Porównanie metod sterowania typu behawioralnego opartych na lokalnych funkcjach potencjałowych

Sebastian Kramer, Krzysztof Kroczak, Automatyka i Robotyka, specjalność: RSS

***Streszczenie*—W raporcie przedstawione zostały symulacje wybranych lokalnych metod sterowania typu behawioralnego na zaimplementowanej platformie testowej. Dokument zawiera przegląd metod sterowania, omówienie implementacji platformy, wyniki symulacji oraz zalecenia do rozbudowy platformy.**

I. Wstęp

NINIEJSZY raport przedstawiona budowę platformy testowej w środowisku MATLAB umożliwiającej symulację zachowania formacji robotów. Celem pracy było powtórzenie symulacji przedstawionych w [1] i [2] w celu ich weryfikacji, oraz porównania wyników działania obu algorytmów. Dokument składa się kolejno z przeglądu artykułów które zawierają definicje symulowanych dalej metod sterowania w rozdziale II, opisu implementacji platformy testowej wraz z opisem poszczególnych elementów składowych w rozdziale III, oraz wyników powtórzenia wybranych symulacji z [1] i [2]. Następnie przedstawione zostały uwagi dotyczące omawianej implementacji platformy, oraz krótki opis jej rozszerzania o kolejne moduły. Raport zakończony jest krótkim podsumowaniem zawierającym porównanie omawianych metod sterowania typu behawioralnego.

II. Przegląd artykułów

W [1] autorzy przedstawiają dwie metody sterowania rozproszonego - metodę pól społecznościowych (*social potential fields*), oraz jej rozwinięcie - metodę praw sprężystych (*spring laws*). Główny algorytm sterowania zakłada określenie relacji panujących między dwoma robotami, lub grupami robotów, umożliwiających obliczenie wynikowej sztucznej siły - *inverse power force*, decydującej o zachowaniu robota mobilnego. Składające się na nią wartości i kierunki poszczególnych sił składowych obliczane są w sposób rozproszony przez każdy z robotów na podstawie jego odległości od innych pojazdów i przeszkód (wykrywanych przez urządzenia komunikacyjne i sensory w zasięgu ich działania), zgodnie z regułami ustalonymi przez globalny kontroler. Metoda pól społecznościowych pozwala na arbitralny dobór parametrów wpływających na ostateczną postać siły sterującej, również w sposób niesymetryczny pomiędzy parami robotów i grup. Różnicowane są w ten sposób odległości aktywujące wzajemne odpychanie się lub przyciąganie robotów, oraz wartości sił składowych decydujących o wynikowym wektorze sterującym, zapobiegające zderzeniu się lub nadmiernemu oddaleniu się robotów od siebie. Wszystkie występujące w artykule pojazdy przydzielane są do jednej z trzech grup: liderów (*leaders*) którzy jedynie wywierają siły na inne roboty i mogą być kontrolowane zdalnie, zwykłe roboty (*ordinary robots*) które poruszają się pod wpływem wszystkich odbieranych ze środowiska informacji, oraz znaki terenowe (*landmarks*) które mają za zadanie oznaczać specyficzne fragmenty formacji lub wybrane partie terenu, jak np. przeszkody, czy koniec symulowanej mapy. W zależności od aktualnie wykonywanego zadania globalny kontroler ma możliwość dynamicznej zmiany przynależności robota do poszczególnych grup, lub modyfikacji parametrów określających sposób przeliczania zebranych danych na sterującą siłę wynikową, o ogólnej postaci

$$F_i = \sum_{j \neq i} F_{i,j}\left(X_i, X_j\right) = \sum_{j \neq i} \left[\sum_{k=1}^{l} \left(\frac{c_{ij}^{(k)}}{r_{ij}^{\sigma_{ij}^{(k)}}} \right) \left(\frac{X_j - X_i}{r_{ij}} \right) \right] \tag{1}$$

gdzie F_i to siła wynikowa powodująca przesunięcie robota i, $F_{i,j}$ to siła składowa działająca między robotami i i j, X_i, X_j to aktualne pozycje robotów i i j, l to arbitralnie określana liczba elementów sumy, $c_{i,j}^{(k)}$ i $\sigma_{i,j}^{(k)}$ to k-ty zestaw stałych odpowiadających parze robotów i i j, a r_{ij} to odległość euklidesowa między robotami i i j. [1] określa również metodologię doboru wspomnianych współczynników funkcji określającej inverse-power force:

1) W pierwszej kolejności określana jest specyfikacja żądanego zachowania - wszystkie informacje dotyczące pożądanego zachowania się formacji, odległości między robotami, ew. kształtu jaki ma przyjąć chmara (uzyskiwany jest on, w przybliżeniu, przez odpowiednie ulokowanie robotów landmarków na odpowiednich pozycjach),
2) następnie, etapowo, dobierane są parametry funkcji określające relacje między parami robotów - najpierw w obrębie poszczególnych grup, a następnie między kolejnymi grupami.
3) Opcjonalnie określane są również parametry definiujące relacje po wystąpieniu określonego zdarzenia, np. upłynięciu zadanego czasu, czy wystawienia przez danego robota określonej flagi.

W [1] autorzy opisują również rozwinięcie metody pól społecznościowych - metodę praw sprężystych (*spring power law*. Pozwala ona, w przeciwieństwie do metody opisanej powyżej, na precyzyjne określanie pożądanych struktur, a nie jedynie na równomierną dystrybucję robotów na zadanym terenie. Kluczowym elementem całej koncepcji jest nieskierowany graf, na którego wierzchołki składają się roboty, a którego krawędzie reprezentują oddziaływania między robotami. Sterowanie w tej metodzie polega na poszukiwaniu takiego grafu (takiego

Dokument sporządzony 10 czerwca 2013

ułożenia robotów), którego energia w stanie ustalonym będzie równa energii pożądanej predefiniowanej struktury, tzn. takiego grafu w którym wszystkie wierzchołki (roboty) znajdują się w swoich stanach equilibrum, co ma miejsce gdy

$$F_i = \sum_{(i,j)\in E} F_{ij} = \sum_{(i,j)\in E} k_{ij}(r_{ij} - l_{i,j})\frac{p_i - p_j}{r_{ij}} = 0 \quad (2)$$

gdzie F_i oznacza wypadkową siłę sterującą robota i, E oznacza zbiór wszystkich krawędzi grafu (zbiór wszystkich sił działających między parami robotów), k_{ij} oznacza stałą związaną z siłą działającą między parami robotów, r_{ij} oznacza aktualną odległość euklidesową między robotami i i j, l_{ij} oznacza pożądaną odległość euklidesową między robotami i i j a p_i i p_j oznaczają aktualne pozycje robotów i i j. Unikalność struktury o minimalnej energii wiąże się z pojęciem spójności grafu - autorzy [3] wykazują, że cecha ta jest zapewniona praktycznie zawsze, gdy omawiany graf jest $d+1$-spójny (gdzie d oznacza ilość wymiarów przestrzeni na których opisany jest graf), tzn. gdy usunięcie mniej niż $d+1$ dowolnych wierzchołków grafu nie powoduje utraty spójności grafu (wciąż istnieje możliwość znalezienia ścieżki łączącej dwa dowolne wierzchołki grafu). Projektowanie formacji i relacji pomiędzy poszczególnymi jej elementami wymaga więc najpierw stworzenia $d+1$-spójnego grafu, oraz określenia $d+1$ robotów liderów którzy są kontrolowani zdalnie. Wszystkie pozostałe roboty kontrolowane są przez prawa sprężystości zdefiniowane pomiędzy poszczególnymi parami robotów, poruszając się w celu zmniejszenia wypadkowych sił oddziałujących na każdego z nich. Poszczególne relacje między robotami (krawędzie grafu) dodawane są aż do momentu w którym cały graf stanie się $d+1$-spójny (dopóki każdy z wierzchołków - robotów nie posiada przynajmniej $d+2$ krawędzi - sił między robotami).

