i zgłoszony do agenta TExMA. W systemie nie realizowano automatycznych metod obsługo błędów, więc zachowaniem agenta TExMA było usunięcie agenta RWA i przekazanie informacji o problemie do interfejsu graficznego użytkownika.

Uszkodzenie agenta RWA powoduje, że przestaje on zgłaszać do TExMA informacje o postępach w wykonaniu trasy. Jeżeli przez dłuższy czas TExMA nie otrzyma informacji o zakończeniu przemieszczania do kolejnego punktu na trasie robota, sprawdza, czy agent RWA funkcjonuje poprawnie. Wykrycie usterki agenta (czyli braku najczęściej obecności

agenta w systemie) powoduje natychmiastowe utworzenie nowego agenta RWA, przejęcie przez niego odpowiedniego robota i kontynuowanie zadania.

Agent TExMA jest w zrealizowanym systemie komponentem udostępniającym interfejs użytkownika. Awaria agenta TExMA będzie zatem łatwo zauważona przez operatora systemu, ponieważ jego interfejs przestanie działać. W przeprowadzonym eksperymencie operator nie podjął żadnych kroków po zniszczeniu agenta TExMA. Bezpieczeństwo robotów nadal było zapewnione, a ich przemieszczanie kontynuowane, ponieważ każdy z agentów RWA autonomicznie przemieszczał robota do określonych punktów, nadal skutecznie omijając przeszkody. Agenty AMA również nadal poprawnie działały, koordynując ruch w określonych obszarach. Po pewnym czasie roboty podległe uszkodzonemu TExMA ukończyły zadane przemieszczanie i zatrzymały się w ostatniej lokalizacji docelowej.

Jeżeli uszkodzeniu ulegnie agent MSA, następne żądanie zaplanowania trasy nie zostanie zrealizowane. Przeprowadzone, testowe zniszczenie agenta MSA spowodowało prawie natychmiastowe poinformowanie operatora systemu o problemie. W zrealizowanej implementacji nie umieszczono metod automatycznego odtwarzania elementów podsystemu planowania trasy, ale tego typu rozwiązanie można z łatwością zrealizować.

10 Wnioski i kierunki dalszych prac

W niniejszej pracy przedstawione zostało podejście do projektowania i implementowania systemów informatycznych zarządzających działaniem grup robotów mobilnych. Ma na celu umożliwienie tworzenia systemów charakteryzujących się wysoką jakością, czyli spełniających szereg wymagań pozafunkcjonalnych. Podstawowe wymagania pozafunkcjonalne, jakimi charakteryzować się powinien taki system, to:

- skalowalność,
- rozszerzalność,
- wysoka dostępność, odporność na awarie,
- łatwość ponownego wykorzystywania elementów.

Powyższe cechy można osiągnąć poprzez odpowiednie zastosowanie paradygmatu agentów programowych do zrealizowania systemu zarządzającego robotami. Podejście agentowe wprowadza podział systemu na luźno powiązane komponenty, które komunikują się za pomocą asynchronicznych wiadomości. Ułatwia też osiągnięcie skalowalności systemu dzięki możliwości łączenia wielu komputerów w jedną maszynę wirtualną, w której wykonywany jest program agentów. Pozwala także na osiągnięcie wysokiej niezawodności systemu dzięki ograniczeniu zależności pomiędzy jego elementami.

Przedstawione w rozdziale 6 rozdzielenie przestrzeni działania agentów od przestrzeni działania robotów pozwala na uniezależnienie implementacji poszczególnych komponentów od wykorzystywanego przez system sprzętu. Ułatwia to późniejsze wykorzystywanie fragmentów tworzonego oprogramowania. Zaproponowane podejście wielomodelowe do rozwiązywania problemów o nieliniowej złożoności pozwala na uzyskanie skalowalności systemów, które muszą takie problemy rozwiązywać.

W rozdziale 7 opisany został projekt systemu informatycznego zarządzającego działaniem grupy robotów. Analiza cech zaprojektowanego systemu pozwala sądzić, że będzie się on charakteryzował wszystkimi pożądanymi cechami pozafunkcjonalnymi. W celu eksperymentalnego zweryfikowania tej tezy zrealizowana została prototypowa implementacja systemu zarządzającego ruchem robotów. Jej szczegółowy opis znajduje się w rozdziale 8.

