Roboty monitorujace skażenie środowiska - symulator V-REP

Anna Wujek
Łukasz Korpal
Wiktor Ślęczka

10 czerwca 2016

Spis treści

1 Zadanie

Celem projektu było stworzenie symulatora mobilnej sieci ad hoc (MANET) do monitorowania skażenia środowiska naturalnego. Węzłami sieci są roboty mobilne, wyposażone w czujniki oraz komunikujące się między sobą. Zadaniem robotów jest lokalizacja chmury skażenia i otoczenie jej robotami, monitorującymi jej położenie i granice, przy założeniu utrzymania spójności sieci.

2 Projekt systemu

2.1 Założenia

- Roboty potrafią się na bieżąco lokalizować.
- System składa się z pewnej liczby robotów mobilnych.
- Roboty mobilne działają autonomicznie i potrafią same zorganizować sieć.
- Roboty są wyposażone w czujniki pozwalające im wykrywać poziom skażenia oraz przeszkody.
- Roboty są wyposażone w urządzenia pozwalające im się między sobą komunikować się między sobą.
- Chmura skażenia jest odgórnie określona i niezmienna w trakcie trwania symulacji.
- Nieznana jest mapa obszaru poszukiwań, ale znane są jego granice i kształt - prostokąt.

2.2 Scenariusz działania

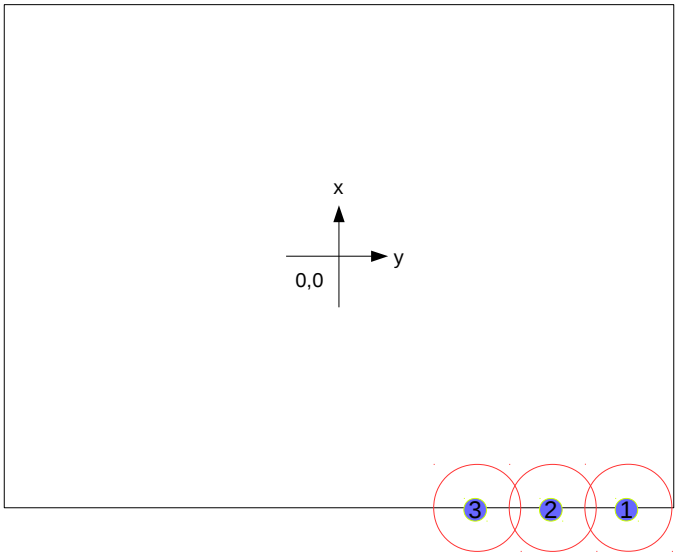
Algorytm realizowany przez każdego robota:

1. Warunek początkowy: robot znajduje się w dowolnym punkcie przestrzeni na mapie. Z punktu tego musi istnieć droga prowadząca do obszaru poszukiwań.
2. Przemieść się do punktu początkowego (rys. 1), zależnego od liczby robotów, wymiarów przestrzeni poszukiwań i zasięgu czujników (rys. 2), zgodnie ze wzorem:

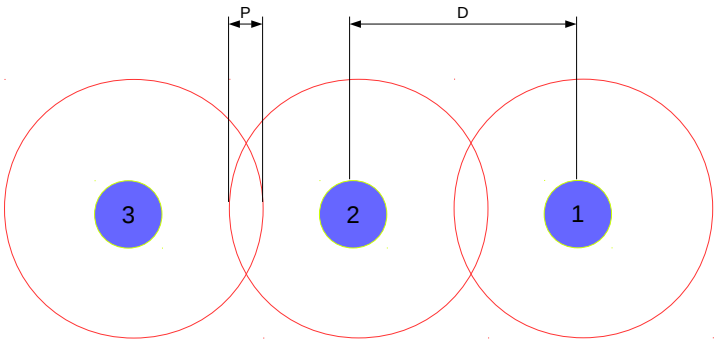
$$\begin{aligned} x &= -length/2 \\ y &= width/2 + distance * robot\_number + distance/2 \end{aligned} \tag{1}$$

W obliczeniach wykorzystywane są następujące parametry:

$$\begin{aligned} max\_spaces &= sensor\_range \cdot precision\_range \\ no\_stages &= ceil(width/(max\_spaces \cdot no\_robots)) \\ spaces\_width &= width/no\_stages \\ distance &= spaces\_width/no\_robots \end{aligned}$$



