AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA AUTOMATYKI I ROBOTYKI

Praca dyplomowa magisterska

Kognitywny system eksploracji środowiska przez inteligentnego agenta z wykorzystaniem uczenia motywowanego.

Cognitive system of environment exploration by an intelligent agent using motivated learning

Autor: *Bartłomiej Styczeń*
Kierunek studiów: *Automatyka i robotyka*
Opiekun pracy: *dr hab. Adrian Horzyk, prof. AGH*

Kraków, 2021

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wprowadzenie

Od początku pierwszych urządzenie elektronicznych, a szczególnie zaprojektowaniu pierwszych komputerów, które mogły wykonywać pewne zbiory instrukcji, naukowcy zastanawiali się jak wykorzystać te moce obliczeniowe. Próbowano zdefiniować pojęcie inteligencji. I tak w roku 1955 termin ształuczna inteligencjaźostał wprowadzony jako dyscyplina naukowa. Natomiast w roku 1950 matematyk Alan Turing opracować test nazwany jego nazwiskiem, którego celem jest sprawdzenie czy maszyna, która posiada inteligencję równą lub przewyższającą człowieka.

Test został wprowadzony w artykule [1], w którym Turing zaproponował zastąpienie pytania: *Can machines think?* (ang. *Czy maszyny myślą?*) pewną formą testu. Polega on na prowadzeniu rozmowy przez 2 osoby i jedną maszynę. Zadaniem jednego człowieka i maszyny jest przekonanie sędziego, że są one człowiekiem. Jeżeli sędzie nie jest wstanie zdecydować, oznacza to, że maszyna zdaje test. Ostatecznie pytanie, które zadawał Turing na początku artykułu zostaje zmienione na: *Can machines do what we (as thinking entities) can do?* (ang. *Czy maszyny umieją robić to, co myślące istoty?*).

Temat inteligencji były coraz bardziej popularny, ale po kilku latach badań, nie osiągano rezultatów. Nastąpiła tzw. zima dla AI (ang. *AI winter*). W trakcie kilku dziesięcioleci nie prowadzono wielu badań w zakresie sztucznej inteligencji. Trwało to do początku lat dziewięćdziesiątych, kiedy to w roku 1998 francuski informatyk Yann LeCun wydał artykuł [2] opisujący konwolucyjne sieci neuronowe, które uczyły bezpośrednio z obrazów, a nie z wcześniej zaprojektowanych cech. Był to początek przełomu w procesie uczenia sztucznych sieci neuronowych, ponieważ pierwszy raz można było przeprowadzić w pełni automatyczne uczenie sieci neuronowej.

Kolejnym przełomem w dziedzinie sztucznej inteligencji była praca Alexa Krizhevskiego, Ilyi Sutskevera i Geoffreya E. Hintona [3]. Ich przełomowa implementacja architektury sieci konwolucyjnej na kartach graficznych umożliwiła stworzenie głębokiej (jak na rok 2012) sieci, która wygrała konkurs ImageNet [4]. Od tego roku tematyka sztucznych sieci neuronowych,

a w szczególności konwolucyjnych sieci neuronowych. Postęp technologiczny w architekturach kart graficznych umożliwił tworzenie coraz większych sieci, które dawały coraz lepsze rezultaty. Powrót do algorytmów uczenia ze wzmocnieniem (ang. *reinforcement learning*) sprawił, że powróciła chęć rozwijania sztucznej inteligencji, a nawet próby regulacji prawnej tych rozwiązań.

Szczególną uwagę należy zwrócić na uczenie ze wzmocnieniem, które przypomina po części silną sztuczną inteligencję (ang. *artificial general intelligence*). Agent mając niewiele informacji o środowisku jest w stanie nauczyć się skomplikowanych zadań bez wcześniejszej wiedzy. Popularnych przykładem jest AlphaGo, który został zaprojektowany przez naukowców z DeepMind do grania w grę Go [5].

Należy jednak pamiętać, że mimo świetnych wyników tych programów komputerowych, to daleko im od silnej sztucznej inteligencji. Są one przystosowane do rozwiązywania jednego problemu, w znanym środowisku. Jak środowisko zostanie zmienione, to wyniki nie są aż tak dobre. AlphaGo nie zacznie nagle autonomicznie sterować samochodem, ponieważ algorytm zna całkowicie inne środowisko. Można zauważyć brak przystosowania do zmieniającego się środowiska.

Jednym z podejść, aby rozwiązać problem dynamicznego środowiska jest umożliwienie agentowi samodzielne decydowanie jakie zadania musi on wykonać, aby rozwiązał główny problem. Uczenie motywowane (ang. *motivated learning*) odnosi się do tego problemu i umożliwia rozwiązywanie złożonych problemów w zmieniających się warunkach działania agenta.

Ta praca jest o uczeniu motywowanym przy eksploracji środowiska przez agenta, który w przypadku tej pracy dyplomowej będzie wykonywana na robocie mobilnym. Zastosowane zostanie środowisko symulacyjne umożliwiające sterowanie takim robotem, odczytywanie różnych parametrów środowiska otaczającego agenta, także tych, których sensory robota, np. kamera, czujniki odległości, nie umożliwiają do odczytania.

1.1. Cele pracy

Celem pracy było zbadanie skuteczności uczenia motywowanego przy eksploracji i poznawaniu wirtualnego środowiska przez inteligentnego agenta kognitywnego. Zakres pracy obejmował przygotowanie wirtualnego środowiska umożliwiającego testowanie zaimplementowanych rozwiązań. Następnie inteligenty agent miał za zadanie poznawanie środowiska i tworzenie nowych skojarzeń i wiedzy poprawiających jego umiejętności oraz stopniowo rozbudowujących hierarchię jego potrzeb i możliwości.

TODO

W ramach pracy należało sprawdzić i porównać skuteczność nowych rozwiązań z zakresu uczenia motywowanego z innymi podejściami, np. uczeniem ze wzmocnieniem.

TODO

Pierwszym etapem był przegląd literatury związanej z tematem uczenia motywowanego oraz uczenia ze wzmocnieniem, którego celem było zgłębienie wiedzy na ten temat w celu zrozumienia różnic i możliwych ograniczeń powyższych rozwiązań.

Bardzo ważnym etapem był wybór środowiska symulacyjnego, aby móc szybko testować zaimplementowane rozwiązania. Takie podejście umożliwia bezpieczne sprawdzenie algorytmów, w takim środowisku wystarczy jeden przycisk, aby przerwać całe zadanie bez zagrożenia dla ludzi w środowisku oraz samego robota. Ważne, aby wybrane środowisko umożliwiło dużą ingerencję w środowisko. Tworzenie dynamicznego świata jest podstawą, aby można było testować skuteczność algorytmów uczenia motywowanego.

Większość algorytmów zaimplementowano w językach C++/Python, tak aby wysoka logika mogła być szybko prototypowana w języku jakim jest Python, natomiast wszelkie obliczenia, które wymagają dużej wydajności zostały zaimplementowane w języku C++.

Istotnym aspektem tej pracy dyplomowej jest analiza porównawcza z innymi znanymi i ugruntowanymi algorytmami do eksploracji środowiska, np. SLAM (ang. *simultaneous localization and mapping*). Porównanie osiągnięć agenta wykorzystującego algorytmy uczenia motywowanego z podobnym agentem korzystającym z innych podejść umożliwi sprawdzenie czy umożliwienie tworzenia własnych wewnętrznych celów sprawia, że agent dostosowuje się do zmiennego środowiska.

1.2. Zawartość pracy

Ta część będzie napisana jak cała reszta pracy zostanie ukończona...

2. Uczenie motywowane

Człowiek od momentu, kiedy jest wstanie poznawać swoje środowisko, w którym się znajduje ma ochotę eksplorować na swoje możliwości. Małe dzieci, które nie umieją jeszcze chodzić albo raczkować, mogą tworzyć skojarzenia pomiędzy różnymi czynnikami. Jednym z przykładów może być nawet karmienie takiego dziecka małą butelką i poprzez umożliwianie mu trzymanie tej butelki samodzielnie, małe dziecko będzie w stanie po pewnym czasie utworzyć skojarzenie, że jeżeli będzie trzymało mocno butelkę i w odpowiedniej pozycji, to będzie mogło zjeść w razie głodu.

Eksploracja środowiska i swobodne poznawanie go, umożliwia tworzenie połączeń asocjacji pomiędzy pewnymi obserwacjami, akcjami jakie wykonaliśmy na nich oraz rezultatów jakie zostały otrzymanie w wyniku takich a nie innych akcji. Innym przykładem u małego człowieka jest to, że niepilnowane młode dziecko może dotknąć czegoś gorącego i się oparzyć, mimo, że rodzice zabronili tego. Dopiero poczucie fizycznego bólu sprawi, że dziecko będzie dobrze pamiętało, żeby więcej tego nie robić. Dlatego też człowiek często uczy się na własnych błędach najlepiej, kiedy sam czegoś doświadczy.

Kluczowe dla uczenia motywowanego (ang. *motivated learning*) jest pojęcie motywacji w ucieleśnionej inteligencji (ang. *motivation in embodied intelligence*). Inteligencja ucieleśniona jest obliczeniowym podejściem do projektowania i rozumienia inteligentnego zachowania u ucieleśnionych agentów poprzez wzięcie pod uwagę ścisłego powiązania między agentem, a jego otoczeniem. Należy brać pod uwagę jego ograniczenia tj. ograniczenia własnego ciała (ograniczenia mechaniczne i/lub elektryczne robota), systemu percepcyjnego (systemy wizyjne i ich sprzętowe ograniczenia) oraz motoryczne. Jedną z najbardziej wpływowych postaci w procesie projektowania ucieleśnionej inteligencji jako metodologii był Rodney Brooks. Zasugerował on tworzenie inteligentnych maszyn poprzez interakcję ze środowiskiem napędzane przez percepcję i akcję niż przez konkretne wyspecjalizowany algorytm.