Autorzy [2] proponują rozszerzenie metody zawartej w [1] - *Gradient climbing law*. Relacje między poszczególnymi robotami są tu analogiczne do tych z metody pól społecznościowych (zmodyfikowane o ograniczenie maksymalnego dystansu oddziaływania robotów na siebie), zebrane są jednak dodatkowo w strukturę - ciało wirtualne (*virtual body*), czyli zbiór połączeń i relacji między robotami występującymi w systemie. Wirtualne ciało jest używane do rozdzielania zadania stabilizacji formacji robotów od zadania dążenia formacji do wykonania zadanego celu. Obliczane, na podstawie informacji otrzymywanych z formacji przez globalny kontroler, modyfikacje położenia wirtualnego ciała pociągają za sobą ruch robotów, dążących do pozostania w określonej odległości od innych pojazdów i na określonym przez wspomnianą strukturę miejscu. Samo ciało poruszać się może np. zgodnie z proponowanym w artykule kierunkiem najgłębszego spadku aproksymaty gradientu pola skalarnego.

III. Implementacja

W celu weryfikacji symulacji z [1] i [2] zaimplementowana została, w środowisku MATLAB, platforma testowa, schematycznie przedstawiona na rys. 1 (schemat ogólny) oraz na rys. 9. Cała implementacja zrealizowana jest w sposób obiektowy [6], z zachowaniem zasad dotyczących enkapsulacji (hermetyzacji) danych. Poszczególne bloki schematu oznaczają zaimplementowane klasy i relacje między nimi, przy czym strzałka wypełniona oznacza, ze obiekt na który wskazuje zawiera w sobie obiekt z którego wychodzi strzałka, natomiast strzałka niewypełniona oznacza, ze obiekt na który wskazuje jest obiektem bazowym dla obiektu z którego strzałka wychodzi. Takie implementacja nie ogranicza ilości symulowanych robotów, umożliwiając jednocześnie szerszą parametryzację i dostosowanie symulacji do potrzeb. Modułowa konstrukcja umożliwia rozwijanie platformy o inne modele robotów, sił sterujących, regulatorów i całych symulacji bez konieczności ponownego przechodzenia procesu projektowania od nowa, co umożliwia rozwijanie platformy w dowolnym kierunku, zależnie od potrzeb i możliwości. Dalsza część tego punktu opisuje poszczególne klasy i skrypty wykorzystywane w symulacjach.

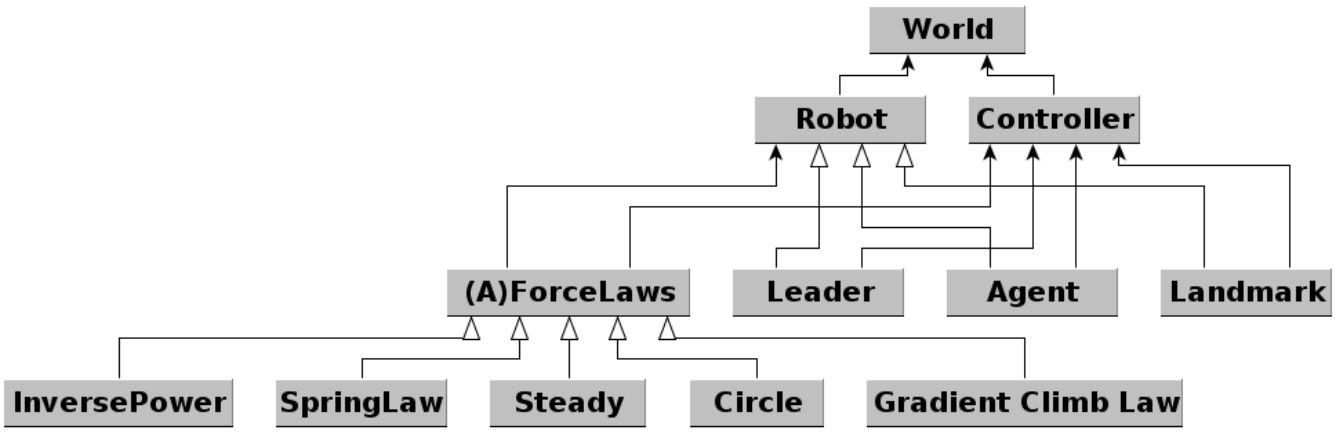
A. Skrypt zarządzający

Wszystkie przykładowe symulacje są konfigurowane i zarządzane przez plik `forceLawTest.m`. W jego nagłówku definiowane są parametry symulacji - ilość poszczególnych robotów (pola howManyLeaders, howManyAgents i howManyLandmarks), długość symulacji (ilość iteracji, pole howManyIterations), częstotliwość wykonywania przez roboty pomiarów odległości od innych pojazdów i generowania sił sterujących (pole howOfterForceCalculation, roboty wykonują ruch w kierunku pożądanej lokalizacji w każdej iteracji), rozmiar mapy na której zostaną losowo rozstawione roboty (pole worldSize), oraz metodę sterowania robotów agentów (pole useForce). Siły używane przez danego robota przy symulacji zadanej metody sterowania określane są przy tworzeniu obiektów robotów (jako jeden z parametrów konstruktora).

B. Klasy World i Controller

Klasa ***WORLD*** jest klasą łączącą wszystkie pozostałe. Zawiera ona definicje wszystkich robotów (pola aAgents, aLandmarks i aLeaders), kontrolera (pole controller), oraz uchwyt do panelu na którym wyświetlane są wyniki symulacji (pole fig). Jej obecność jest wymagana przed rozpoczęciem dodawania innych obiektów do symulacji. Implementuje ona m.in. metody ***addRobots*** pozwalającą na dodawanie wcześniej utworzonych obiektów dziedziczących z klasy ***ROBOT*** do symulacji, oraz ***update*** odświeżającą okno z wynikami symulacji. W trakcie tworzenia instancji klasy ***WORLD*** tworzony jest automatycznie obiekt klasy ***CONTROLLER*** zawierający predefiniowane parametry pozwalające na przeliczanie odległości robotów na siły sterujące pojazdy.