Rysunek 1: Ustawienie początkowe robotów



Rysunek 2: Sposób ustawienia robotów w szeregu. D - distance, P - zależne od precision_range

3. Wyznacz kolejny punkt docelowy, zgodnie ze wzorem:

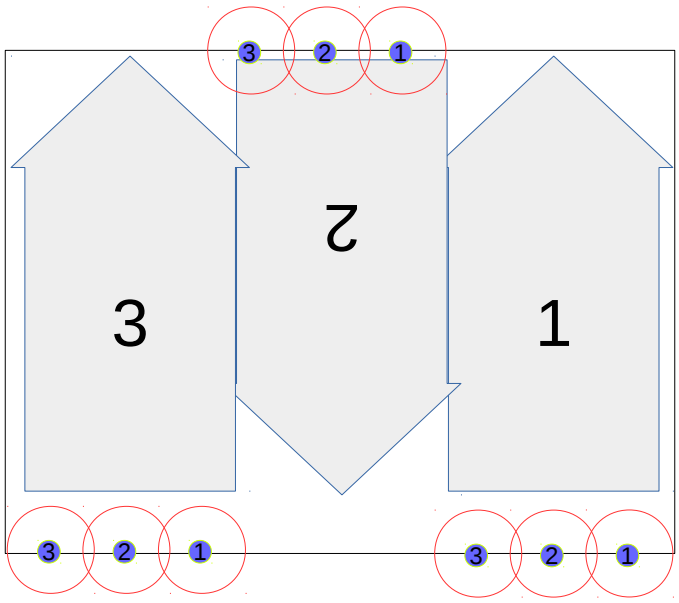
$$x_n = direction \cdot length/2 + direction + substage\_length \cdot substage$$
$$y_n = width/2 + distance \cdot robot\_number + stage \cdot spaces\_width + distance/2$$

4. Udaj się do punktu docelowego, monitorując skażenie oraz wykrywając przeszkody.

5. Jeśli wykryto przeszkodę, omiń ją algorytmem Bug2. Wyznacz następny punkt docelowy, jest aktualny został minięty.

6. Jeśli wykryto chmurę skażenia, zatrzymaj się.

7. Jeśli cały obszar poszukiwań sprawdzony, to koniec algorytmu; jeśli nie, to przejdź do punktu 3.



Rysunek 3: Ruch szeregu robotów

2.3 Struktura systemu

System prezentowany w projekcie składa się z następujących elementów:

- symulator przestrzeni roboczej
- symulator robotów
- symulator chmury skażenia
- komunikacja

2.3.1 Symulator przestrzeni roboczej

Symulator ten jest odpowiedzialny za wizualizację całej symulacji - przestrzeni roboczej, przeszkód, robotów oraz chmury skażenia. Generuje także fizyczne oddziaływania między elementami, np. kolizje oraz odczyty czujników (oprócz czujników skażenia). Pożądane jest wykorzystanie w tym celu gotowego narzędzia, oferującego modele robotów i czujników, silnik fizyczny oraz interfejsy komunikacyjne.

Narzędziem wykorzystanym w projekcie był symulator V-Rep. Jest on dedykowany do tego typu symulacji i spełnił postawione założenia. W projekcie symuluje

rzeczywiste receptory (czujniki odległości) i rzeczywiste efektory (silniki robotów) systemu, a także pełni rolę wirtualnego efektora (warstwy pośredniczącej między rzeczywistym efektorem a podsystemem sterowania, przekładającej polecenia sterowania na uruchamianie silników, a tym samym ruch robota).

2.3.2 Symulator chmury skażenia

Symulator ten jest odpowiedzialny za generowanie odczytów czujników skażenia. Odczyt jest obliczany na podstawie założonego wcześniej kształtu chmury oraz sposobu, w jaki skażenie rośnie wgłąb chmury. Ze względu na brak możliwości zaimplementowania tego typu obiektu bezpośrednio w wybranym symulatorze V-Rep, chmura skażenia została zasymulowana osobno. W projekcie pełni rolę wirtualnego receptora (warstwy pośredniczącej między rzeczywistym receptorem – czujnikiem skażenia – a podsystemem sterowania), interpretującego odczyt czujnika skażenia.

2.3.3 Symulator robota

W systemie symulator ten będzie zwielokrotniony, aby uzyskać grupę kilku robotów. Odpowiedzialny jest za wyznaczanie sterowania robotów w zależności od odczytów z czujników i wewnętrznego imperatywu, czyli zadania, jakie roboty realizują.