W dzisiejszy czasach badania w zakresie sztucznej inteligencji są kojarzone z wyspecjalizowanym rozwiązywaniem problemów tj. reprezentacja wiedzy, zrozumienie języka naturalnego czy oglądanej sceny czy odpowiadanie na pytania. Wszystkie te zagadnienia są bardzo ciekawe

i sprawiają, że wiele problemów zostaje rozwiązywane przez komputery. Jednak nie ma w nich systemu, który spaja wszystkie te zagadnienia w jeden system, który mógłby być określaną śilną inteligencją".

2.1. Definicje związane z projektowaniem ucieleśnionej inteligencji

W celu projektowania maszyn ucieleśnionej inteligencji zostały zdefiniowane pewne zasady i założenia. Pierwszym z nich było to, że agent rozwija się w zmieniającym środowisku, które może manipulować i postrzegać korzystając z dostępnych sensorów. Nie istnieje potrzeba tworzenia modelu środowiska, ponieważ ono się może zmieniać.

Dodatkowe reguły projektowania zostały zawarte w [6]:

1. Reguła taniego projektu i nadmiarowości – projekt maszyny powinien być prosty i jego projekt sprawiał, że funkcjonalności podzespołów nachodzą na siebie pozwalając na działanie maszyny w razie awarii jakiegoś podzespołu.

2. Reguła równoległych, luźno związanych procesów – ta reguła wymusza na maszynie to, że inteligencja pojawia się przez interakcję niskopoziomowych procesów np. sterowania ramionami ze środowiskiem.

3. Reguła wartości – reguła ta może wymuszać, aby maszyna robiła, to co jest dobre dla niej. Umożliwia ona sugerowanie maszynie, co powinna zrobić w danej sytuacji (agenty uczenia ze wzmocnieniem uczą się korzystając m.in. z tej reguły)

2.2. Inteligencja

Do dzisiaj nie ma dokładnej definicji inteligencji. Jest wiele interpretacji, a jednocześnie należy mieć na uwadze, aby nie mylić inteligencji ze złożonym zachowaniem. To bardzo ważne, ponieważ możemy wziąć pewien algorytm, np. szukania najkrótszej ścieżki w grafie, która ogółem jest złożonym problemem, który składa się ze skończonej liczby kroków. Zdecydowanie nie można stwierdzić, że ten algorytm jest inteligenty. To tylko pewien zbiór prostych czynności, które rozwiązują konkretny problem i użyte do czegoś innego (można tak nazwać zmieniające się środowisko), na pewno nie da dobrych rezultatów.

Opisy inteligencji skupiają się na opisie właściwości umysły, niż samym umyśle. W swojej pracy [7] John Stewart zdefiniował systemy kognitywne jako:

Definicja: System jest kognitywny wtedy i tylko wtedy, gdy wejścia z sensorów powodują akcje w konkretny sposób, tak aby spełnić wymagania do przeżycia w środowisku.

Definicja: Uczeleśniona inteligencja (ang. *embodied intelligence* EI) jest zdefiniowana jako mechanizm, który uczy się jak przetrwać we wrogim środowisku (wrogie można także określić jako zmieniające się).

Ta druga definicja odnosi się do wszystkich form ucieleśnionej inteligencji: biologicznej, mechanicznej czy wirtualnych agentów. Wynika z tego, że agent EI wchodzi w interakcje ze środowiskiem i postrzega wprowadzone zmiany poprzez sensory. Agent musi się nauczyć jak przetrwać w środowisku. Natomiast wrogość środowiska może być określana na różne sposoby. Może to być jego ciągła zmienność, czy mała dostępność surowców potrzebnych agentowi przetrwania. Ważne jest to, że ta wrogość jest stała. Przykładowo, poziom naładowania baterii jest stałych zagrożeniem dla agenta. Jeżeli poziom naładowania będzie się zmniejszał, to dyskomfort wywoływany przez sensor odpowiedzialny za pomiar baterii będzie rósł.

Kolejnym ważnym elementem definicji ucieleśnionej inteligencji jest aspekt uczenia. Jeżeli agent wie jak przetrwać we wrogim środowisku, ale nie wie jak się nauczyć nowych umiejętności, nie jest inteligentny. Przykładowo, algorytm, która zajmuje się sterowaniem jakiegoś procesu technologicznego nie jest inteligentny, ponieważ nie będzie w stanie nauczyć się sterować samochodem. W zamierzeniu jego zadaniem było regulowanie wartości zadanej konkretnego procesu technologicznego.

Należy zwrócić uwagę, że EI odróżnia wiedzę od inteligencji. Wiedza może być w pewien sposób zintegrowana z agentem w trakcie jego projektowania, np. jako zbiór pewnych reguł zachowania. Natomiast inteligencja sprawia, że ta wiedza może być użyta do tworzenia nowych umiejętności i zarządzania nimi w sposób zorganizowany i koherentny.

2.3. Projektowanie ucieleśnionej inteligencji

Uczenie jest aktywnym procesem. Informacje o otoczeniu EI zdobywa poprzez koordynację pary sensor – motor. Uczenie się, które akcje są pożądane, a które nie, sprawiają, że agent uczący jest lepiej przystosowany do wrogiego środowiska. Można odróżnić co najmniej dwa sposoby na adaptowanie do zmiennego środowiska [8]:

1. ewolucyjne – naturalna selekcja agentów (gatunków zwierząt), które są najlepiej przystosowane lub rozwój nowych umiejętności np. pocenie się, aby utrzymać odpowiednią temperaturę,

2. kognitywne – poprzez uczenie, używając pamięci i asocjacji, stosując rozpoznawanie schematów, budowanie reprezentacji i implementowanie celów.

Schematy na wielu poziomach abstrakcji są uczone i zapamiętywane przez agenta. Pewne abstrakcyjne reprezentacje są budowane aby reprezentować akcje i umiejętności. Cała struktura wzorów i relacji pomiędzy nimi jest reprezentowana w pewnej postaci w pamięci, która jest asocjacyjna i epizodyczna. Ten system jest rozproszony, nadmiarowy i równoległy a dodatkowo krótko i długoterminowy. Cały ten system łączy się ze sobą i wchodzi w interakcję w czasie rzeczywistym.

Krytycznym aspektem rozwoju umysłu człowieka jest samoorganizacja. Dzięki niej neurony mogą szybko tworzyć nowe reprezentacje zapamiętanych schematów, uczyć się jak wchodzić w interakcję z otoczeniem oraz budować oczekiwania związane z przyszłymi wydarzeniami, czyli umiejętność prognozowania przyszłych wydarzeń na podstawie aktualnego stanu oraz tego, co stało się w przeszłości.

Agent uczenia motywowanego dzieli wspólne właściwości ze specjalnym typem agenta racjonalnego (ang. *rational software agent*) znanego jako agenta przekonań – pragnień – intecji (ang. *belief – desire – intention*). To pewien z modeli tworzenia oprogramowania, który umożliwia rozdzielenie funkcjonalności od siebie. Ten model skupia się na planowaniu i sposobie wykonania pewnych zadań. Cechy wspólne z agentem uczenia motywowanego to tworzenie reprezentacji poznanego środowiska, pragnienia to motywacje dla agenta motywowanego. Różnicą jest sposób implementacji reprezentacji wiedzy. Dla agenta uczenia motywowanego nie ma konkretnego sposobu implementacji, ponieważ widza ta jest reprezentowana przez struktury sieci neuronowych a tym samym wszelkie plany, co do wykonywanych akcji muszą być pochodzą z aktywacji neuronów w sieci neuronowej.

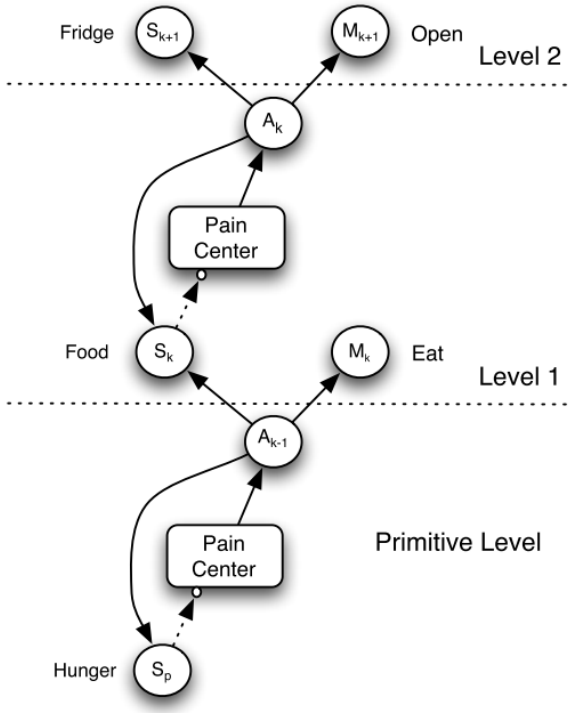
2.4. Ból jako motywacja

Zadaniem każdej maszyny jest wykonywanie pewnych z góry określonych zadań. Pytaniem jest, co może być motywujące, aby dana maszyna/program wykonywały go w sposób perfekcyjny. Dla pewnych grup algorytmów można zdefiniować z góry określoną funkcję kosztu, którą w sposób iteracyjny usprawnia działanie danego algorytmu. Jednakże takie podejście daje bardzo mało możliwości do generalizowania zdobytych umiejętności na nowe potrzeby, np. zmiennego środowiska.

Aby odpowiedzieć na pytanie, co może motywować maszyny do ulepszania swoich możliwości w sposób bardziej zgeneralizowany, można zastanowić się, co motywuje ludzi do rozwijania swoich umiejętności. Próbą odpowiedzi na to pytanie podjął się psycholog Csíkszentmihályi w książce opisującej teorię przepływu [9]. W tej teorii opisano, że ludzie otrzymują wewnętrzne nagrody za wszelkie aktywności, które są trochę ponad ich aktualny poziom rozwoju.