CONTROLLER przechowuje ostatnie znane pozycje wszystkich robotów (pola statLeader, statAgent, statLandmark), nazwy wszystkich grup na które podzielone zostały występujące w symulacji roboty (pole groups), oraz parametry wykorzystywane do przeliczania sił w metodach social fields law (pole forces) i spring force law (pole springForces). Parametry te można rozszerzyć poza wartości domyślnie zaimplementowane w konstruktorze przez wykorzystanie funkcji ***addForce*** i ***addSpringForce***. Metoda ***update*** zapisuje wartości ostatnio zmierzonych przez roboty potencjałów pola skalarnego (pole potentials), oraz ostatnio wyliczoną, metodą



Rysunek 1. Ogólny schemat koncepcyjny implementacji platformy testowej. Strzałka wypełniona oznacza, że obiekt na który wskazuje zawiera w sobie obiekt z którego wychodzi strzałka. Strzałka niewypełniona oznacza, że obiekt na który wskazuje jest obiektem bazowym dla obiektu z którego wychodzi strzałka.

najmniejszych kwadratów, aproksymację rzeczywistych wartości pól skalarnych mierzonych przez roboty i rzeczywistego gradientu generowanego przez te wartości (pole xLs). Wielkości te wymagane są w procesie obliczania gradientu pola skalarnego, który ma wskazywać kierunek przemieszczania się wirtualnej struktury. Modyfikacja ta nie została jednak zaimplementowana ze względu na problemy z interpretacją zmiennej s występującej w [2].

C. Klasy sił

Wszystkie siły sterujące przypisywane do robotów definiowane są przez pochodne abstrakcyjnej klasy ***FORCELAWS***. Definiuje ona m.in. kierunek i wartość siły wynikowej (pola direction, value), krok o jaki przemieszcza się robot w wyniku działania siły (pole step) i skrótową nazwa wykorzystywanej metody sterowania (pole forceName). Klasa pochodna ***INVERSEFORCE*** implementuje pole groupForces zawierające struktury określające relacje robota z innymi pojazdami (odpowiednie definicje relacji z kontrolera) określane w metodzie social fields law. Implementuje również wirtualną metodę ***update*** klasy ***FORCELAWS*** pozwalającą na obliczanie wartości wynikowej siły sterującej zgodnie z (1). Analogicznie, klasa ***SPRINGFORCE*** implementuje pole springForces przechowujące struktury, oraz metodę ***update*** wyliczającą sterowanie zgodnie z (2). Klasa ***GRADCLIMBLAW*** implementuje pola hDist i dDist, definiujące odległości między agentami i liderami, oraz między samymi agentami, przy których aktywowane są siły sterujące, oraz pola fh_par i fl_par określające parametry funkcji przeliczającej błąd pozycji robota względem innego pojazdu, na siłę sterującą. Relacja ta w [2] nie została zdefiniowana analitycznie, przyjęta została więc postać funkcji $f_i = a_0 x + a_1 x^2 + \frac{a_3}{x}$, gdzie f_i to składowa sterującej siły wynikowej, a_0, a_1 i a_3 to parametry dobierane w zależności od zadanych warunków, a x to aktualna odległość robota od innego, aktualnie analizowanego pojazdu. Analizowane w tej metodzie pole skalarne zostało zaimplementowane w postaci sił generowanych przez nieruchome roboty landmarki - ich obecność powoduje generowanie różnych wartości pola, zależnie od dystansu od znacznika. Dodatkowo zaimplementowane zostały również klasy ***STEADY*** i ***CIRCLE*** które powodują odpowiednio nie poruszanie się robota, i poruszanie się robota po okręgu o wcześniej zadanym promieniu.

D. Klasy robotów

Wszystkie klasy robotów definiowane są jako pochodne klasy ***ROBOT***. Definiuje ona pola i metody wspólne dla wszystkich robotów (zaimplementowane klasy ***AGENT***, ***LANDMARK*** i ***LEADER*** różnią się od siebie jedynie domyślnymi przypisaniami do grup). Klasa ***ROBOT*** określa nazwy grup do których należy robot (pole groups), aktualny stan obiektu (pozycję w osiach X i Y oraz kąt orientacji φ, pole state) oraz aktualne prędkości liniowe robota w poszczególnych osiach (pole speed). Każdy z robotów przechowuje również uchwyty do siły sterującej (pole forceLaw), kontrolera (pole controller), oraz świata (pole world). Klasa ***ROBOT*** definiuje metody ***getDistToRobot*** zwracającą odległość euklidesową od innego robota, ***getPotentialField*** powodującą wykonanie przez robota pomiaru wartości pola skalarnego (używane w metodzie gradient climbing), ***makeMeasurement*** pobierającą informację o stanach pozostałych robotów, oraz metodę ***update*** powodująca wyliczenie nowego pożądanego stanu robota na podstawie informacji ze środowiska. Nowo określony wektor stanu osiągany jest przez robota przez wykonywanie metody ***steerToDestState*** która uruchamia generowanie sygnałów sterujących przez regulatory.

E. Klasa regulatora PID

Klasa ***ROBOT*** implementuje również dwa regulatory typu PID o wcześniej zdefiniowanych nastawach - jeden z nich odpowiada za regulację orientacji robota do kierunku zadanego celu (aktualnie pożądanej pozycji), a drugi za regulację dystansu od zadanej pozycji (regulacja odpowiednio zmiennej stanu φ i odległości euklidesowej $|x^2 + y^2|$). Sama regulacja odbywać się może na dwa różne sposoby:

Regulacja ciągła
: Oba regulatory PID pracują równolegle, generując sygnały sterujące prędkością liniową i kątową robota. O wyborze, z dwóch predefiniowanych wartości, zadawanej prędkości liniowej decyduje aktualny błąd orientacji podawany w stopniach - dla dużego błędu robot porusza się powoli, a po zmniejszeniu się uchybu orientacji do wartości mniejszej niż wartość z pola toleratedError regulatora (określa ona dopuszczalny błąd regulacji odpowiednio orientacji i położenia) prędkość liniowa robota zwiększa się. Takie rozwiązanie pozwala ograniczyć obszar potrzebny robotowi na zorientowanie się na zadany cel (ograniczeniu ulega promień okręgu po którym porusza się robot, gdy jest on źle skierowany).

Regulacja trzy etapowa
: Przy wybraniu tej metody regulacji oba regulatory pracują wymiennie. Pierwsza faza regulacji polega na obrocie robota wokół własnej osi i zorientowania go na zadany cel. Następnie uaktywniany jest regulator prędkości liniowej który współpracując z regulatorem orientacji robota doprowadza go do zadanego punktu (z dokładnością do tolerowanych błędów toleratedError). Regulator prędkości liniowej jest następnie dezaktywowany, a regulator orientacji generuje sygnał prędkości kątowej pozwalający robotowi osiągnąć pożądaną orientację końcową.

Ilość regulatorów PID oraz różne metody sterowania wynikają z rodzaju układu regulacji - MIMO (ang. *multiple inputs, multiple outputs*). Z uwagi na brak doświadczenia w regulacji w/w układów, problem sterowania został rozdzielony, umożliwiając wykorzystanie regulatorów układów SISO.