Zrealizowany system zarządzający ruchem robotów został szczegółowo przebadany pod kątem zdefiniowanych cech pozafunkcjonalnych. Wyniki przeprowadzonych eksperymentów, przedstawione w rozdziale 9, pozwalają stwierdzić, że charakteryzuje się on wszystkimi pożądanymi cechami pozafunkcjonalnymi.

Przebadane zostały dwa podstawowe kierunki zwiększania skali systemu: powiększanie środowiska oraz zwiększanie liczby robotów. Dzięki zastosowaniu podejścia wielomodelowego do rozwiązania problemu planowania tras dla robotów, wielkość środowiska obsługiwanego przez system jest praktycznie nieograniczona. Wykazano również, że system jest w stanie sprawnie zarządzać ruchem bardzo licznych grup robotów. Nawet przy dużym zagęszczeniu robotów w środowisku zachowana została płynność ruchu, a skuteczność działania systemu był zadowalająca.

Zastosowanie zaproponowanego podejścia do projektowania systemów zarządzających robotami pozwoliło na uzyskanie bardzo elastycznej architektury. W trakcie badań wielokrotnie dodawano do działającego systemu nowe elementy i algorytmy. Nigdy nie wymagało to przerywania działania systemu ani zatrzymywania agentów niezwiązanych z bezpośrednio wprowadzanymi zmianami. Zaprezentowano sposób rozszerzania systemu o elementy takie jak komponenty koordynujące ruch we fragmentach środowiska czy roboty nieznanego wcześniej typu.

Odporność systemu na awarie została wykazana poprzez przetestowanie zachowania systemu w sytuacjach wyjątkowych. Uszkodzenie poszczególnych elementów systemu zostało poprawnie zdiagnozowane i obsłużone przez odpowiednie komponenty. W każdej z przetestowanych sytuacji zachowane zostało bezpieczeństwo przemieszczania robotów.

Odpowiednie zastosowanie paradygmatu agentowego pozwala na podzielenie oprogramowania, składającego się na system, na możliwie niezależne elementy. Pozwala to na ponowne wykorzystanie komponentów w innych systemach. W zrealizowanym systemie jedynie agent bezpośrednio sterujący danym robotem jest ściśle powiązany z rodzajem wykorzystywanego sprzętu. Pozostałe elementy, jak podsystem planowania trasy czy agent koordynujący ruch grupy robotów, mogą zostać bez zmian zastosowane w innym systemie zarządzającym działaniem innego typu robotów.

Można zatem stwierdzić, że postawiona w pracy teza jest prawdziwa. Logiczne oddzielenie agenta programowego od obiektu robota oraz jednoczesne zastosowanie wielu modeli środowiska pozwala na skonstruowanie systemu zarządzającego grupą robotów mobilnych, który będzie się charakteryzował wszystkimi pożądanymi cechami pozafunkcjonalnymi.

Zaproponowane w pracy podejście do tworzenia systemów informatycznych zarządzających robotami mobilnymi może stanowić podstawę do dalszych prac i badań w tej dziedzinie. Najciekawszym kierunkiem rozwoju podejścia są badania nad platformami agentowymi dedykowanymi do zarządzania robotami. Wykorzystane w prototypowej implementacji środowisko JADE jest funkcjonalne i łatwe w użyciu, jednak możliwość jego zastosowania w bardzo złożonych systemach wydaj się ograniczona. Główną przyczyną jest niska zbyt wydajność warstwy komunikacji i ograniczona stabilność.

Platforma agentowa dedykowana do zastosowań w systemach zarządzających robotami powinna zapewniać bardzo wydajną komunikację pomiędzy agentami, ponieważ czas wykonania niektórych poleceń jest kluczowy dla bezpieczeństwa robotów. Powinna także umożliwiać uruchamianie fragmentów platformy na niezbyt rozbudowanych komputerach, jakie często są wykorzystywane jako wbudowane sterowniki robotów. Umożliwi to uruchamianie agentów RWA na lokalnych komputerach robotów, co poprawi ich skuteczność i bezpieczeństwo działania.

Innym ciekawym kierunkiem rozwoju podejścia wydaje się rozbudowywanie mechanizmu usług o ich semantykę. Tego typu rozwiązania są w ostatnich latach intensywnie badane w dziedzinie usług programowych (Web Services), a przeniesienie uzyskanych wniosków i zdefiniowanych standardów na grunt usług realizowanych przez roboty w świecie rzeczywistym powinno dać wartościowe wyniki.

Literatura

[1] Agmon, N.; Kraus, S.; Kaminka, G.A.; *Multi-Robot Perimeter Patrol in Adversarial Settings.* Proceedings of: IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 2008, s. 2339–2345.