Starzyk w pracy [8] sugeruje, że to wrogość środowiska, które jest ujęte w definicji agenta ucieleśnionej inteligencji (EI) jest najbardziej efektywnym czynnikiem motywującym do uczenia. Ma to odniesienie do ludzkiego sposobu uczenia. Podstawowym bólem może być ciekawość. To ona sprawia, że chcemy poznawać nowe rzeczy. Dla małego dziecka taka ciekawość może wywołać ból fizyczny, kiedy z ciekawości oparzy się dotykając czegoś gorącego. Ale ten ból sprawi, że w przyszłości będzie wiedziało jak postępować, aby nie zwiększać tego bólu.

Według Starzyka bóle prymitywne są bezpośrednio związane z bodźcami odbieranymi przez sensory maszyny. Natomiast definiuje on bóle abstrakcyjne (ang. *abstract pains*). Pojawiają się one, kiedy agent nie jest w stanie wykonać akcji, która zmniejszy wartość bólu prymitywnego. W ten sposób mogą się tworzyć struktury grafowe opisujące relacje pomiędzy różnymi bólami. Co ważne, bóle abstrakcyjne nie są stymulowane przez sensor wartości fizycznej, np. niski poziom naładowania baterii dla robota. 2.1.



Rys. 2.1. Tworzenie sygnałów abstrakcyjnych bóli. Źródło: [10].

Bóle tworzą hierarchię wszerz i wgłąb, tworząc struktury grafowe. W takich strukturach teoretycznie mogą pojawiać się zamknięte ścieżki, np. brak jedzenia może sprawić, żeby kupić więcej jedzenia, a do tego potrzebne są pieniądze, które można zdobyć sprzedając jedzenie. Należy wziąć pod uwagę takie sytuacje w procesie tworzenia całej architektury i sposobu uczenia maszyny.

2.5. Tworzenie celów w uczeniu motywowanym

Jednym z podstawowych źródeł tworzenia nowych celów jest wcześniej wspomniany ból. Najbardziej prymitywnym bólem u ludzi jest uczucie dyskomfortu, gdy czują głód. W maszynach można to odnieść do poziomu naładowania baterii w robocie i coraz niższy poziom sprawia, że ból związany z brakiem energii zwiększa się. Agent uczenia motywowanego musi się nauczyć jak ten ból zmniejszać, tak jak człowiek uczy się zdobywać jedzenie oraz jeść. Mimo, że wszelkie abstrakcyjne bóle, które wykształcają się w dalszym rozwoju, to te prymitywne umożliwiają realizację kolejnych zadań czy celów.

Dokładny opis algorytmu tworzenia celów został opisany w [11]

Algorytm tworzenia celów (ang. *goal creation system*)

1. Wybierz dominujący ból stosując regułę zwycięzca bierze wszystko (ang. *winner takes all*) spomiędzy konkurujących ośrodków bólu.
   - Jeżeli żaden z bóli nie przekracza wcześniej zdefiniowanego progu, czekaj aż któryś z nich przekroczy ten próg.
2. Jako aktualny cel wybierz zmniejszenie dominującego bólu.
   - aktualny cel motywuje agenta do działania.
3. Wybierz wcześniej nauczoną akcje, która z najwyższym prawdopodobieństwem spełni aktualny cel.
   - Jeżeli nie ma żadnego, idź do punktu 6.
4. Sprawdź czy wybrana czynność może być wykonana w aktualnym środowisku. Jeśli nie, idź do punktu 3.
5. Wykonaj akcję.
   - Jeśli ta akcja *obniżyła* wartość dominującego bólu:
     (a) Zwiększ wartości wag zależności pomiędzy aktualnym bólem a akcją jaka została wykonana i zwiększ wartość wag odpowiadających abstrakcyjnemu bólowi powiązanego z tą akcją.
     (b) Idź do punktu 1.
   - Jeśli ta akcja *nie obniżyła* wartości dominującego bólu.

(a) Zmniejsz wartości wag zależności pomiędzy aktualnym bólem a akcją jaka została wykonana i zmniejsz wartość wag odpowiadających abstrakcyjnemu bólowi powiązanego z tą akcją.

(b) Idź do punktu 3.

6. Wykonaj eksplorację przestrzeni akcji mającą na celu spełnienie celu.

- Jeśli nowa akcja *zmniejszyła* wartość dominującego bólu.

  (a) Zwiększ wartości wag zależności pomiędzy aktualnym bólem a akcją jaka została wykonana i stwórz nowy abstrakcyjny ból związany z niemożliwością wykonania tej akcji.

  (b) Idź do punktu 1.

- Jeśli nowa akcja *nie zmniejszyła* wartości dominującego bólu, idź do punktu 6.

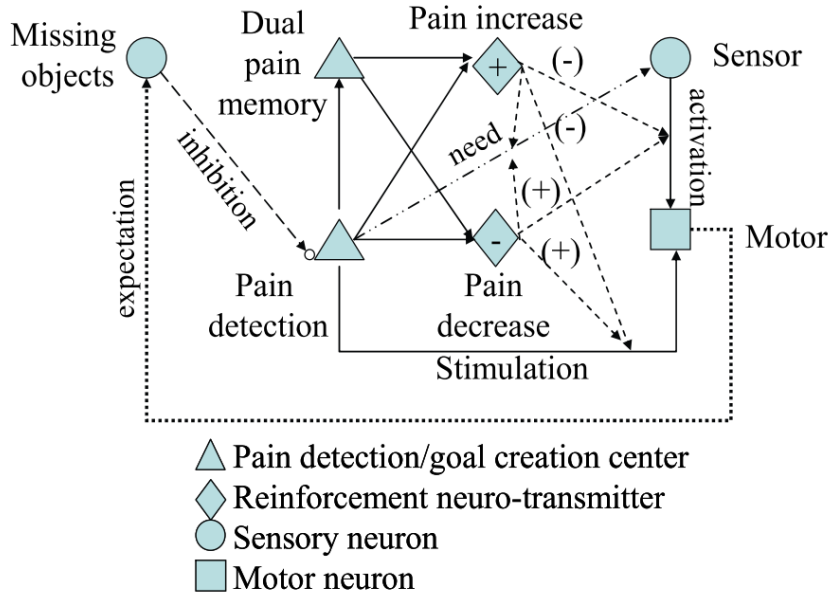
Uczenie motywowane może być zastosowane wspólnie z uczeniem opartym na ciekawości (ang. *curiosity based learning*) – ciekawość poinformuje agenta o nowych odkryciach, podczas gdy uczenie motywowane skupi się na poszukiwaniu konkretnych celów, które nie są podane przez twórcę (tak jak w uczeniu ze wzmocnieniem).

2.6. Podstawowy element systemu tworzenia celów

Mechanizm budowania celów składa się z wielu, prostych elementów nazywanych jednostkami tworzenia celów (ang. *goal creation unit*). Składa się on głównie z trzech typów neuronów, które wchodzą ze sobą w interakcję. Są to:

- neurony odpowiadające za ból (ang. *pain center neurons*),
- neurony wzmacniające lub zmniejszające odczucie bólu (ang. *reinforcement neurotransmitter neurons*),
- neurony odpowiadające za motorykę (ang. *neurons in sensory and motor pathway*).

Schemat reprezentujący połączenia zawarto na rysunku 2.2.



Rys. 2.2. Schemat podstawowego elementu systemu tworzenia celów. Źródło: [8].

Neurony odpowiadające za odczuwanie bólu (te po lewej stronie na rysunku 2.2) są stymulowane przez sygnał reprezentujący ból oznaczany przez I_P. Reprezentuje on negatywny stymulant, np. ból, dyskomfort czy nieprzyjemność. Ponieważ ból istnieje z powodu braku czegoś w środowisku (oznaczone na rysunku jako "missing objects"), to neurony odpowiedzialne za detekcję bólu aktywują się w momencie wyciszenia neuronu odpowiedzialnego za czytanie wartości z otoczenia. Dodana została także pamięć wartości bólu z poprzedniego wydarzenia, oznaczone jako I_{Pd}. Neurony wykorzystują tą wartość do wzmacniania uczucia odczuwania bólu, jeżeli zwiększył się on od poprzedniego wydarzenia. Wartość tego wzmocnienia zapisano w równaniu 2.1.

$$r = I_P - I_{Pd} \tag{2.1}$$

Ostatnią grupą (po prawej stronie na rysunku 2.2) są neurony odpowiedzialne za obliczanie wartości akcji jakie ma wykonać robot. Na początku centrum detekcji bólu bezpośrednio wpływa na neurony odpowiedzialne za wykonywanie akcji, ponieważ wszelkie wagi neuronów W_{MP} są losowo zainicjalizowane. W trakcie poznawania środowiska, maszyna dostraja wartości tych wag. Dodatkowo, aby pewna akcja była wykonana muszą być dostępne pewne obiekty z otoczenia (na rysunku 2.2 oznaczone jako Ŝensor"po prawej stronie). Na początku neuron będzie miał powiązania z wieloma różnymi obiektami poprzez losowa zainicjalizowane wagi W_{MS}.

Po wykonanej akcji, która zmniejszyła albo zwiększyła wartość odczuwanego bólu (wykrytego przez drugą grupę neuronów odpowiedzialną za detekcję bólu), generowany jest sygnał r (obliczony zgodnie z równaniem 2.1). Wykorzystywany jest to wzmocnienia lub osłabienia wag W_{MP} i W_{MS} zgodnie z równaniem 2.2

$$\begin{aligned} W_{MP} &= W_{MP} + r \cdot \beta^n \\ W_{MS} &= W_{MS} + r \cdot \beta^n \end{aligned} \tag{2.2}$$

gdzie:

- β – oznacza współczynnik uczenia, mniejszy od wartości 1,
- n – oznacza jak wiele razy dane połączenie było zmieniane.