Sam regulator PID implementowany jest jako zdyskretyzowana wersja regulatora ciągłego. Jego parametry dobrane zostały w oparciu o odpowiedź skokową modelu, przybliżającego układ regulacji orientacji i położenia robota przez elementy inercyjne I rzędu z opóźnieniem, zgodnie z algorytmem QDR [4]. Obliczanie sygnału sterującego odbywa się w metodzie ***genU***. Klasa ***PID*** definiuje także metodę ***reset*** przywracającą wszystkie ustawienia regulatora do początkowych wartości (reset regulatora odbywa się po każdym zadaniu kolejnej pozycji pożądanej).

IV. Wyniki symulacji

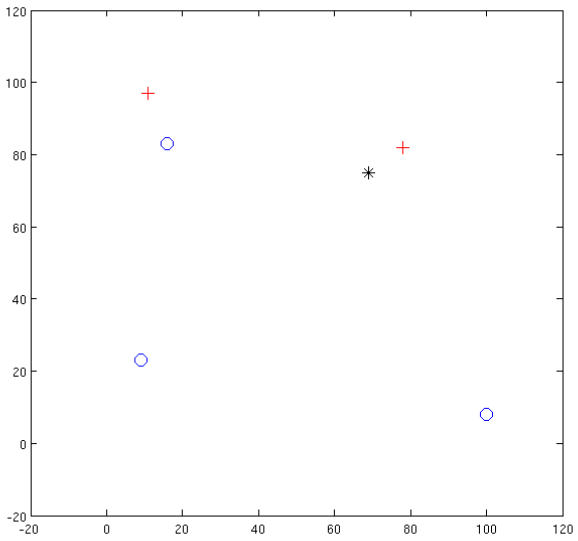
Wszystkie przygotowane symulacje sił sterujących kontrolowane są przez skrypt opisany w III-A. Modyfikacji parametrów symulacji należy dokonywać w nagłówku skryptu, oznaczonym etykietą `Configuration`. Ewentualna zmiana sił sterujących poszczególnymi robotami w odpowiednich symulacjach powinna zostać wprowadzona przez modyfikację wywołań konstruktorów klas poszczególnych typów robotów poniżej. Możliwe jest również różnicowanie sił używanych przez roboty tego samego typu, poprzez utworzenie instancji tych klas poza zaproponowanymi pętlami. W przedstawionych w następnych podpunktach wynikach symulacji rozmiary znaczników robotów zostały powiększone dla lepszej widoczności. Na wszystkich rysunkach krzyżykami oznaczone zostały roboty typu landmark, kółkami roboty typu agent, a gwiazdkami roboty typu leader. Wszystkie symulacje były przeprowadzane przy ustawieniu sterowania robotów landmark na regulację ciągłą, a robotów leader i agent na regulację trzy etapową.

A. Social potential law, inverse power force

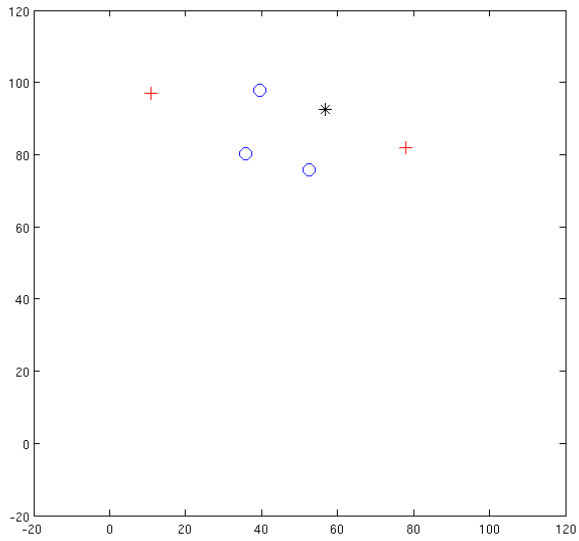
Przyjęte przez autorów na potrzeby symulacji założenia zakładają ruch wszystkich robotów w celu równomiernego ich rozproszenia (ruch tylko na podstawie składowej siły wynikowej określanej jako globalna) - nie określany jest cel do którego ma zbiegać formacja (pomijana jest składowa siły wynikowej lokalna). Zakładany jest również rozmiar przeszkód występujących na nieznanym terenie dużo mniejszy od wielkości całego symulowanego terenu. Każdy z cykli symulacji zakłada pobranie informacji z czujników i urządzeń komunikacyjnych przez każdego z robotów, obliczenie przez niego globalnej siły sterującej i wykonanie pod jej wpływem (zgodnie z jej kierunkiem) ruchu o stały krok. Przedstawione symulacje ograniczają postać funkcji obliczającej siłę sterującą do dwóch elementów $f(r) = \frac{-c_1}{r^{\sigma_1}} + \frac{c_2}{r^{\sigma_2}}$ gdzie $\frac{-c_1}{r^{\sigma_1}}$ odpowiada za odpychanie się robotów od siebie gdy zbytnio się do siebie zbliżą, a $\frac{c_2}{r^{\sigma_2}}$ powoduje wzajemne przyciąganie się robotów do siebie w przypadku ich zbytniego oddalenia się. Przygotowane do powtórzenia zostały dwie pierwsze symulacje przedstawione w [1].

1) Równomierne rozproszenie: Celem oryginalnej symulacji było równomierne rozproszenie robotów. Przygotowana została jednak wersja zmodyfikowana symulacji, implementująca w robotach typu landmark sterowanie siłą steady. Oznacza to, że wszystkie pozostałe roboty sterowane siłą inverse power force zostaną rozdystrybuowane równomierne pomiędzy nieruchome, losowo rozmieszczone landmarki. Zastosowane do symulacji domyślne parametry inverse power force są definiowane jednakowo pomiędzy wszystkimi rodzajami robotów jako `c: [-20 1] , sig: [2 1]`, odpowiadających wartościom parametrów odpowiednio c_1, c_2, σ_1, σ_2. W symulacji zastosowano dwa roboty landmarki, trzy roboty agenty i jednego robota lidera. Wyniki symulacji przedstawione zostały na rys. 2. Roboty agenci i lider rozstawilali się równomiernie pomiędzy dwoma nieruchomymi znacznikami, lub tworzyły równomierną dystrybucję w której landmarki składały się na jeden z boków.

2) Zachowanie chronienia obiektu: Drugą przedstawioną w [1] symulacją było chronienie wybranego robota przez pozostałe, realizowane poprzez zbieranie się pojazdów wokół znacznika. Ta symulacja została powtórzona bez modyfikacji. Do jej realizacji konieczna była zmiana jednej z relacji pomiędzy robotami: `to: landmark, from: agent, c: [-80 1], sig: [2 0.1]`. Wszystkie pozostałe relacje miały wartości domyślne. W symulacji zastosowano jednego nieruchomego (sterowanego siłą steady) robota landmarka, oraz 20 robotów agentów. Wyniki symulacji przedstawione zostały na rys 3. Roboty agenci przyciągani byli przez nieruchomy cel, po czym okrążały go tworząc pierścień o równomiernych odstępach między robotami.