W projekcie pełni dwie bardzo ważne role: wirtualnego receptora (warstwy pośredniczącej między rzeczywistymi receptorami – czujnikami odległości – a podsystemem sterowania) interpretującego dane z czujników odległości (przetwarzanie polega na filtracji wyników i stwierdzeniu które pomiary są przydatne) oraz podsystemu sterowania (logiki robota, która na podstawie danych z wirtualnych receptorów podejmuje decyzje co do stanu i dalszego zachowania robota, a następnie wysyła odpowiednie sterowania do wirtualnego efektora; kolejnym zadaniem podsystemu sterowania jest komunikacja z innymi robotami w systemie).

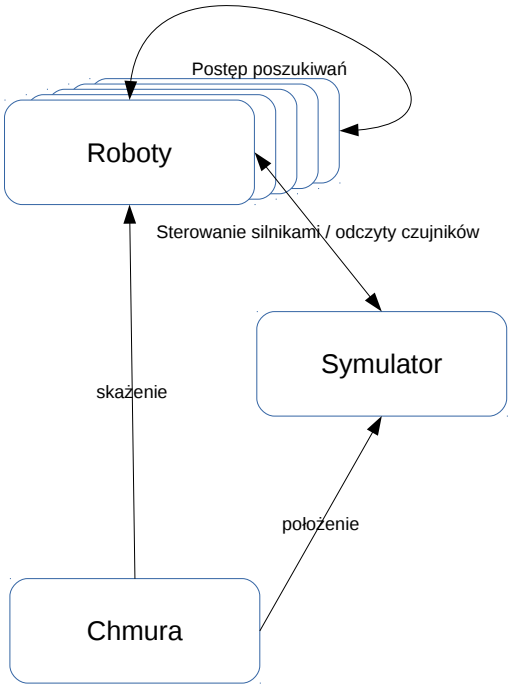
Każdy robot realizuje dokładnie ten sam algorytm - nie ma jednego wyróżnionego lidera.

2.3.4 Komunikacja

Aby cały system działał poprawnie, poszczególne elementy muszą mieć możliwość komunikacji między sobą. W systemie rozróżniamy następujące kanały komunikacyjne i przesyłane informacje:

- Symulator robota ↔ Symulator przestrzeni roboczej:
  - sterowanie poszczególnymi silnikami
  - odczyty z czujników (oprócz czujnika skażenia)
  - sprawdzenie pozycji robota (położenie + orientacja)
- Symulator robota ↔ Symulator chmury skażenia :
  - odczyt czujnika skażenia
- Symulator robota ↔ Symulator robota:
  - ostatni punkt docelowy, do którego udało się dotrzeć

– informacja o wykrytej chmurze skażenia



2.4 Algorytmy

2.4.1 Algorytm poszukiwania chmury

Roboty rozpoczynają pracę od ustawienia się w formacji linii. Następnie dla każdego robota wyznaczany jest następny punkt docelowy, do którego musi dojechać. Punkty docelowe wyznaczane są w taki sposób, aby roboty ponownie ustawiły się w linii. Po dojechaniu do punktu docelowego robot wysyła informację o tym zdarzeniu do innych robotów. Linia punktów docelowych jest jednocześnie miejscem synchronizacji - jeśli jakiś robot jechał wolniej (np. z powodu omijania przeszkody), to reszta na niego czeka. Powoduje to wolniejsze przeszukiwanie, ale zapewnia spójność sieci – robot nigdy nie jest oddalony od swoich sąsiadów bardziej, niż odległość do następnego punktu docelowego. Odległość ta dobrana jest w taki sposób, aby spójność sieci została zachowana.