Dodatkowo neuron odpowiedzialny za reprezentację obiektu, który był użyty do zmniejszenia bólu zostanie dodany z wagą W_{SP} w celu połączenia centrum detekcji bólu z neuronem sensorycznym. Ten neuron będzie uczony korzystając uczenia metodą Hebba.

Z kolei z drugiej strony, obiekt, którego zabrakło i stworzył on ból abstrakcyjny, staje się dostępny. Zostaje stworzony neuron reprezentujący połączenie neuronu odpowiedzialnego za motorykę z neuronem z brakującym obiektem z wagą W_{SM}.

Oba wymienione wcześniej połączone będą uczone metodą wzmocnienia. Oznacza to, że im mocniejsza zmiana poziomu bólu, tym mocniejsza korekta wartości tych wag.

2.7. Budowanie hierarchii celów

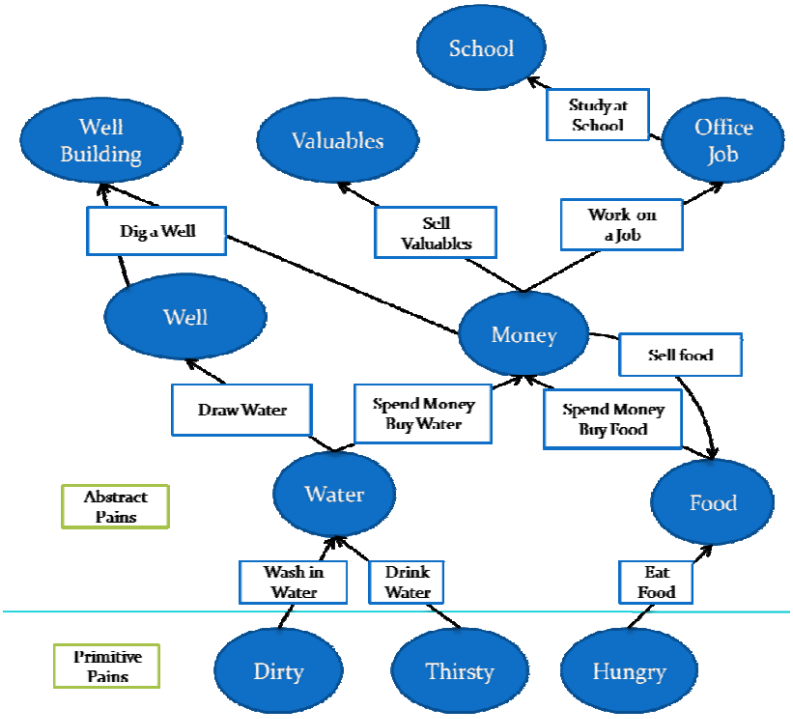
Prymitywny ból to sygnał otrzymywany przez sensory prymitywnych bóli. Stymulują one centrum detekcji prymitywnego bólu. Aby rozwiązać ten problem, maszyna jest zmuszona do eksploracji dostępnych akcji albo wykonywania akcji, która zmniejsza ten ból. Na początku symulacji, taka eksploracja opiera się na losowych sprawdzaniu przestrzeni dostępnych akcji lub wykorzystaniu wprowadzonych akcji przez twórcę maszyny. Można to określić jako genetycznie ustawione pewne akcje, które nie muszą być poszukiwane w procesie eksploracji, ponieważ agent od początku ma o nich wiedzę. U człowieka przykładem może być bicie serce czy niekontrolowane przez nasz bezpośrednio oddychanie, które umożliwia podtrzymania funkcji życiowych.

W celu opisania schematu tworzenia całej hierarchii posłużono się prostym przykładem prymitywnego bólu opisującego niski poziom cukru we krwi. W robocie mobilnym można to określić jako niski poziom naładowania baterii. Aby rozwiązać problem niskiego poziomu cukru (naładowania baterii), po pewnym czasie eksploracji dostępnych akcji, akcja "jedzenie",

wraz percepcją "jedzenia"jest nagrodzona, ponieważ zmniejsza ból związany z niskim poziomem cukru. Następuje asocjacja, żeby obniżyć poziom bólu związany z niskim poziomem cukru należy akcję "jedzenia"połączyć ze znalezieniem i obserwacją "jedzenia". Dodatkowo akcja "jedzenie"będzie powodowała oczekiwanie na zapewnienie odpowiedniego poziomu cukru (poziomu naładowania baterii w robocie).

Brak "jedzenia"(brak możliwości używania baterii albo brak wiedzy, gdzie się ona znajduje) sprawia, że w momencie wysokiego poziomu bólu spowodowanego niskim poziomem cukru, spowoduje pojawienie się stresu z powodu braku "jedzenia", który można nazwać abstrakcyjnym centrum bólu. Ból abstrakcyjny nie jest pobudzany przez odczucia sensoryczne. Symbolizuje on pewien dyskomfort spowodowany brakiem możliwości obniżenia poziomu bólu związanego z prymitywnym bólem.

Jeżeli "jedzenieńie jest dostępne, agent nie jest w stanie zredukować prymitywnego bólu. Zaczyna szukać rozwiązania ąbstrakcyjnego bólu". Uczenie odbywa się w podobny sposób jak dla prymitywnego bólu. Agent poszukuje jakie pary akcji i obserwacji sprawiają, że ból się zmniejsza i neurony wzmacniające to odczucie aktualizują swoje wagi. Cały schemat kolejnych abstrakcyjnych bóli i ich relacji do prymitywnych bóli pokazano na rysunku 2.3.



Rys. 2.3. Tworzenie abstrakcyjnych bóli. Źródło: [11].

3. Uczenie ze wzmocnieniem

Przez wiele lat, wszelkie problemy optymalizacji były rozwiązywane przez określenie funkcji kosztu, która może być wielokryterialna, tzn. wiele zmiennych są optymalizowane. Często takie problemy są bardzo skomplikowane do rozwiązania analitycznego a czasem nawet niemożliwe. Często problemem jest wiele wymiarów, które składają się na problem optymalizacji. Dodatkowo zmienne stanu wpływają na siebie, a tym samym nie ma możliwości na optymalizację każdego parametru osobno.

Sama dziedzina uczenia ze wzmocnieniem jest uznawana, obok uczenia nienadzorowanego i uczenia nadzorowanego, za trzecią gałąź sztucznej inteligencji. Początki uczenia ze wzmocnieniem się dzielą na trzy oddzielne wątki, które przez wiele lat ewoluowały oddzielnie, aby pod koniec lat 80.-tych XX wieku połączyć się w uczenie ze wzmocnieniem jakie jest znane dzisiaj.

Pierwszy wątek opiera się na metodzie "prób i błędów"i wywodzi się z psychologii i badaniami nad zwierzętami. Drugi wątek traktował o sterowaniu optymalnym i rozwiązań stosujących funkcje kosztów i dynamicznego programowanie, które jest bardzo ważnym elementem uczenia ze wzmocnieniem, ale także rozwiązywanie problemów optymalizacji.

Mniej znaną gałęzią rozwijaną były badania związane nad uczeniem *temporal – difference*, które polega na porównywaniu wartości funkcji kosztów dla dwóch różnych chwil czasowych.

Z biegiem czasu pojawiał się coraz mocniejszy sprzęt komputerowy, który umożliwiał uruchamianie bardziej zaawansowanych i rozbudowanych algorytmów. Dużą popularność uzyskał algorytm *AlphaZero* opracowany przez firmę DeepMind w roku 2017, który przez mechanizm grania ze sobą (ang. *self play*) i przeszukiwanie bardzo dużej przestrzeni stanów stosując heurystykę *Monte – Carlo Tree Search* był w stanie opanować gry t.j. szachy, shogi czy go i pokonać światowej sławy graczy. Szczególnie imponujące było bardzo szybkie opanowanie gry go, którą przez wiele lat uważano za niemożliwą do opanowania, ze względu na ogromną ilość możliwych pozycji, która została oszacowana na $2.1 \cdot 10^{170}$.

Kolejnym bardzo dużym postępem w aktualnych badaniach związanych z uczeniem ze wzmocnieniem była kolejna iteracja AlphaZero od firmy DeepMind, *MuZero* [12]. Algorytm

także stosował metody przeszukiwania Monte – Carlo Tree Search do stworzenia ogólnego modelu, który był w stanie grać na 57 grach Atari. Nie znając wcześniej reguł powyższych gier, MuZero było w stanie osiągnąć wyniki przewyższające te osiągane przez człowieka.

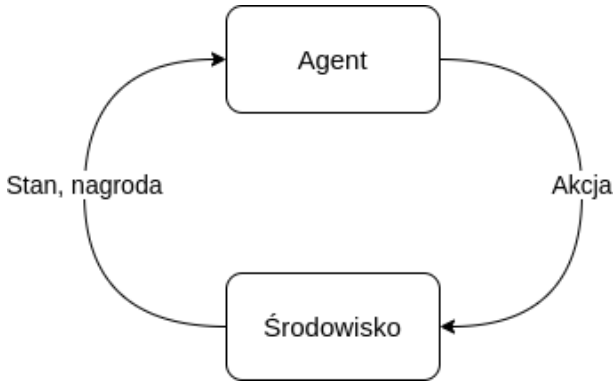
Do nauczenia powyższych algorytmów uczenia ze wzmocnieniem potrzebna było środowisko symulacyjne, wysokiej jakości sprzęt. Przeprowadzono wiele milionów iteracji, w których algorytm poprzez grę z samym sobą (termin w literaturze anglojęzycznej opisujący to podejście to *self play*). Obiecujące jest to, że MuZero był w stanie opanować wiele gier różnych od siebie i o innych regułach w stopniu, który człowiekowi zająłby wiele lat.

3.1. Podstawowe określenia i terminologia

Zanim zostaną przedstawione najbardziej znane algorytmy uczenia ze wzmocnieniem, należy wprowadzić kilka podstawowych określeń i oznaczeń stosowanych w opinie problemów związanych z algorytmami uczenia ze wzmocnieniem.