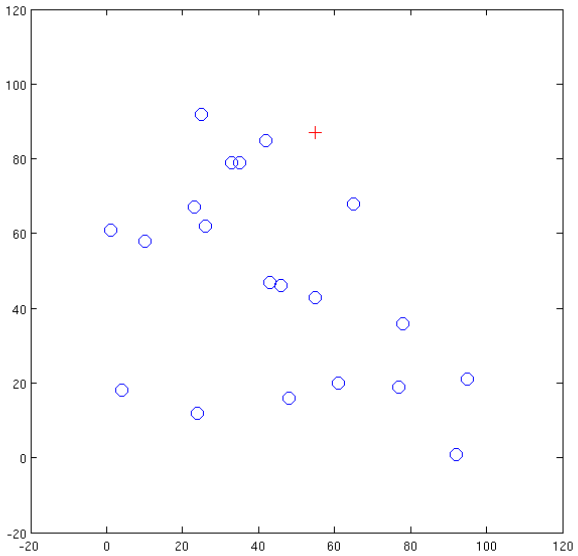


(a) Stan początkowy symulacji

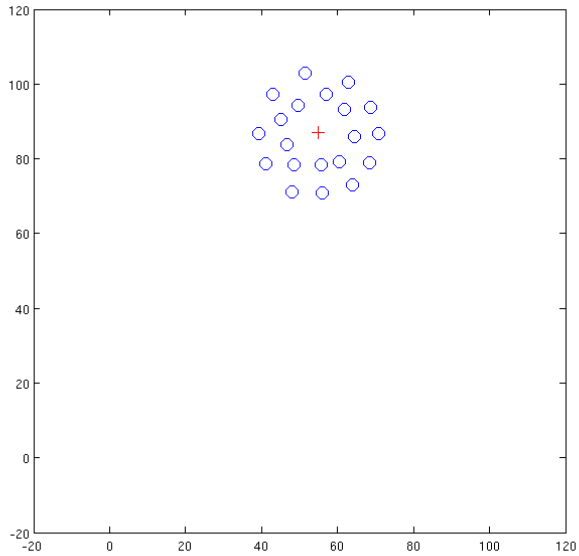


(b) Stan symulacji po 100 iteracjach

Rysunek 2. Wyniki symulacji równomiernego rozproszenia robotów przy wykorzystaniu inverse power law. Roboty landmark w trakcie symulacji były nieruchome. Pozostałe pojazdy poruszały się pod wpływem sił inverse power law.



(a) Stan początkowy symulacji



(b) Stan symulacji po 100 iteracjach

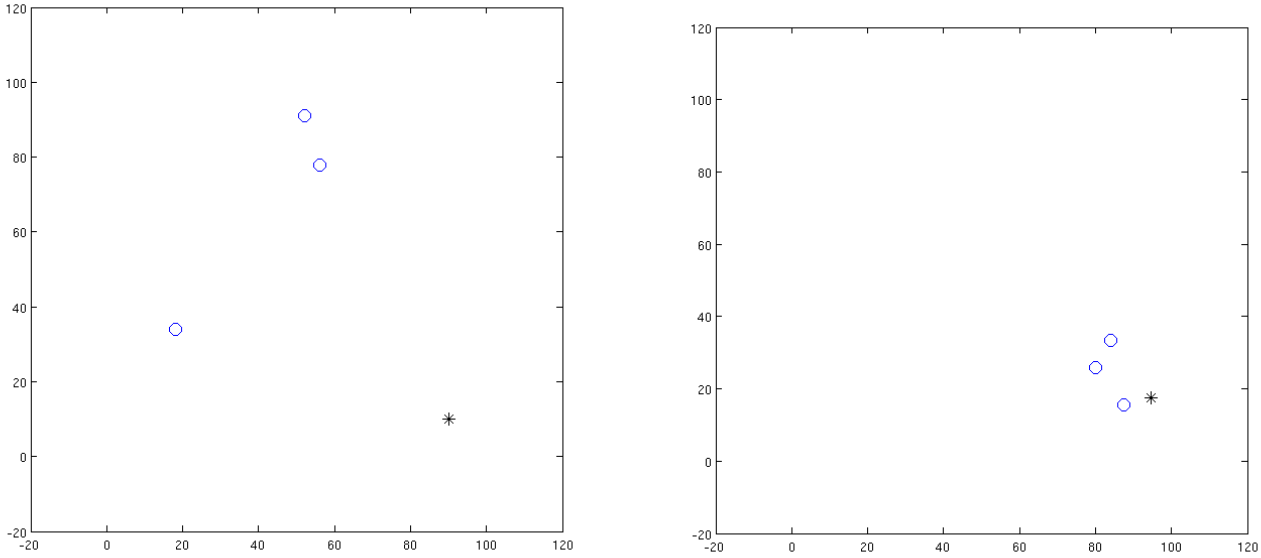
Rysunek 3. Wyniki symulacji chronienia robota przy wykorzystaniu inverse power law. Robot landmark w trakcie symulacji był nieruchomy. Pozostałe pojazdy poruszały się pod wpływem sił inverse power law.

B. Siły sprężyste

Przeprowadzona została również dwie symulacje metody sił sprężystych przedstawionej w [1] jedynie analitycznie. Określane tam relacje między robotami są symetrycznie określane pomiędzy konkretnymi robotami, przedstawiona została więc tylko ogólna reguła. W trakcie symulacji parametry sił sterujące ustawione zostały następująco:

```
to: agent1, from: agent2, k: 5, l: 15
to: agent1, from: agent3, k: 5, l: 15
to: leader1, from: agent1, k: 15, l :10
to: leader1, from: agent3, k: 15, l: 20
```

gdzie `k` oznacza amplitudę danej siły, a `l` określa pożądaną odległość między odpowiednimi robotami. W obu symulacjach zastosowano po trzy roboty agenty i jednego robota leadera. W



(a) Stan początkowy symulacji

(b) Stan symulacji po 100 iteracjach

Rysunek 4. Wyniki symulacji precyzyjnego ustawiania robotów przy wykorzystaniu spring force law. Robot leader w trakcie symulacji poruszał się zgodnie z wymuszeniami generowanymi przez siłę circle. Pozostałe pojazdy poruszały się pod wpływem sił spring forces.

pierwszej z nich robot leader poruszał się przez wymuszenia generowane przez siłę force (ruchy koliste), w drugiej za sprawą siły steady (pozostawał w bezruchu). Powyższe przykładowe ustawienia nie biorą pod uwagę obecności robotów landmarków. Wyniki symulacji przedstawione zostały odpowiednio na rys. 4 i rys. 5. Widoczne są na nich analogiczne ustawienia robotów agentów w trójkąt o zadanych bokach, oraz ustawienie odpowiednich agentów w zadanej odległości od lidera. Przy symulacji z poruszającym się liderem agenci wciąż zmieniali swoje pozycje aby zminimalizować błąd ustawienia. W drugiej wersji symulacji formacja robotów zbiegła do stabilnego ustawienia po ok. 50 iteracjach.