[2] Ahmad, H.F.;
Multi-agent systems: overview of a new paradigm for distributed systems. Proceedings of: High Assurance Systems Engineering, IEEE, 2002, s. 101–107.

[3] Albus, J. S.;
A reference model architecture for intelligent systems design. An Introduction to Intelligent and Autonomous Control Kluwer Academic Publishers, 1993, s. 27–56.

[4] Ambroszkiewicz, S.; Bartyna, W.;
Architektura systemu wielorobotowego w oparciu o paradygmat SOA. Proceedings of: IX Krajowa Konferencja Robotyki, Piechowice, 13-16 września, 2006.

[5] Ambroszkiewicz, S.; Borkowski, A.; Cetnarowicz, K.; Turek, W.;
Adaptive Bilayer Control of Multiple Mobile Robots. Proceedings of 16th International Symposium on Measurements and Control in Robotics. Warszawa, 2007, s. 31–50.

[6] Ambroszkiewicz, S.; Cetnarowicz, K.;
On the concept of agent in multi-robot environment. Innovative Concepts for Autonomic and Agent-Based Systems. Lecture Notes in Computer Science, Workshop on Radical Agent Concept, Washington DC, USA, 2006, Springer-Verlag Berlin-Heidelberg 3825, s. 135–146.

[7] Ambroszkiewicz, S.; Cetnarowicz, K. and Turek, W.;
Multi-Robot Management Framework based on the Agent Dual-Space Control Paradigm. Proceedings of: AAAI'07 Fall Symposium, Arlington, Virginia, USA, 2007, s. 32–37.

[8] Azarm, K.; Schmidt, G.;
Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. Proceedings of: IEEE Int. Conf. on Robotics and Automation, Albuquerque, NM, USA, 1997, s. 3526–3533.

[9] Balch, T.; Arkin, C.;
Behavior-based Formation Control for Multi-robot Teams. EEE Transactions on Robotics and Automation, 14, 1997, s. 926–939.

[10] Balch, T.;
The Impact of Diversity on Performance in Multi-robot Foraging. Proceedings of: International Conference on Autonomous Agents, Washington, USA, 1999, s. 92–99.

[11] Balch, T.; Arkin, R.;
Behavior-based Formation Control for Multi-robot Teams. IEEE Transactions on Robotics and Automation, 14, 1999, s. 926–939.

[12] Batalin, M. A.; Sukhatme, G. S.; Nardi, D.;
Spreading out: A local approach to multi-robot coverage. Springer-Verlag, Distributed Autonomous Robotic Systems, 5, 2002, s. 373–382.

[13] Bellifemine, F.; Poggi, A.; Rimassa, G.;
JADE – A FIPA-compliant agent framework. Proceedings of: PAAM'99, London, UK, 1999, s. 97–108.

[14] Bellifemine, F.; Poggi, A.; Rimassa, G.;
Developing multi-agent systems with a FIPA-compliant agent framework. Software – Practice And Experience, 31, 2001, s. 103–128.

[15] Bennewitz, M.; Burgard, W.; Thrun, S.;
Optimizing Schedules for Prioritized Path Planing of Multi-Robot System. Proceedings of: International Conference on Robotics and Automation, Seoul, Korea, May 21–26, 2001, s. 271–276.

[16] Bennewitz, M.; Burgard, W.; Thrun, S.;
Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. Robotics and Autonomous Systems, 41 (2), 2002, s. 89–99.

[17] Bien, Z.; Lee, J.;
A minimum-time trajectory planning method for two robots. IEEE Transactions on Robotics and Automation 8 (3), 1992, s. 414–418.

[18] Binder, T.; El-Fattah, Y.; Nawarecki E.; Perret, R.;
Synthesis of complex control objects. Proceedings of: Second Europeen Meeting on Cybernetics and Systems Research. Wiedeń, Austria, 1974.

[19] Binder, T.; Nawarecki E.;
A computer algorithm of variable structure for the control. Proceedings of: 4-th IFAC/IFIP Conference, Zurrich, Szwajcaria, 1974.

[20] Bondi, A.B.;
Characteristics of scalability and their impact on performance. Proceedings of: 2nd

international workshop on Software and performance, Ottawa, Ontario, Canada, 2000, s. 195–203

[21] Borkowski, A.; Cetnarowicz, K.; Turek, W.;
Porównanie metod koordynacji ruchu grupy robotów mobilnych. Proceedings of: X Krajowa Konferencja Robotyki, Piechowice, 3-6 września, 2008.