3.1.1. Agent i środowisko

Głównymi elementami każdego algorytmu uczenia ze wzmocnieniem jest **agent** i **środowisko**, na który ten agent może wpływać. Bardzo często każdą symulację dzieli się na krótkie chwile czasowe, w których agent wykonuje jakąś akcję. Odnosząc to do naszego agenta, którym jest robot mobilny, taką akcją może być żądana prędkość liniowa i prędkość kątowa robota nieholomicznego. Zakładając, że środowisko jest choć częściowo obserwowalne, agent otrzymuje aktualny **stan środowiska** oraz **nagrodę**, która określa czy dana akcja zmieniała stan symulacji. Cały ten mechanizm został przedstawiony na rysunku 3.1.



Rys. 3.1. Schemat interakcji agenta ze środowiskiem. Źródło: opracowanie własne

Oznaczenia jakie się przyjmuje w literaturze:

- a_t – akcja w chwili czasu t,
- s_t – stan (a dokładniej obserwacja) po wykonaniu akcji,
- r_t – nagroda, jaką agent otrzymał dla wykonanej akcji.

Głównym celem każdego algorytmu uczenia ze wzmocnieniem jest maksymalizacja skumulowanej nagrody. W trakcie nauki, agent może otrzymywać czasem wysokie nagrody w danej epoce, a czasem nawet utracić część nagrody, ale jeżeli skumulowana wartość rośnie, oznacza to, że agent zwiększa wiedzę na temat środowiska w jakim operuje.

3.1.2. Przestrzenie akcji

Różne algorytmy wymagają innych typów akcji. Wiele gier wirtualnych jak Atari czy Go ma wielowymiarowe przestrzenie akcji, które są **dyskretne**. Oznacza to, że liczba możliwych akcji, które agent może wykonać jest skończona. Przykładowo dla robota mobilnego takimi akcjami są: włącz silnik, wyłącz silnik, obróć się w lewo i obróć się w prawo. Niestety takie akcje dla wielu problemów nie mają zastosowania w praktyce. Dlatego kolejnym typem jest **przestrzeń ciągłych akcji**. Często te przestrzenie opisuje się wektorem liczb rzeczywistych. Przykładowo dla robota mobilnego taką akcją może być liczba rzeczywista z przedziału od 0 do 1, która opisuje wartość sterowania silnikiem, która przenosi się na wartość napięcia na silniku prądu stałego dla wartości od 0 wolt to 12 wolt.

3.1.3. Polityki

Każdy agent algorytmów uczenia ze wzmocnieniem podejmuje decyzję aby zmaksymalizować skumulowaną nagrodę. Jednak potrzebne są pewne reguły, jakimi taki agent będzie się kierował w trakcie eksploracji środowiska. Tak określana jest **polityka**, z którą agent podejmuje decyzje. W nomenklaturze oznacza się tą politykę grecką literą μ. Polityki dzieli się na deterministyczne 3.1:

$$a_t = \mu(s_t) \tag{3.1}$$

Oznacza to, że akcja jaką wykona agent jest determinowana przez politykę tylko na podstawie aktualnego stanu środowiska, z którym agent wchodzi w interakcję.

Polityki mogą być także stochastyczne, czyli takie dla którym nie ma na sztywno zdefiniowanych akcji, dla konkretnych stanów środowiska. Akcja jest wybierana z pewnego rozkładu, na którym polityka (czyli agent podejmujący decyzję), zgodnie z równaniem 3.2:

$$a_t \sim \pi(\cdot|s_t) \tag{3.2}$$

Bardzo ważną dziedziną, która od momentu rozwoju wydajności sprzętu, jest głębokie uczenie ze wzmocnieniem *deep reinforcement learning*. Jak nazwa sugeruje, w głębokim uczeniu ze wzmocnieniem korzysta się z sieci neuronowych, a wartości jej parametrów są optymalizowane korzystając z algorytmu optymalizacji. Dla sieci neuronowych najczęściej jest to algorytm gradientu najszybszego spadku. Dla rozwiązywania problemów głębokiego uczenia ze wzmocnieniem stosuje się sparametryzowane polityki, w których parametry oznacza się literą θ. W literaturze często określa się równanie polityki jako 3.3:

$$\begin{aligned} a_t &= \mu_\theta(s_t) \\ a_t &\sim \pi_\theta(\cdot|s_t) \end{aligned} \tag{3.3}$$

3.1.4. Trajektorie

Kolejnym elementem uczenia ze wzmocnieniem są tzw. *trajektorie*. Przyjmuje się, że jest to sekwencja stanów (używane zamiennie z obserwacjami) i akcji oznaczaną:

$$\tau = (s_0, a_0, s_1, a_1, \dots) \tag{3.4}$$

Przejścia między stanami, określane jako tranzycje, w chwili czasu t oraz chwilą $t+1$, są tylko zależne od reguł danego środowiska oraz akcji, która została wykonana w chwili czasu t. Tak jak i z politykami mogą być one deterministyczne:

$$s_{t+1} = f(s_t, a_t) \tag{3.5}$$

lub stochastyczne:

$$s_{t+1} = P(\cdot|s_t, a_t) \tag{3.6}$$

Akcja a_t jest wybierana zgodnie z polityką zastosowaną w danym algorytmie uczenia ze wzmocnieniem. Zwykle trajektoriami nazywa się **epizody**.

4. Uczenie motywowane i ze wzmocnieniem – porównanie

Wyniki eksperymentów zawarte w [11] pokazują, że agent korzystający z uczenia motywowanego radzi sobie lepiej w skomplikowanym środowisku niż agent z uczeniem ze wzmocnieniem. Szczególnie istotne jest to, że środowisko staje się coraz bardziej wrogie dla agenta, np. poprzez coraz mniejsze zasoby potrzebne dla agenta. Jednakże, aby agent mógł rozwijać się w sposób optymalny, skomplikowanie i wrogość środowiska musi się stopniowo zwiększać. Nie może to być nagła zmiana, ponieważ wtedy agent może mieć problemy z dostosowaniem się dostatecznie szybko. Można to odnieść do środowiska małego dziecka, które się rozwija poznając świat. Na początku jego rozwoju, dziecko ma wiele udogodnień zapewnionych przez rodziców.

Pomimo istotności zaspokajania bóli prymitywnych, o których więcej pisano w 2, to bardziej skomplikowane maszyny korzystające z uczenia motywowanego rozwijają skomplikowaną strukturę bóli abstrakcyjnych. Cele te mogą różnić się od tych wyznaczonych przez projektanta. Maszyna może zdecydować się na dążenie do celu wyższego poziomu, nawet jeśli może mieć środki i wiedzieć, jak realizować cele niższego poziomu. Ponieważ decyzja, do jakiego celu dążyć, opiera się na sile różnych sygnałów bólowych, w każdej chwili może dominować abstrakcyjny ból i agent wykona działania mające na celu uśmierzenie tego bólu. Dlatego agent musi być odpowiednio prowadzony (być może ze specjalnymi zachętami, które uznają za satysfakcjonujące), aby wykonywać użyteczne działania, bez zmuszania ich do tego. Zdecydowanie nie jest to to, co zrobi agent z uczeniem ze wzmocnieniem, ponieważ zawsze dąży do określonych, z góry określonych celów. Maszyna uczenia ze wzmocnieniem może nauczyć się wykonywania celów podrzędnych tylko wtedy, gdy służą one do osiągnięcia z góry określonego celu, jak omówiono w hierarchicznych algorytmach wzmocnienia [13].

Uczenie ze wzmocnieniem może wykorzystywać sztuczną ciekawość do odkrywania i uczenia się hierarchii celów podrzędnych. Jednak wszystkie kroki, które wykona, odpowiadają głównym celom wyznaczonym przez projektanta i są tylko krokami na drodze do ich osiągnięcia. W najlepszym przypadku można je uznać za cele cząstkowe wyznaczonego celu. Żaden

z tych celów podrzędnych nie może sam w sobie być oddzielnym celem i może być realizowany tylko wtedy, gdy został wywołany potrzebą osiągnięcia celu – jako etapy pośrednie do osiągnięcia celu.

W przeciwieństwie do tego agent z uczeniem motywowanym może szukać rozwiązania abstrakcyjnego celu, nawet jeśli może osiągnąć cel wyznaczony przez projektanta. Jednak robi to, aby nauczyć się złożonych relacji w środowisku, które są istotne dla jego zdolności do realizacji określonych przez projektanta celów. Jest więc gotowa wykorzystać tę wiedzę w razie potrzeby, gdy warunki środowiskowe ulegną zmianie. Prawdą jest, że uczenie się oparte na ciekawości może dostarczyć maszynie podobnej wiedzy; Jednak prawdopodobieństwo przypadkowego odkrycia wiedzy, która jest istotna dla celów maszyny, jest bardzo niskie.

Tabela 4.1. Porównanie uczenia ze wzmocnieniem do uczenia motywowanego. Źródło: [11].

Uczenie ze wzmocnieniem	**Uczenie motywowane**
Jedna funkcja kosztu (określona zewnętrznie)	Wiele funkcji kosztu (tworzone przez agenta)
Mierzalne nagrody – może być zoptymalizowane	Wewnętrzne nagrody (niemierzalne) – nie może być zoptymalizowane
Możliwy do przewidzenia	Niemożliwy do przewidzenia
Zadania zdefiniowane przez badacza	Agent sam stwarza sobie cele
Maksymalizacja nagrody – potencjalnie niestabilne przy optymalizacji	Algorytm minimax – stabilne
Brak wewnętrznych motywacji i tworzenia celów	Wewnętrzne motywacje i tworzenie celów
Zawsze aktywne	Wykonuje akcje, kiedy jest taka potrzeba albo agent uzna to za konieczne
Wysiłek uczenia wzrasta wraz ze zwiększaniem skomplikowania środowiska	Agent uczy się lepiej w skomplikowanych środowiskach

W tabeli 4.1 przedstawiono główne różnice między ogólnymi metodami uczenia ze wzmocnieniem i uczenia motywowanego. Agent z uczeniem ze wzmocnieniem uczy się funkcji wartości dla określonego celu (za który otrzymuje zewnętrzną nagrodę). Agent z uczeniem motywowanym uczy się wielu funkcji wartości nie tylko dla celu określonego zewnętrznie, ale także dla wszystkich wewnętrznych motywacji i abstrakcyjnych celów. Ponieważ wszystkie nagrody dostarczane z zewnątrz są widoczne w środowisku, maszynę RL można w pełni zoptymalizować.