C. Gradient climbing

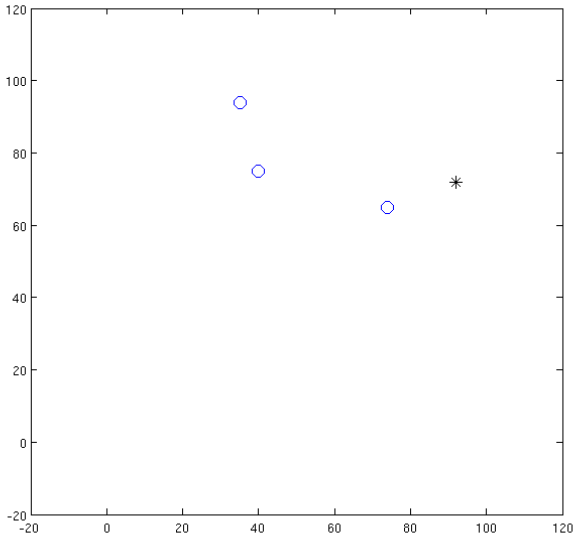
Wykonana została również, analogiczna do poprzednich, symulacja działania metody gradient climbing law. Z uwagi na opisany w III brak implementacji poruszania się struktury wirtualnej, implementowani liderzy sterowani byli siłą circle. Występujące w symulacji roboty landmarki nie biorą bezpośredniego udziału w sterowaniu robotami agentami - mają one za zadanie różnicować wartości pola skalarnego, na podstawie których wyliczana jest aproksymata gradientu sterującego wirtualną strukturą. Symulacja ogranicza się więc do utrzymywania równomiernego rozproszenia robotów agentów i ich podążania za robotem liderem, przy jednoczesnym wykonywaniu pomiarów wartości pola skalarnego w aktualnie zajmowanej przez nie pozycji. W każdej iteracji kontroler otrzymuje wyniki tych pomiarów, i oblicza aktualną aproksymację rzeczywistych wartości pola i gradientu pola skalarnego do sterowania wirtualną strukturą. W symulacji zastosowano jednego robota lidera, dwa roboty landmarki oraz sześć robotów agentów. Wyniki symulacji przedstawione zostały na rys. 6. Roboty agenci najpierw zbiegają do równomiernego rozproszenia, a następnie rozpoczynają śledzenie robota lidera. Osiągnięcie tego stanu wymagało ok. 75-100 iteracji. Wirtualny kontroler w każdej iteracji obliczał, na podstawie pomiarów przeprowadzanych przez roboty, wartości gradientu pola skalarnego możliwego do wykorzystania do sterowania strukturą wirtualną.

D. Symulacja ruchu pojedynczego robota

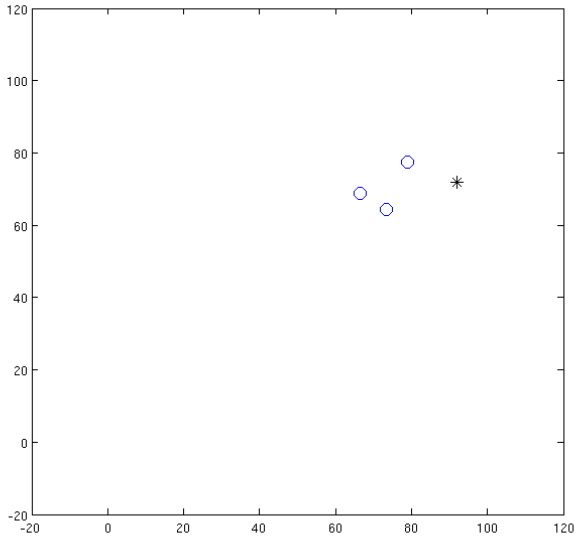
Zaimplementowana platforma daje również możliwość testowania różnych modeli robotów oraz ich regulatorów. W celu przedstawienia przykładowej realizacji tego zadania przygotowany został dodatkowy skrypt `singleRobotTest.m`. Zawiera on definicję pojedynczego robota (wraz z odpowiednim łańcuchem znakowym w konstruktorze dezaktywującym konieczność tworzenia instancji klas ***WORLD*** i ***CONTROLLER***. Przygotowana symulacja przeprowadza robota od pozycji początkowej do zadanego stanu końcowego korzystając jedynie z zintegrowanego z robotem regulatora PID. Regulacja odbywa się zgodnie z zasadami opisanymi w III-E i IV. Porównanie wyników wspomnianej symulacji dla robotów typu agent lub leader (gdzie domyślnie implementowana jest trzy etapowa metoda regulacji) i robotów landmark (domyślnie regulacja ciągła) pozwala zauważyć różnice między obiema zaproponowanymi metodami. Trajektoria ruchu do punktu robota z regulacją metodą ciągła przedstawiona została na rys. 8, a z regulacją metodą trzy etapową na rys. 7.

V. Uwagi do symulacji i implementacji

Zaimplementowana platforma testowa umożliwia testy różnych rodzajów sił, struktur komunikacyjnych (scentralizowanych, rozproszonych), modeli robotów, regulatorów oraz

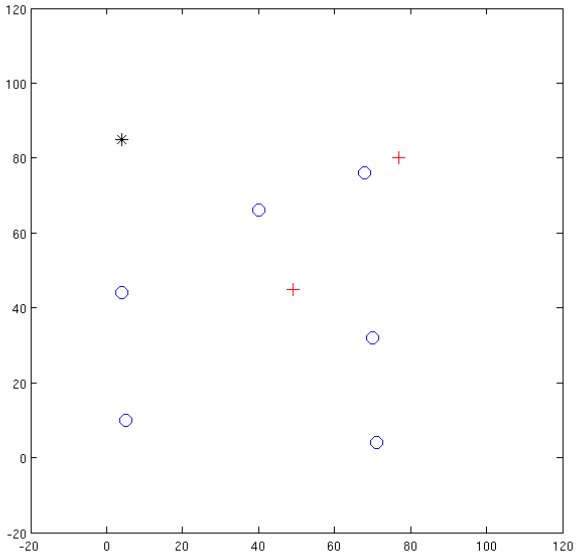


(a) Stan początkowy symulacji

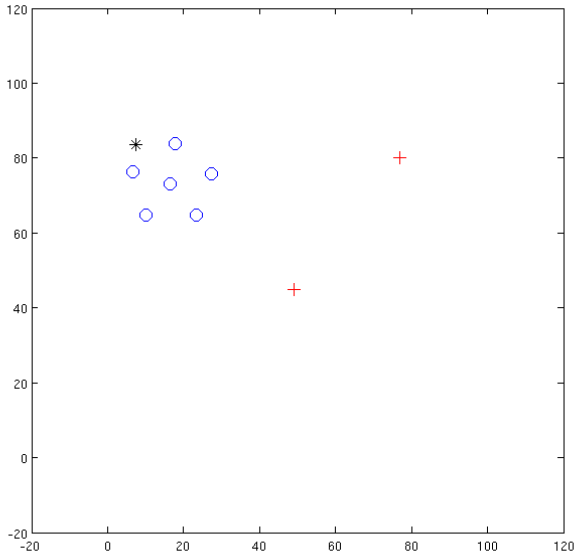


(b) Stan symulacji po 50 iteracjach

Rysunek 5. Wyniki symulacji precyzyjnego ustawiania robotów przy wykorzystaniu spring force law. Robot leader w trakcie symulacji pozostawał w spoczynku zgodnie z wymuszeniami generowanymi przez siłę steady. Pozostałe pojazdy poruszały się pod wpływem sił spring forces.