[22] Brooks, R.;
A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 2 (1), 1986, s. 14–23.

[23] Cetnarowicz, K.;
M-Agent Architecture Based Method of Development of Multiagent Systems. Proceedings of: International Conference on Physics Computing, 1996.

[24] Choset, H. et al.;
Principles of Robot Motion – Theory, Algorithms, and Implementations. MIT-Press, 2005.

[25] Dechter, R.; Pearl, J.;
Generalized best-first search strategies and the optimality of A.* Journal of the ACM. 32 (3), 1985, s. 505–536.

[26] Dias, M.B.; Zlot, R.; Kalra, N.; Stentz, A.;
Market-Based Multirobot Coordination: A Survey and Analysis. Proceedings of the IEEE, 94 (7), 2006, s. 1257–1270.

[27] Dijkstra, E. W.;
A note on two problems in connexion with graphs. Numerische Mathematik. 1, 1959, s. 269–271.

[28] Dudek, G.; Jenkin, M.; Wilkes, D.;
A taxonomy for multi-agent robotics. Autonomous Robots, 3, 1996, s. 375–397.

[29] Elfes, A.;
Using occupancy grids for mobile robot perception and navigation. Computer. 22 (6), IEEE Computer Society Press, 1989, s. 46–57.

[30] *„Navigation”.* Encyclopædia Britannica. 2009.
http://www.britannica.com/EBchecked/topic/407011/navigation

[31] *„Information system”.* Encyclopædia Britannica. 2009.
http://www.britannica.com/EBchecked/topic/287895/information-system

[32] *„Agent – computer science".* Encyclopædia Britannica. 2009.
http://www.britannica.com/EBchecked/topic/705121/agent

[33] Farinelli, A.; Iocchi, L.; Nardi, D.;
Multirobot systems: a classification focused on coordination. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34 (5), 2004, s. 2015–2028.

[34] Ferrari, C.; Pagello, E.; Ota, J.; Arai, T.;
Multirobot motion coordination in space and time. Robotics and Autonomous Systems 25, 1998, s. 219–229.

[35] Federation of International Robot-soccer Association;
FIRA Middle League MiroSot Game Rules.
http://fira.net/soccer/mirosot/Middle_League_MiroSot.pdf, 2006.

[36] Federation of International Robot-soccer Association;
Welcome to the World of FIRA.
http://www.fira.net/about/greetings.html, 2009.

[37] Friedrich, H.; Rogalla, O.; and Dillmann, R.;
Integrating skills into multi-agent systems. Journal of Intelligent Manufacturing, 9, 1998, s 119–127.

[38] Gallina, P.; Gasparetto, A.;
A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot. Journal of Intelligent and Robotic Systems 27 (3), 2000, s. 237–262.

[39] Gerkey, B.P.; Mataric, M.J.;
Pusher-watcher: an approach to fault-tolerant tightly-coupled robot coordination. Proceedings of: IEEE International Conference on Robotics and Automation, 1, 2002, s. 464–469.

[40] Gnatowski, M.;
Search-and-rescue using team of robots. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 9 (28), 2005, s. 17–24.

[41] Goldberg, D.; Mataric, M.;
Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks. Robot Teams: From Diversity to Polymorphism, 2001.

[42] Guo, Y.; Parker, L.;
A Distributed and Optimal Motion Planning Approach for Multiply Mobile Robots.

Proceedings of: International Conference on Robotics and Automation, Washington, DC, 05.2002, s. 2612–2619.

[43] Howard, A.; Blair, A.; Walter, D.;
Motion control for fast mobile robots: a trajectory-based approach. Proceedings of: Australian Conference on Robotics and Automation, Melbourne, Australia, 09.2000, s. 13–18.

[44] Howard, A.; Mataric, M.J.; Sukhatme, G.S.;
An Incremental Deployment Algorithm for Mobile Robot Teams. Proceedings of: IEEE International Conference on Robotics and Intelligent Systems, Lausanne, Szwajcaria, 2002, s. 2849–2854.

[45] Iocchi, L.; Nardi, D.; Salerno, M.;
Reactivity and Deliberation: A Survey on Multi-Robot Systems. Lecture Notes in Computer Science, vol. 2103, 2001, s. 9–32.

[46] Iocchi, L.; Nardi, D.; Piaggio, M.; Sgorbissa, A.;
Distributed Coordination in Heterogeneous Multi-Robot Systems. Autonomous Robots, 15 (2), 2003, s. 155–168.