Wręcz przeciwnie, agent z uczeniem motywowanym nie może zostać zoptymalizowany, ponieważ jego stany wewnętrzne są nieznane środowisku; w szczególności środowisko nie może obserwować całkowitej kwoty nagrody, jaką otrzymuje agent uczenia motywowanego.

Po optymalizacji agent uczenia ze wzmocnieniem jest przewidywalny, co nie ma miejsca w przypadku agenta z uczeniem motywowanym. W klasycznym uczeniu ze wzmocnieniem cele są wyznaczane przez projektanta, podczas gdy w uczeniu motywowanym w dowolnym momencie maszyna może wyznaczać i realizować nowe cele, które nie są ustalone lub zrozumiane przez projektanta lub mogą nawet nie być mu znane. Cele te zmieniają się dynamicznie wraz ze zmieniającymi się motywacjami, nie są przewidywalne na początku uczenia się i są w pełni zależne nie tylko od zewnętrznych sygnałów nagrody i bólu, ale także od całego doświadczenia uczenia się.

Uczenie ze wzmocnieniem rozwiązuje zasadniczo problem optymalizacji w celu maksymalizacji nagrody, podczas gdy uczenie motywowane oparty na bólu (ujemnej nagrodzie) rozwiązuje problem minimaksów. Różnica jest znacząca. Maksymalizacja może prowadzić do destrukcyjne zachowanie i nie zapewnia naturalnego mechanizmu przełączania w zarządzaniu konkurencyjnymi celami. Minimax prowadzi do rozwiązania wieloobiektywowego z naturalnym mechanizmem przełączania, jak omówiono w tekście. Agent z uczeniem ze wzmocnieniem jest zawsze aktywny, ponieważ system próbuje zmaksymalizować nagrodę, podczas gdy ML może odpocząć, gdy nie odczuwa bólu powyżej określonej wartości progowej. Wreszcie wyższą wydajność ML w porównaniu z RL wykazano na wielu przykładach, w których środowisko szybko się zmienia, ze złożonymi zależnościami między zasobami, które wymagają świadomości maszyny na temat tych zmian.

Tak więc w zmieniającym się środowisku maksymalizacja nagrody w oparciu o klasyczne reguły RL nie jest już najbardziej produktywna czy hierarchiczne uczenie ze wzmocnieniem, algorytm temporal difference itp.). Maszyna musi zwracać uwagę na zmiany w środowisku, które mogą wpłynąć na jej przyszłą wydajność. Chociaż ML i RL są różne, ML może skorzystać na zastosowaniu wydajnych algorytmów RL w swoich poszukiwaniach rozwiązania określonego problemu (celu). To jest system, który rządzi motywacjami maszyn i wybiera odpowiadające im cele, które je najbardziej różnicują.

4.1. Porównanie na hierarchicznym środowisku

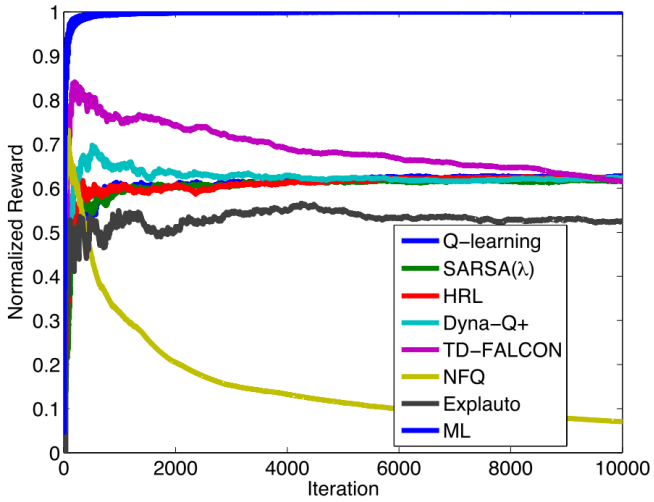
Poniżej przedstawiono rezultaty eksperymentów porównującego algorytm uczenia motywowanego z różnymi algorytmami uczenia ze wzmocnieniem, które zostały omówione w rozdziale 3. Opis opiera się na rezultatach Starzyka i innych z pracy [14].

Środowisko składa się z zasobów, które tworzą 8 poziomową hierarchię zależności między sobą, w którym odnowienie każdego z surowców jest zależne od innych zasobów. W środowisku znajduje się jeden surowiec, którego brak będzie wpływał na jeden prymitywny ból. Aby zmniejszyć ból związany z brakiem tego surowca należy wykonać akcję, która korzysta z innego zasobu, którego z czasem także będzie brakowało. Aby odnowić ilość tego zasobu będzie konieczne wykonanie innej akcji i kolejnego surowca. I takie zależności tworzy reszta zasobów.

Algorytmy, których użyto związanych z uczeniem ze wzmocnieniem to:

- Q – learning,
- SARSA(λ),
- Hierarchical Reinforcement Learning,
- Dyna – Q++,
- Explauto,
- TD – FALCON,
- NFQ RL (Neural Fitted Q iteration reinforcement learning)

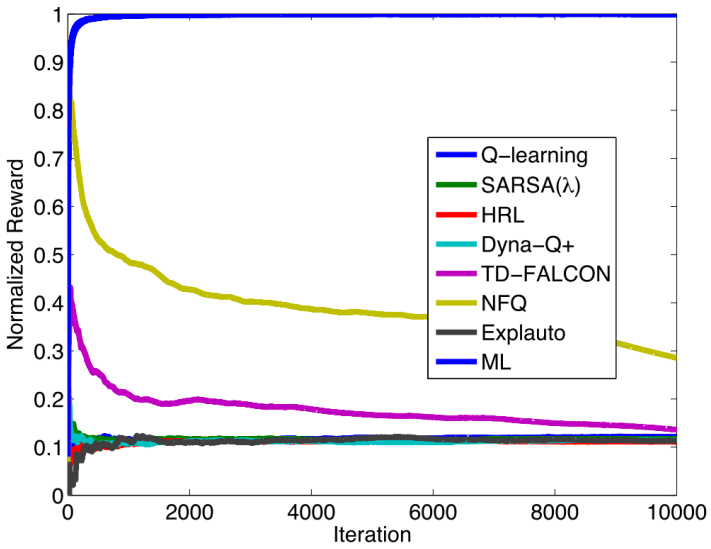
Poniżej zamieszczono porównanie wyników agenta z uczeniem motywowanym z agentam i korzystającymi z algortymów uczenia ze wzmocnieniem dla różnych współczynników S_{rate}, który opisuje współczynnik szybkości zmniejszania się zasobów, tzn. im większy współczynnik, tym zasoby się szybciej zmniejszają, a tym samym środowisko jest bardzoej wrogie.



Rys. 4.1. Porównanie resultatów agenta ML z agentami RL dla $S_{rate} = 1.0$.
Źródło: [14].

Dla współczynnika $S_{rate} = 1$ na początku większość agentów daje sobie radę porównywalnie tak samo. Sytuacja się zmienia już po około 500 iteracjach, gdzie wiele agentów przestaje sobie dawać radę w środowisku o coraz mniejszej liczbie zasobów. Po około 2000 iteracjach większość agentów stabilizuje się na tej samej wartości nagrody. Jedynym agentem, który daje sobie radę ze zmnieniającym się otoczeniem jest agent z uczniem motywowanym. Wynika to z najlepszego przystosowania do hierarchiczności świata tzn. zasobność surowców jest zależna od siebie hierarchicznie.

Jeszcze bardziej zróżnicowane rezultaty i większe różnice pomiędzy agentem z uczeniem motywowanym i agentem z uczeniem ze wzmocnieniem pojawiają się dla współczynnika $S_{rate} = 8.0$. Rezultaty widać na rysunku 4.2.



Rys. 4.2. Porównanie resultatów agenta ML z agentami RL dla $S_{rate} = 8.0$.
Źródło: [14].

Dla bardzo wrogiego środowiska, agent z uczeniem motywowanym daje sobie wciąż radę, natomiast większość agentów z uczeniem ze wzmocnieniem, bardzo szybko stabilizuje się na niskiej wartości nagrody, ponieważ środowisko stało się bardzo szybko wrogie poprzez niskie zasoby surowców. Agent z uczniem motywowanym jest w stanie szybko stworzyć hierachię abstrokcyjnych bóli, których obniżenie sprawia, że brakujące zasoby mogą zostać odbudowane.

5. Środowisko symulacyjne

Gdy projektowano pierwsze maszyny, które miały wykonywać konkretne zadania, opierano się na dokładnych modelach matematycznych opisujących fizykę. Nie istniały narzędzia do weryfikacji skuteczności działania danego urządzenia. Dopiero rozwój technologii komputerowych i wzrost mocy obliczeniowej procesorów umożliwił bardzo szybką weryfikację stworzonych maszyn na komputerach, które w kilka sekund były w stanie sprawdzić poprawność obliczeń. Duże znaczenie symulacje zaczęły odgrywać kiedy testowanie projektowanych maszyn mogło sprawiać zagrożenia dla ludzi czy samego środowiska, w którym były testowane.

Ze względu na coraz niższą cenę sprzętu komputerowego i rozwojowi oprogramowania, coraz więcej osób może w swoich domach uruchamiać modele wielu urządzeń bez ryzyka uszkodzenia czegokolwiek i móc prowadzić swoje badania. W poniższej pracy konieczne było zastosowanie takiego środowiska symulacyjnego, ponieważ testowane rozwiązania są nowe i agenci uczenia motywowanego mogą czasem wykonywać czynności, które nie były uwzględnione przez projektanta, zgodnie z tabelą 4.1.