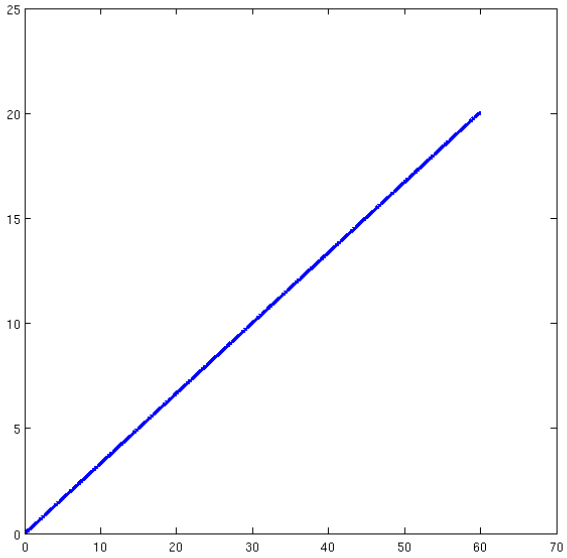
(a) Stan początkowy symulacji



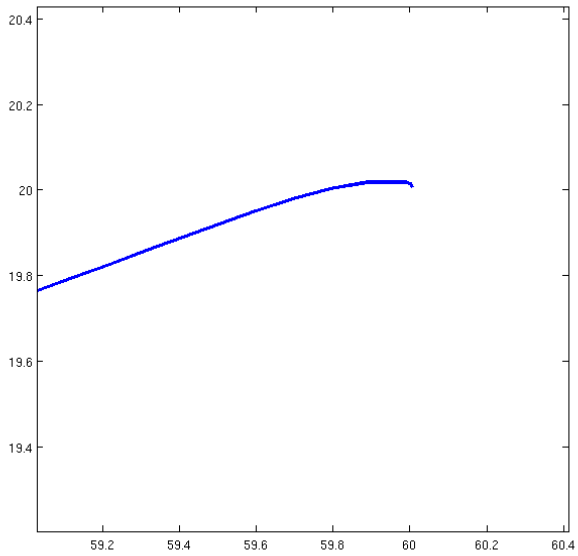
(b) Stan symulacji po 100 iteracjach

Rysunek 6. Wyniki symulacji ustawiania robotów przy wykorzystaniu nie w pełni zaimplementowanego gradient climbing force law. Robot leader w trakcie symulacji poruszał się zgodnie z wymuszeniami generowanymi przez siłę circle. Nieruchome roboty landmarki umieszczone były jedynie do generowania różnych wartości pola skalarnego, nie były brane pod uwagę przez siły sterujące liderem i agentami. Roboty agenty poruszały się za sprawą wymuszeń gradient climbing force law.

modyfikację parametrów symulacji odpowiednio do potrzeb. Modułowa konstrukcja umożliwia modyfikację wspomnianych parametrów przez implementację dodatkowych klas definiujących metody o odpowiedniej nazwie, np. nowego implementacja regulatora powinna posiadać pole toleratedError, oraz metodę ***genU***, nowa metoda sterowania (siła sterująca) powinna dziedziczyć z klasy ***FORCELAWS*** i definiować abstrakcyjną metodę ***update***, a zmiana zaproponowanych metod regulacji ciągłej i trzy etapowej może nastąpić przez stworzenie klasy dziedziczącej z klasy ***ROBOT***, redefiniującej metodę ***steerToDestState***. Możliwa jest również ingerencja w strukturę całej platformy, np. poprzez dodanie dodatkowej klasy

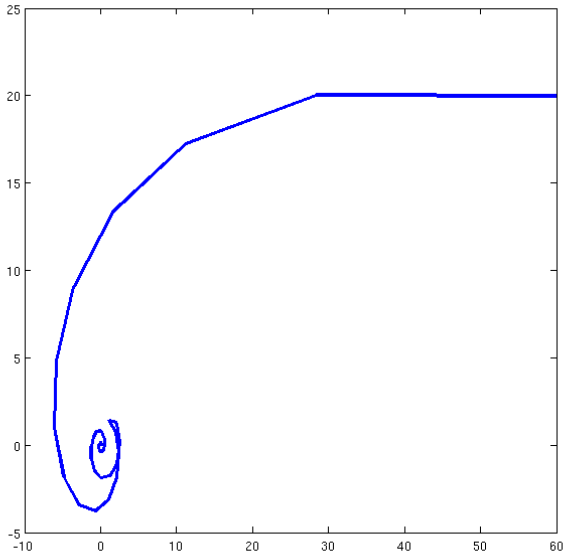


(a) Pełna trajektoria ruchu robota przy sterowaniu do punktu



(b) Początkowe modyfikacje trajektorii robota landmarka przy sterowaniu do punktu. Widoczne jest zmniejszenie promienia po którym porusza się robot w przypadku dużej wartości uchybu orientacji względem zadanego celu, i jego zwiększenie po osiągnięciu wartości mniejszej od wcześniej ustalonej.

Rysunek 8. Trajektorie ruchu (pełna i początkowa) robota sterowanego metodą regulacji ciągłej.



Rysunek 7. Trajektoria ruchu robota sterowanego metodą regulacji trzy etapowej.

pośredniczącej między robotami a kontrolerem, definiującej parametry relacji między robotami dla wybranego rodzaju zachowania formacji w odpowiednim czasie (możliwy sposób implementacji dynamicznych zmian formacji).

W trakcie implementacji należy zwrócić szczególną uwagę na kilka mankamentów związanych z obiektowością oferowaną przez środowisko MATLAB. Tworzone metody powinny, pośrednio lub bezpośrednio, dziedziczyć z klasy ***HANDLE***, zgodnie z [5]. Zapewni to zapisywanie się wartości pól i wyników działania metod po utworzeniu instancji klasy.

Zaprezentowana metoda symulacji jest niestety wrażliwa na sposób jej przerywania. Zamknięcie okna symulacji przed jej zakończeniem z poziomu przestrzeni roboczej (np. skrótem klawiszowym CTRL+C) powoduje generowanie błędu `Error using handle.handle/set Invalid or deleted object.` przy próbie otwarcia kolejnych okien symulacji. Konieczna jest wtedy jakakolwiek modyfikacja pliku `World.m` i jej zapis (np. usunięcie dowolnego znaku tekstowego, zapis pliku, przywrócenie stanu sprzed usunięcia i ponowny zapis). Odświeżana jest wtedy przestrzeń pamięci środowiska przechowującą uchwyt do definiowanego z klasie ***WORLD*** okna symulacji. Poza tym nieudogodnieniem, wszelkie modyfikacje kodu oraz czyszczenie przestrzeni roboczej mogą być uruchamiane bez konieczności wprowadzania innych modyfikacji - konieczne jest tylko pozostawianie uruchomionego okna symulacji, lub odświeżenie pliku `World.m` w przypadku i gdy zostało ono zamknięte.