[47] Jennings, N.R.; Sycara, K.; Wooldridge, M.;
A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems, 1, 1998, s. 7–38.

[48] Kalman, R.E.;
A new approach to linear filtering and prediction problems. Journal of Basic Engineering 82 (1), 1960, s. 35–45.

[49] Khoo, A.; Horswill, I.;
An Efficient Coordination Architecture for Autonomous Robot Teams. Proceedings of: IEEE International Conference on Robotics and Automation, 1, 2002, s. 287–292.

[50] Kinny, D.; Georgeff, M.; Rao, A.;
A Methodology and Modelling Technique for Systems of BDI Agents. Agents Breaking Away, Springer LNCS, 1038, 1996, s. 56–71.

[51] Koren, Y.; Borenstein, J.;
Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. Proceedings of: IEEE Conference on Robotics and Automation, Sacramento, California, 7-12.04.1991, s. 1398–1404.

[52] Kube, C. R.; Zhang, H.;
Collective robotic intelligence. Proceedings of: Simulation of Adaptive Behavior, Honolulu, Hawai USA, 1992, s. 460–468.

[53] Latombe, J.;
Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, 1991.

[54] Leonard, J.J.; Durrant-Whyte, H.F.;
Simultaneous map building and localization for an autonomous mobile robot. Proceedings of Intelligent Robots and Systems. 3-5.11.1991, s. 1442–1447.

[55] Marchetti, L.; risetti, G.; Iocchi, L.;
A Comparative Analysis of Particle Filter Based Localization Methods. Lecture Notes in Computer Science 4434, Springer, 2007, s. 442–449.

[56] Mataric, M.;
Behavior-based robotics. MIT Encyclopedia of Cognitive Sciences, 1999, s. 74–77.

[57] Mauve, M.; Widmer, J.; Hartenstein, H.;
A Survey on Position-Based Routing in Mobile Ad Hoc Networks. IEEE Network 1 (6), s. 30–39

[58] Mazur, M.;
Pojęcie systemu i rygory jego stosowania. Postępy Cybernetyki, (10) 2, 1987, s. 21–29.

[59] Merlin Robotics. 2009.
http://www.merlinrobotics.co.uk/

[60] Minguez, J.; Montano , L.; Santos-Victor, J.;
Reactive navigation for non-holonomic robots using the ego kinematic space. Proceedings of: International Conference on Robotics and Automation, Washington, USA, 2002, s. 3074–3080.

[61] Neves, M.; Oliveira, E.;
A control architecture for an autonomous mobile robot. Proceedings of: First International Conference on Autonomous Agents, New York, NY, USA, 1997, s. 193–200.

[62] Nurnberg, P.;
Extensibility in component-based open hypermedia systems. Journal of Network and Computer Applications, 24, 2001, s. 19–38.

[63] Owen, C.; Nehmzow, U.;
Landmark-based navigation for a mobile robot. Proceedings of: Simulation of Adaptive Behaviour. MIT Press, 1998, s. 240–245.

[64] Parker, L.;
ALLIANCE: An Architecture for Fault Tolerant Cooperative Control of Heterogeneous Mobile Robots. Proceedings of: IEEW/RSJ Conference on Intelligent Rohots and Systems (IROS). Munich, Germany, 1995, s. 776–783.

[65] Parker, L.;
ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. IEEE Transactions on Robotics and Automation, 14, 1998, s. 220–240.

[66] Parker, L.;
Cooperative Robotics for Multi-Target Observation. Intelligent Automation and Soft Computing, 5, 1999, s. 5–19.

[67] Rao, A.; Georgeff, M.;
BDI-agents: from theory to practice. Proceedings of: First International Conference on Multiagent Systems, San Francisco, USA, 1995.

[68] RoboCup Federation;
RoboCup Brief Introduction.
http://www.robocup.org/Intro.htm, 2009.

[69] Sommerville, I.;
Software Engineering. Pearson Education, 2007.

[70] Siemiatkowska, B.;
Hybrydowa reprezentacja otoczenia robota mobilnego. Pomiary, Automatyka, Robotyka 3, 2007.

[71] Smith, R.;
Constraints in Rigid Body Dynamics, Game Programming Gems 4, Charles River Media Inc., 2004.

[72] Smith, R.;
Open Dynamics Engine. http://ode.org/, 2009.

[73] Smith, R.C.; Cheeseman, P.;
On the Representation and Estimation of Spatial Uncertainty. The International Journal of Robotics Research 5 (4), 1986, s. 56–68.