W poniżej pracy uczenie motywowane będzie testowane na robocie mobilnym w środowisku symulacyjnym Gazebo Sim [15].

5.1. Symulator Gazebo

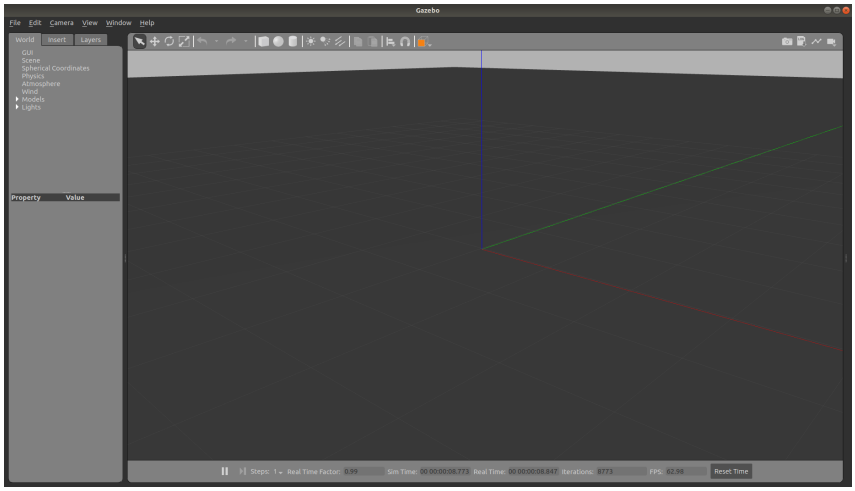
Rozwój symulatora Gazebo rozpoczął się jesienią 2002 roku na Uniwersytecie Południowej Kalifornii. Oryginalnymi twórcami byli dr Andrew Howard i jego uczeń Nate Koenig. Koncepcja symulatora o wysokiej wierności wynikała z potrzeby symulacji robotów w środowiskach zewnętrznych w różnych warunkach. Jako komplementarny symulator do Stage, nazwa Gazebo została wybrana jako struktura najbliższa scenie zewnętrznej. Nazwa utknęła pomimo faktu, że większość użytkowników Gazebo symuluje środowisko wewnętrzne.

Symulator ten ma zintegrowane silnik fizyki ODE (Open Dynamics Engine). Wykorzystuje renderowanie korzystając z OpenGL. Dzięki temu możeliwe jest symulowanie wielu rodzajów sensorów:

Rys. 5.1. Logo Gazebo Sim. Źródło: [15].

- ciśnieniomierz,
- wysokościomierz,
- kamera (RGB, głębi),
- czujnik kontaktu (dotyku),
- czujnik nacisku,
- GPS,
- lidar,
- skaner laserowy,
- IMU (ang. *inertial measurement unit*),
- magnetometr,
- czujnik odległości,
- inne.

Tak duży zasób czujników umożliwia sprawdzanie na wiele sposobów jaki zestaw sensorów może być wykorzystany dla konkretnych celów oraz sprawia, że testowanie nowych funkcjonalności bez ryzyka uszkodzenia robota czy środowiska.



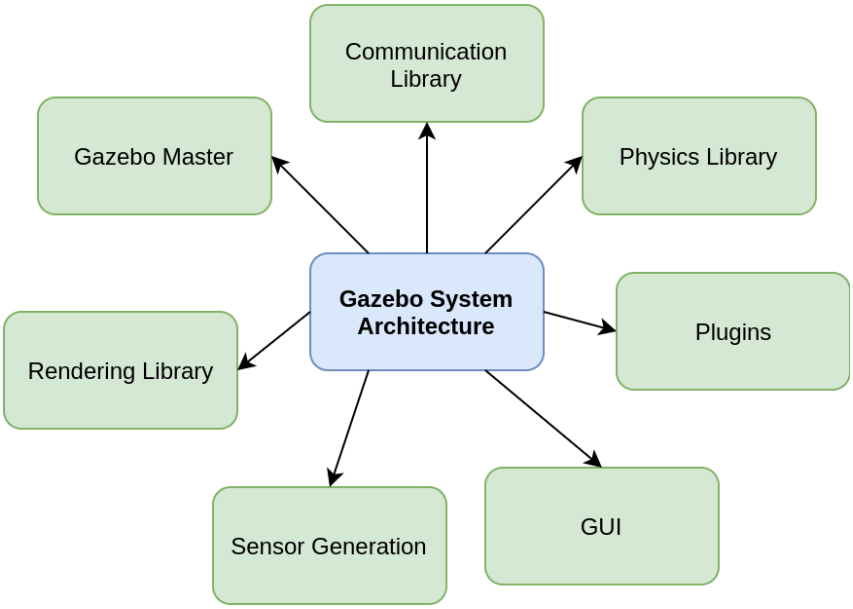
Rys. 5.2. Główne okna symulatora Gazebo. Źródło: opracowanie własne.

Graficzny interfejs umożliwia szybkie projektowanie nowych robotów czy elementów środowiska. Wygląd głównego okna, w którym wykonywana jest większość pracy podczas projektowania środowiska symulacyjnego pokazano na rysunku 5.2.

Dodatkowowo w Gazebo oprócz silnika fizycznego ODE można skorzystać także z innych bardzo popularnych silników fizyczny jak: Bullet. Dzięki rzeczywistemu renderowaniu sceny w kamerach możliwe jest testowanie systemów wizyjnych. Takie na pewno muszą być zastosowane dla agenta z uczeniem motywowanym, ponieważ to jeden z podstawowych sensorów stosowanych przez człowieka na etapie poznawania nowego środowiska.

Środowisko Gazebo Sim składa się z kilku elementów, które sprawiają, że możliwe jest uruchomienie symulacji robota. Są to zgodnie z [16].

- **World files** – pliki zawierające opis danego środowiska, w którym robot ma działać, można w nich modyfikować parametry świata, np. siłę grawitacji,
- **Model files** – pliki w formacie SDF zawierające opisy struktury robota,
- **Environment variables** – zmienne środowiskowego, które umożliwiają w prosty sposób uruchamiać Gazebo i ładować modele robotów do symulacji,
- **Gazebo Server** – główny program symulacji, który wykonuje wszelkie obliczenia tj. liczenie kolizji, symulowanie czujników,
- **Graphical Client** – umożliwia wyświetlanie wyników obliczeń z serwera Gazebo,
- **Plugins** – małe biblioteki, które umożliwiają w prosty sposób wchodzić w interakcję w symulacją poprzez tworzenie własnych sensorów czy wtyczek modyfikujących parametry świata z poziomu kodu.



Rys. 5.3. Architektura systemu Gazebo. Źródło: opracowanie własne.

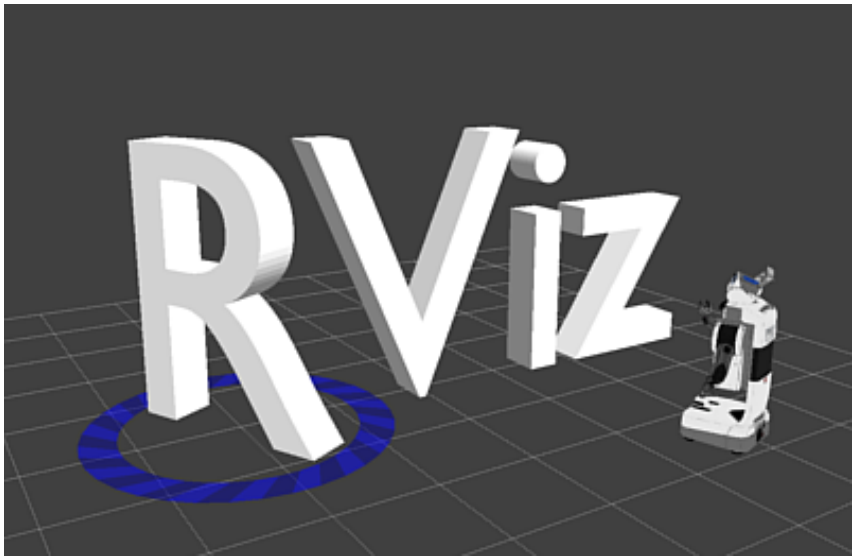
Architektura całego systemu umożliwia proste projektowanie sensorów, robotów czy całych środowisk do symulowania różnych scenariuszy środowiska jak parametry, np. siła grawitacji czy opory powietrza, jeżeli symulowane są roboty latające jak drony.