VI. Podsumowanie

Zaimplementowana została platforma umożliwiająca testy różnych metod sterowania oraz różnych modeli robotów mobilnych i regulatorów, a także różnych struktur symulacji. Modułowa konstrukcja powoduje, że dodanie kolejnej siły sterującej, lub zwiększenie dokładności modeli sprowadza się jedynie do definicji dodatkowej klasy, nie jest potrzebna modyfikacja wszystkich innych plików.

Metoda pól społecznościowych pozwala na równomierne rozproszenie robotów na zadanym terenie, definiując jedynie kilka relacji pomiędzy grupami do których należą roboty. Metoda ta sprawdza się dopóki nie wymagamy przyjęcia przez formację określonego kształtu, np. ustawienia się robotów w jednej linii. Co prawda prostsze kształty mogą być przybliżane przez odpowiednie ustawienie robotów znaczników, jednak dokładne definiowanie struktur może być implementowane przy użyciu metody sił sprężynowych. Wymaga ona jednak określenia większej ilości relacji między robotami. Obie metody zdają się dobrze sprawdzać w warunkach symulacyjnych. Ostatnie z porównywanych praw sterowania - gradient climbing law, bazuje na wspomnianej metodzie praw społecznościowych. Zakłada ono, tak jak w social potencial law, równomierne rozproszenie robotów za sprawą inverse force power. Modyfikacja polega na wymuszaniu położenia poszczególnych pojazdów mobilnych w ramach poruszającej się wirtualnej struktury obliczanej przez kontroler.

DODATEK A
SZCZEGÓŁOWY SCHEMAT PLATFORMY TESTOWEJ

LITERATURA

[1] John H. Reif and Hongyan Wang, *Social potential fields: A distributed behavioral control for autonomous robots*, Robotics and Autonomous Systems, 1999.

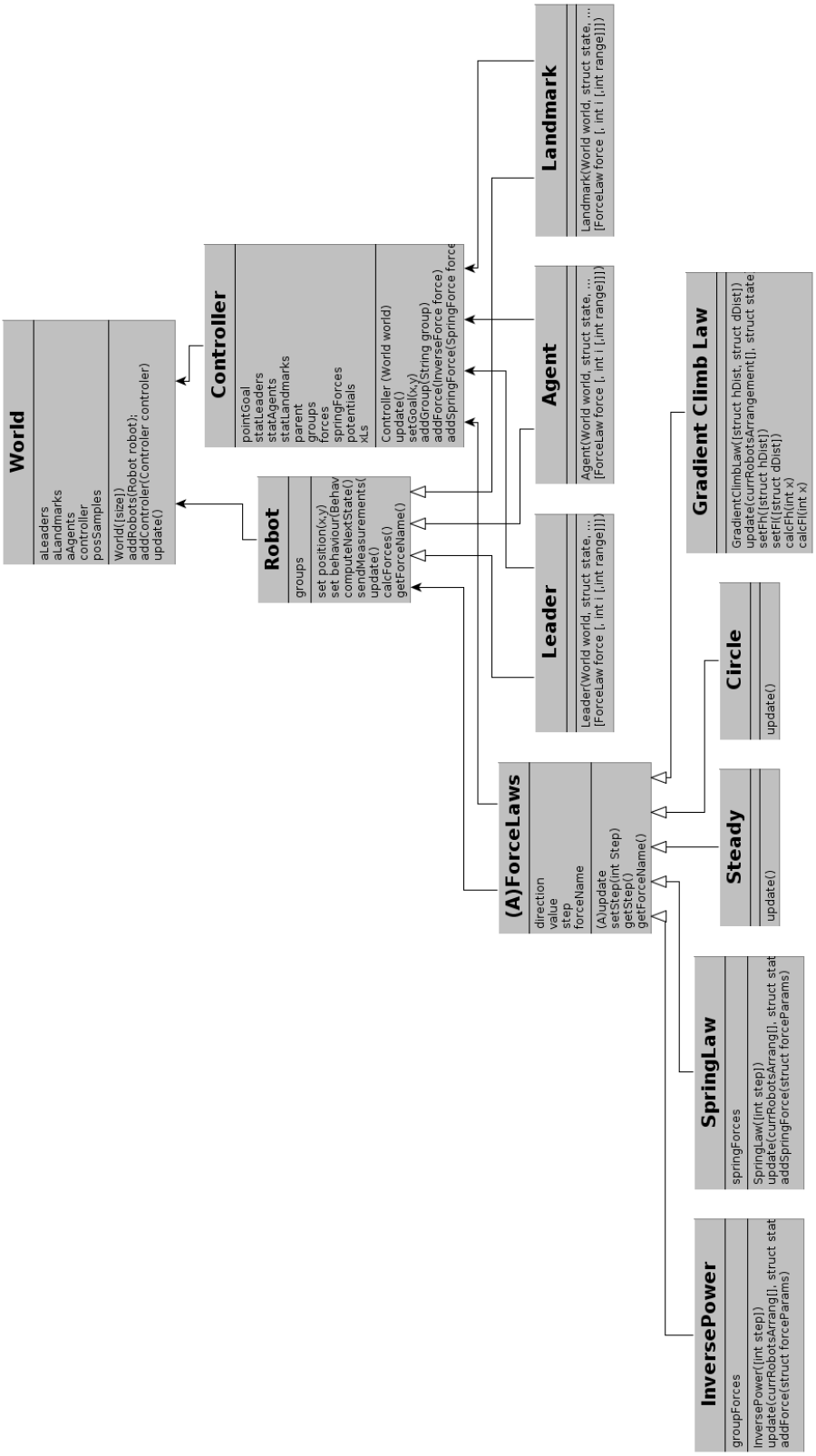
[2] Orgen P., Fiorelli E. and Leonard N.E., *Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment*, IEEE Transactions on automatic control, vol. 49, no. 8, 2004.

[3] Linial N., Lovasz. L. and Wigderson A., *Rubber bands, convex embeddings and graph connectivity*,Combinatorica 8(1) 91-102, 1988.

[4] R. Grygiel *Metody strojenia regulatora typu "PID"*, http://www.zsir.ia.polsl.pl/ dydaktyka/pa _gliwice/PA _mechatronika _NS/lab/DODATEK %20A.pdf

[5] MathWorks *Handle Classes*, http://www.mathworks.com/help/matlab/handle-classes.html

[6] MathWorks *Obiect-oriented programming*, http://www.mathworks.com/help/matlab/object-oriented-programming.html?s_tid=doc_12b



Rysunek 9. Szczegółowy schemat koncepcyjny implementacji platformy testowej. Strzałka wypełniona oznacza, że obiekt na który wskazuje zawiera w sobie obiekt z którego wychodzi strzałka. Strzałka niewypełniona oznacza, że obiekt na który wskazuje jest obiektem bazowym dla obiektu z którego wychodzi strzałka.