[74] Smith, R.C.; Self, M.;Cheeseman, P.;
Estimating Uncertain Spatial Relationships in Robotics. Proceedings of: Second Annual Conference on Uncertainty in Artificial Intelligence, University of Pennsylvania, Philadelphia, PA, USA, s. 435–461.

[75] Spero, D. J.;
A review of outdoor robotics research. Technical Report MECSE-17-2004, Department of Electrical and Computer Systems Engineering, Monash University, Melbourne, Australia, 2004.

[76] Trevai, C.; Fujii, N.; Ota, J.; Arai, T.; *Multiple Mobile Robot Exploration and Patrol Strategy Using a Self-Organizing Planner Based on a Reaction-Diffusion Equation on a Graph.* Journal of Robotics and Mechatronics, 20 (1), 2008, s. 24–37.

[77] Turek, W.;
Extensible Multi-Robot System. Lecture Notes in Computer Science 5103, Springer, 2008, s. 574–583.

[78] Turek, W.; Marcjan, R.; Cetnarowicz, K.;
A Universal Tool for Multirobot System Simulation. Knowledge-Driven Computing, Springer, 2008, s. 289–303.

[79] Organization for the Advancement of Structured Information Standards;
UDDI Specification. http://www.oasis-open.org/committees/tc_home.php ?wg_abbrev=uddi-spec, 2006

[80] Urdiales, C. et al.;
A purely reactive navigation scheme for dynamic environments using case-based reasoning. Autonomous Robots, 21 (1), 2006, s. 65–78.

[81] Wagner, I.; Lindenbaum, M.; Bruckstein, A.;
Distributed Covering by Ant-Robots Using Evaporating Traces. IEEE Transactions on Robotics and Automation, 15, 1999, s. 918–933.

[82] Wooldridge, M.; Jennings, N.R.;
Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 10, 1995, s. 115-152.

[83] Zaborowski, W.;
Zdecentralizowany system rozdziału zadań w zespole homogenicznych robotów w warunkach ograniczeń komunikacyjnych. Rozdział w POSTĘPY ROBOTYKI: Systemy i współdziałanie robotów, Wydawnictwo Komunikacji i Łączności, 2006, s. 329–338.

[84] Zieliński, C.;
Specification of behavioural embodied agents. Proceedings of: Fourth International Workshop on Robot Motion and Control, 2004, s. 79–84.

[85] Zieliński, C.; Szynkiewicz, W.; Trojanek, P.; Majchrowski, M.;
Sterowanie zespołu heterogenicznych robotów na przykładzie pchania pudła. Rozdział w POSTĘPY ROBOTYKI: Systemy i współdziałanie robotów. 2006, s. 299–308.

A Specyfikacje sprzętu wykorzystanego w testach

A.1 Komputery klasy PC

W testach wydajności algorytmów wykorzystano komputery klasy PC o następujących parametrach:

- procesor dwurdzeniowy Intel Core Duo 2,5 GHz,
- pamięć operacyjna 4 GB 800 MHz,
- karta graficzna NVidia GeForce 256 MB,
- dysk twardy SATA 250 GB,
- karta sieciowa 1 Gb Ethernet.

Komputery pracowały pod kontrolą systemu operacyjnego Windows XP. W testach wymagających kilku komputerów wykorzystano sieć lokalną o prędkości 1 Gb.

A.2 Robot FIRA Miabot

Roboty wykorzystywane w testach, to niewielkie, dwukołowe roboty mobilne o napędzie różnicowym. Zostały wykonane przez firmę Merlin Robotics [59]. Są zgodne z specyfikacją ligi robotów FIRA MiroSotMiddle League MiroSot[35]. Następujące parametry charakteryzują roboty Miabot:

- wymiary: 7,5 x 7,5 x 7,5 [cm],
- napęd: dwukołowy, różnicowy, optyczny enkoder o rozdzielczości 0,04 mm,
- prędkość maksymalna: 3,5 m/s,
- źródło zasilania: 6 x akumulator 1,2V NiMH, AA,
- jednostka sterująca: Atmel ATMega64
  - prędkość: 14,5 MIPS (RISC),
  - pamięć programu: 64KB Flash,
  - pamięć operacyjna: 4KB SRAM,
  - pamięć nieulotna: 2KB EEPROM,
  - programowanie: w systemie, interfejs JTAG
- programowanie: kompilator GCC,
- możliwość rozbudowy: port ośmiobitowy.