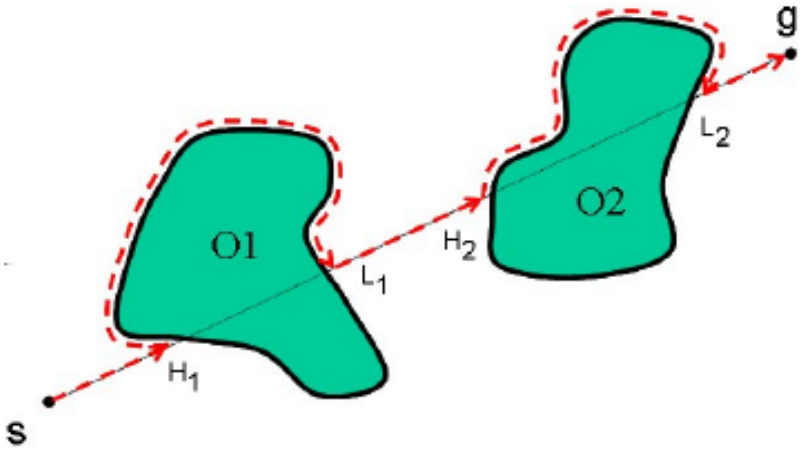
Roboty w ten sposób poruszają się do przodu do momentu dotarcia do przeszkody, toksycznej chmury lub granicy obszaru poszukiwań. W przypadku powyższych wydarzeń, uruchamiane są odpowiednie algorytmy.

Gdy roboty dotrą do końca obszaru poszukiwań, zatrzymują się, przesuwają w bok i kontynuują poszukiwania w drugą stronę, graniczc z poprzednim obszarem poszukiwań – w ten sposób szerokimi pasami przeszukują cały obszar mapy.

2.4.2 Zmodyfikowany algorytm bug 2

Algorytm ten jest stosowany w przypadku napotkania przeszkody przez robota. Polega on na podążaniu wzdłuż ściany przeszkody do momentu osiągnięcia punktu

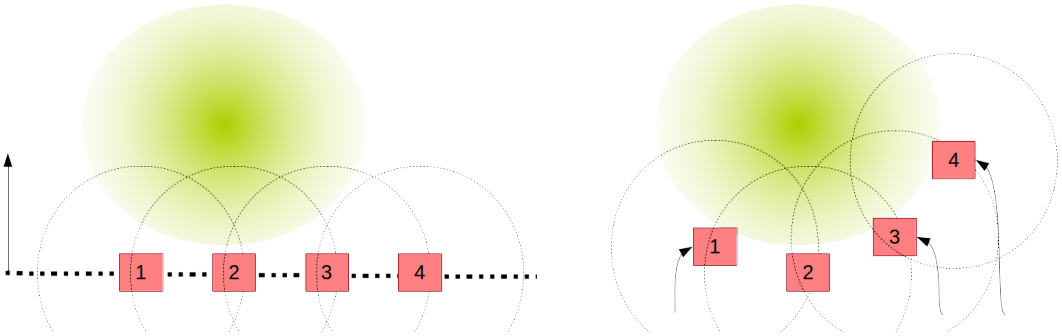
leżącego za napotkaną ścianą. Jedyną modyfikację stanowi oderwanie się od ścieżki w momencie, gdy na drodze do celu nie stoi żadna przeszkoda. Działanie oryginalnego algorytmu zostało przedstawione na rysunku 4.



Rysunek 4: Działanie algorytmu bug 2

2.4.3 Algorytm otaczania chmury

W momencie napotkania chmury przez któregokolwiek robota, zatrzymuje się on (po osiągnięciu natężenia toksyn określonego arbitralnie). Pozostałe roboty jadą dalej i po pewnym czasie skręcają do domniemanego środka chmury. Gdy w trakcie trwania tych operacji napotkają określone stężenie chmury, zatrzymują się.



Rysunek 5: Działanie algorytmu otaczania chmury. Robot 2 napotyka chmurę, pozostałe roboty zbliżają się do chmury aż do wartości granicznej, następnie cała sieć zatrzymuje się.

3 Realizacja

3.1 Plik konfiguracyjny

Plik konfiguracyjny powinien zawierać informacje o planszy, robotach i chmurze – w trzech rozdzielonych modułach.

- Moduł [board] opisuje wielkość sektora poszukiwań.
  Posiada następujące parametry:
  - width – szerokość planszy
  - length – długość planszy
- Moduł [robots] opisuje roboty biorące udział w symulacji.
  Posiada następujące parametry:
  - count – liczba robotów
  - sensor_range – zasięg czujników toksyczności
  - precision_range – parametr wymaganej precyzji, ogranicza maksymalny zasięg czujników, wpływając na mniejsze odległości między robotami (zwiększa zmienną P, por. rys. 2)
- Moduł [cloud] opisuje położenie i wielkość chmury.
  Posiada następujące parametry::
  - center – położenie środka chmury
  - radius – średnica chmury

3.2 Komunikacja

Komunikacja przebiega na zasadzie ogłaszania aktualnych informacji na swój temat przez roboty (oraz chmurę). Informacje przesyłane są za pomocą struktury JSON do gniazd, przypisywanych każdemu modułowi osobno. Wiadomości opatrzone są typem, pozwalającym sklasyfikować komunikat. Dane tworzone są w postaci usystematyzowanych słowników, których struktura została odgórnie ustalona. Komunikaty są wysyłane i odbierane asynchronicznie.

3.2.1 Roboty

Roboty wysyłają informacje o swoim postępie w wiadomościach typu "progress". Zawierają one następujące elementy:

- "robot" - numer identyfikacyjny robota
- "substage" - postęp robota wzdłuż wyznaczonej ścieżki
- "stage"- postęp robota w numerze ścieżki

Poniżej przedstawiono przykład tworzenia wiadomości o postępie robota.

```
def broadcast_info(self):
    message = str(self.robot._name)+str(self.substage)
    message = dict()
    message["robot"] = self.robot.name
    message["stage"] = self.stage
    message["substage"] = self.substage
    message["type"] = "progress"
    self.robot.commutron.broadcast(json.dumps(message))
```

Dodatkowo, symulując działanie czujników toksyczności, wysyłają zapytania do procesu chmury o stężenie w swoim położeniu. Są to

3.2.2 Chmura

Skrypt odpowiadający za symulację chmury wysyła informację o jej położeniu do robotów w wiadomościach typu ćloudread". Zawierają one następujące elementy:

- "concentration" - stężenie toksycznych substancji
- "robot" - numer robota, z którym przebiega komunikacja
- "position" - pozycja robota

Poniżej przedstawiono przykład tworzenia wiadomości o chmurze.

```
res = dict()
res['concentration'] = self.concentration(data["position"])
res['robot'] = data['robot']
res['position'] = data['position']
res['type'] = "cloudread"
```

3.3 Roboty

Wykorzystywane w realizacji zadania roboty Pionieer p3dx wyposażone w ultradźwiękowe czujniki odległości - będą one zapewnione przez symulator V-Rep. Posiadają one napęd różnicowy, pozwalający na dużą swobodę poruszania się po symulowanej przestrzeni.

Roboty te mają napęd różnicowy, co pozwala im na obracanie się w miejscu. Dlatego też realizacja ruchu robota odbywa się poprzez zadanie punktu docelowego, a następnie takie sterowanie, aby dotrzeć do niego z dowolną orientacją. Do sterowania wykorzystane zostały dwa regulatory proporcjonalne, jeden odpowiedzialny za jazdę do przodu – wolniej lub szybciej, w zależności od odległości od miejsca docelowego, drugi odpowiedzialny za obrót robota – tak, aby robot był ustawiony przodem do miejsca docelowego. Złożenie tych dwóch ruchów pozwala na płynne i gładkie dojechanie do miejsca docelowego.

Regulator odpowiedzialny za jazdę do przodu:

$$\begin{aligned} u_{11}(k) &= u_{12}(k) = K_1 \cdot e(k) \quad \text{jednakowe prędkości na oba koła} \\ e(k) &= y_{zad}(k) - y(k) \\ K_1 &= v_{max} \\ 0 &\leqslant u_{11}(k), u_{12}(k) \leqslant v_{max} \end{aligned}$$

gdzie:
$u_{11}(k), u_{12}(k)$ – sterowanie, czyli prędkości zadawane na oba koła
$y_zad()$ – współrzędne punktu docelowego
$y(k)$ – aktualne położenie robota
v_{max} – maksymalna prędkość.

Regulator odpowiedzialny za obrót robota:

$$\begin{aligned} u_{21}(k) &= -u_{22}(k) = K_2 \cdot e(k) \quad \text{przeciwne prędkości na oba koła} \\ e(k) &= \theta(k) \\ K_2 &= v_{max2} \\ 0 &\leqslant u_{21}(k), u_{22}(k) \leqslant v_{max2} \end{aligned}$$

gdzie:
$u_{21}(k), u_{22}(k)$ – sterowanie, czyli prędkości zadawane na oba koła
$\theta()$ – kąt pomiędzy kierunkiem jazdy robota, a prostą poprowadzoną przez robota i punkt docelowy (0, jeśli robot skierowany przodem do punktu docelowego)
v_{max2} – maksymalna prędkość podczas obrotu.

Sterowanie wysyłane do robota jest złożeniem tych dwóch ruchów:

$$V(k) = \Big[v_1(k), \quad v_2(k)\Big] = \Big[u_{11}(k) + u_{21}(k), \quad u_{21}(k) + u_{22}(k)\Big]$$

Dodatkowo, czujniki stężenia toksycznych substancji będą realizowane w skryptach Pythona, ułatwiając jej symulację. Informacje o jej położeniu i kształcie będą przekazywane do V-Repa, gdzie nastąpi ich wizualizacja.

4 Wnioski

4.1 Wykorzystane narzędzia

Symulator V-Rep okazał się narzędziem o wielu możliwościach, bardzo rozbudowanym i przydatnym, lecz także posiadającym ograniczenia:

- symulator posiada wiele gotowych modeli robotów przeznaczonych do różnego typu zadań i wyposażonych w czujniki; z drugiej strony tworzenie własnego modelu robota jest skomplikowane i czasochłonne, dlatego zdecydowaliśmy się na skorzystanie z gotowego modelu spełniającego nasze wymagania;

- program ten dobrze symuluje fizyczne oddziaływania (np. kolizje) oraz dane z czujników, jednak czasem zdarza mu się błędnie określić pozycję, orientację lub odczyt z czujnika odległości – zdarzenia te są losowe i dość rzadkie, ale wpływają negatywnie na działanie algorytmów;

- zdecydowaną zaletą jest możliwość uruchomienia serwera, który pozwala na komunikację z symulatorem z zewnątrz, co wykorzystaliśmy w projekcie;

- program podczas symulacji wymaga dość dużej mocy obliczeniowej.

Podsumowując, jest to dobre narzędzie do niewielkich projektów i symulacji oraz nauki przed programowaniem prawdziwych robotów, lecz w przypadku dużych i skomplikowanych projektów może okazać się zbyt powolny i niestabilny.

Oprogramowanie poza V-Repem było tworzone przy wykorzystaniu języka Python. Rozwiązanie to miało wiele zalet:

- tworzenie części systemu poza V-Repem umożliwiło skorzystanie z wielu bibliotek Pythona;

- każdy robot działał w osobnym wątku (podobnie chmura skażenia), dzięki czemu symulacja była bardziej zbliżona do rzeczywistości;

- wadą rozwiązania była konieczność komunikacji z V-Repem, która wprowadzała opóźnienia, częstotliwość odświeżania stanu robotów oraz symulacji w V-Repie się różniła, a komunikacja nie zawsze działała zgodnie z oczekiwaniami (np. w pewnym momencie pytając V-Repa o pozycję robota otrzymywaliśmy w odpowiedzi pozycję kamery).

4.2 Realizacja założeń

4.2.1 Symulacja środowiska i robotów

Udało się stworzyć symulację zawierającą grupę robotów mobilnych. Każdy robot składał się z:

- symulatora rzeczywistych efektorów i receptorów (realizowanych w V-Repie oraz w symulatorze chmury skażenia w Pythonie);
- wirtualnych efektorów i receptorów (realizowanych w V-Repie oraz w symulatorze robota w Pythonie);
- podsystemu sterowania (realizowanego w symulatorze robota w Pythonie).

Każdy model w V-Repie miał swój unikalny odpowiednik po stronie Pythona, z którym komunikował się na tyle często, by stan logiki robota i wizualizacja w V-Repie nie różniły się między sobą.

4.2.2 Komunikacja

Dzięki zaimplementowanej komunikacji między wątkami Pythona a V-Repem możliwe było określenie aktualnej pozycji i orientacji robota.

Dzięki czujnikom odległości dostępnym w gotowym modelu oraz zamodelowanemu czujnikowi chmury skażenia możliwa była symulacja odczytów, na podstawie których robot podejmował decyzje. Symulacja wiernie odpowiadała rzeczywistości – robot nie korzystał z żadnych danych, których by nie miał w rzeczywistości.

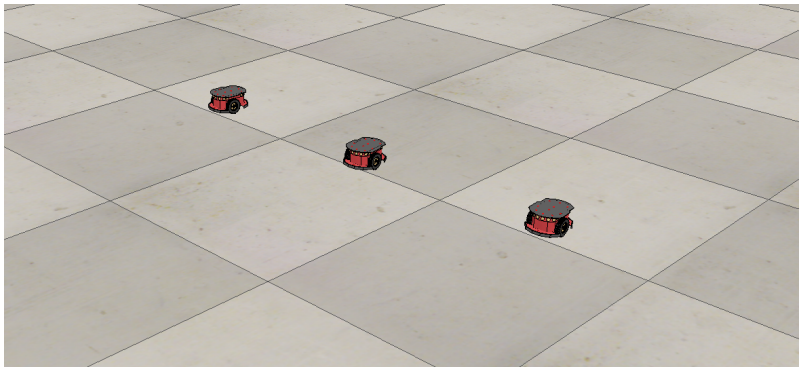
Komunikacja między robotami, zrealizowana w całości po stronie Pythona, zapewniła zachowanie spójności sieci - roboty zawsze miały informację, na jakim etapie poszukiwań znajdują się inne jednostki.

4.2.3 Realizacja zadania poszukiwania chmury

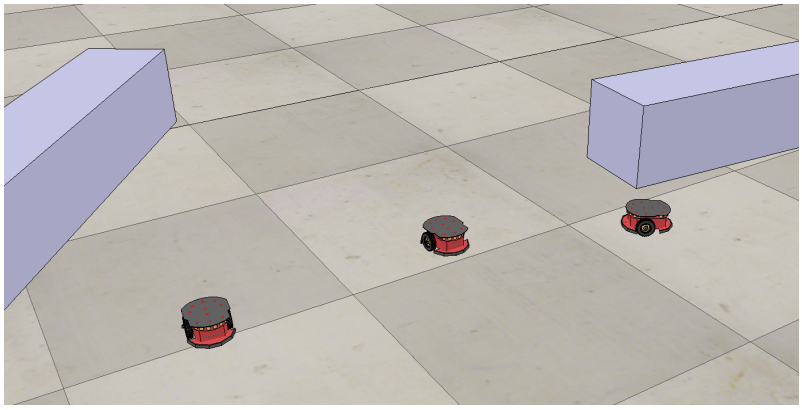
Zaproponowany algorytm przeczesywania zadanego obszaru sprawdził się bardzo dobrze. Przeczesywanie obszaru poszukiwań szerokim pasem robotów jadących w linii daje gwarancję wykrycia chmury (nawet niewielkiej), a jednocześnie nie pozwala na utracenie spójności sieci, ponieważ całą linia robotów czeka na te, które jadą wolniej (podstawą do zachowania spójności jest odpowiednio mała odległość między robotami).

Zaproponowane algorytmy omijania przeszkód sprawdziły się dla prostych i typowych przypadków, takich jak plansza z niewielkimi przeszkodami (tak, aby podczas omijania nie najechał na trasę innej jednostki).

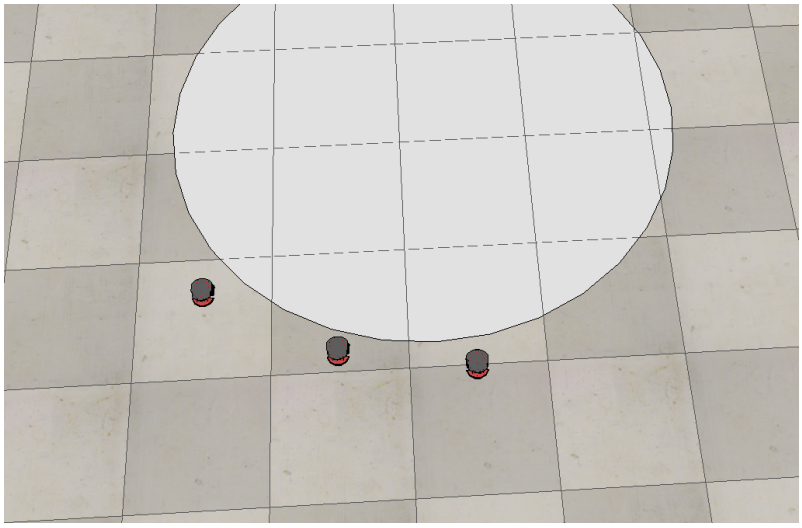
Algorytm otaczania chmury został przetestowany dla przypadku chmury nieruchomej okrągłego kształtu i sprawdził się bardzo dobrze.



Rysunek 6: Poruszanie się robotów w rzędzie



Rysunek 7: Omijanie przeszkody



Rysunek 8: Otaczanie chmury