Gazebo używa rozproszonej architektury z rozdzielonymi bibliotekami do symulacji fizyki, renderowania sceny, interfejsu użytkownika, komunikacji i generowaniu sensorów. Dodatkowo w celu uruchomienia wszystkich możliwości symulatora, wcześniej wspomnianych: *gzserver* do symulowania fizyki, renderowania i sensorów oraz *gzclient*, który umożliwia wyświetlanie interfejsu użytkownika. Elementy architektury Gazebo przedstawiono na rysunku 5.3:

5.2. Wizualizacja w RViz

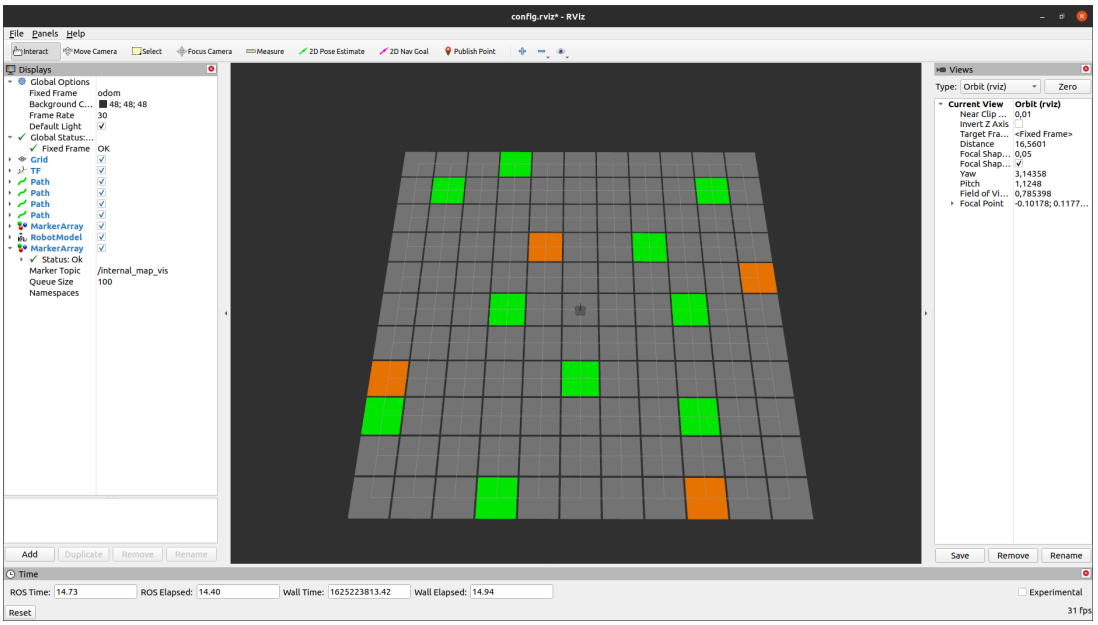
Kolejnym bardzo przydatnym narzędziem stosowanym przy rozwijaniu oprogramowania na roboty jest program RViz. Służy on do wizualizacji danych z różnych sensorów w czytelny i przejrzysty sposób. Dostarcza także szereg tzw. interaktywnych markerów, z którymi użytkownik może wchodzić w interakcję i wydawać komendy do robota, który działa na innej maszynie.

Podgląd danych z kamer czy chmur punktów oraz połączenie przez sieć z robotem, możliwe jest zdalne sterowanie robotem przy jednoczesny wglądzie do aktualnych danych z kamer czy odometrii robota. RViz to bardzo prosty przykład interfejsu człowiek komputer HMI (ang. *human – machine interface*) znany szeroko ze sterowników przemysłowych. Sam program dostarcza API (ang. *application programming interface*), który umożliwia rozszerzanie dostępnych funkcjonalności o bardziej dostosowane do wymagań danego projektu.



Rys. 5.4. Logo programu RViz oraz robot PR2. Źródło: [17]

W tym projekcie został on zastosowany do podglądu aktualnej pozycji robota na przygotowanej mapie oraz wizualizacji samej mapy, co obrazuje figura 5.5.

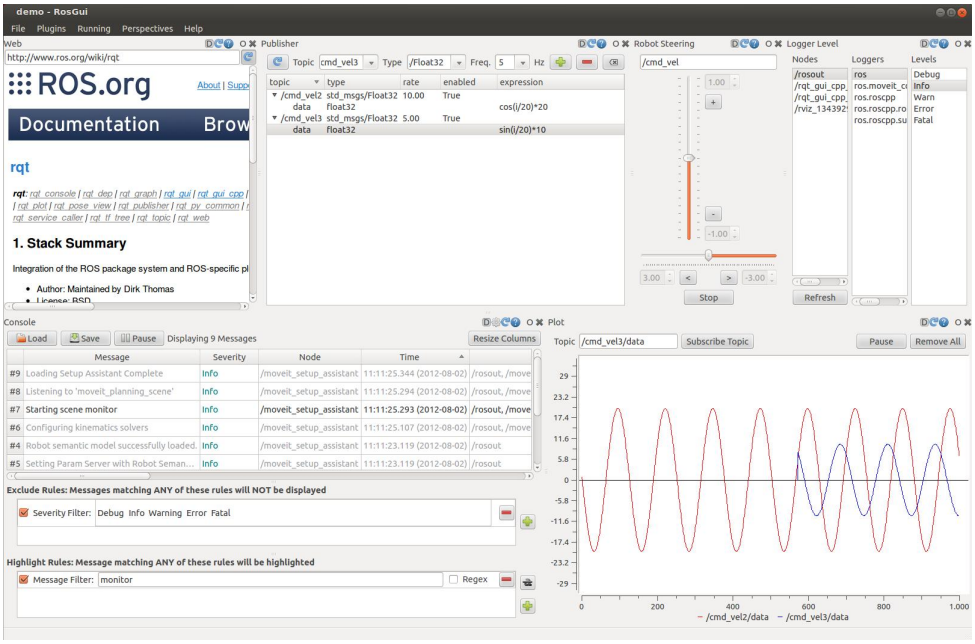


Rys. 5.5. Przykładowy zrzut ekranu aplikacji RViz z wyświetlonym modelem robota i mapą. Źródło: opracowanie własne.

5.3. Zbiór bibliotek RQT

Ostatnim narzędziem, które znacznie usprawnia pracę w trakcie rozwijania oprogramowania na wszelkiego rodzaju roboty jest platforma programistyczna (ang. *framework*) RQT. Oparta

ona została na szeroko znanej bibliotece QT, służącej do projektowania interfejsów graficzny tzw. GUI (ang. *graphical user interface*). Stosując RQT można łatwo projektować interfejsy graficzne do wygodnego sterowania robotem. Dzięki temu, że RQT korzysta z QT, możliwe jest stosowanie narzędzi dostarczanych przez QT, tj. QT Designer. Platforma RQT dostarcza podstawowe interfejsy, dzięki którym nasza aplikacja ma bezpośrednie połączenie do systemu robotycznego ROS, który został użyty w tym projekcie. Każda z takich aplikacji zachowuje się jak plugin do RQT. Możemy otworzyć wiele różnych pluginów w jednym oknie, aranżować ich rozmieszczenie w oknie aplikacji oraz dynamiczne włączać i wyłączać. Przykładowy widok okna RQT na obrazku 5.6 (proszę zwrócić uwagę na plugin z załadowaną stroną internetową w lewym górnym rogu).



Rys. 5.6. Przykładowy widok załadowanych pluginów do RQT. Źródło: [18]

6. Agent w nieznanym środowisku

Po zapoznaniu się z zagadnieniami związanymi z uczeniem motywowanym, środowiskiem symulacyjnym, należało zdefiniować środowisko w jakim robot mobilny może się poruszać. Jako robot testowy, użyto bardzo popularnej platformy Turtebot3 w wersji Waffle ([19]).

Ze względu na brak dostępu do prawdziwego robota oraz problemów jakie mogłoby sprawiać zaprojektowanie środowiska w skali 1 : 1, zdecydowano o symulacyjnym rozwiązaniu. Omawiane w poprzednich rozdziałach Gazebo, RViz i RQT zostały użyte do zaprojektowania środowiska, sterowaniu robota i wizualizacji aktualnej pozycji, stanu wewnętrznego robota oraz stanu środowiska.

Uczenie motywowane jest jeszcze mało zbadanym zagadnieniem, a celem tej pracy jest zbadanie wykorzystania algorytmów uczenia motywowanego na robocie mobilnym. Definiowane są dla niego pewne prymitywne bóle i akcje jakie może on wykonać w danej chwili czasu. W pierwszym rozwiązaniu ograniczono do zdefiniowania kilku prymitywnych bóli. Agentem jest robot mobilny, który ma możliwość wykonania ruchu na mapie

Bibliografia

[1] Alan Turing. „Computing Machinery and Intelligence". W: *Mind* 236 (1950), s. 433–460. DOI: *https://doi.org/10.1093/mind/LIX.236.433*.

[2] Yann LeCun. „Gradient-based learning applied to document recognition". W: (1998). DOI: *http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf*.

[3] Alex Krizhevsky, Ilya Sutskever i Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks". W: (2012). DOI: *https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf*.

[4] Stanford Vision Lab. *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. URL: *http://www.image-net.org/challenges/LSVRC/*. dostęp: 30.08.2020.

[5] DeepMind. *AlphaGo*. URL: *https://deepmind.com/research/case-studies/alphago-the-story-so-far*. dostęp: 30.08.2020.

[6] Rolf Pfeifer i Josh Bongard. *How the Body Shapes the Way We Think: a New View of Intelligence*. 2007. DOI: *10.7551/mitpress/3585.001.0001*.

[7] John Stewart. „Cognition without neurons: Adaptation, learning and memory in the immune system." W: (1993).

[8] Janusz Starzyk. „Motivation in Embodied Intelligence". W: (2008).

[9] M. Csikszentmihalyi i M. Csikszentmihalyi. *Creativity: Flow and the Psychology of Discovery and Invention*. Harper Perennial Modern Classics. HarperCollinsPublishers, 1996. ISBN: 9780060171339.

[10] Janusz Starzyk i in. „Motivated Learning for the Development of Autonomous Systems". W: (2012). DOI: *https://doi.org/10.1016/j.cogsys.2010.12.009*.

[11] Janusz Starzyk. „Motivated Learning for Computational Intelligence". W: (2011). DOI: *https://www.researchgate.net/publication/267375356_Motivated_Learning_for_Computational_Intelligence*.

[12] DeepMind. „Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". W: (2019).

[13] Bram Bakker i Jurgen Schmidhuber. „Hierarchical Reinforcement Learning Based on Subgoal Discovery and Subpolicy Specialization". W: (2004).

[14] James Graham i in. „A comparative study between motivated learning and reinforcement learning". W: (2015). DOI: *https://doi.org/10.1109/IJCNN.2015.7280723*.

[15] Open Source Robotics Foundation. *Gazebo Sim*. URL: *http://gazebosim.org/*.

[16] Open Source Robotics Foundation. *Gazebo Components*. URL: *http://gazebosim.org/tutorials?tut=components*.

[17] Open Source Robotics Foundation. *RViz*. URL: *http://wiki.ros.org/rviz*.

[18] Open Source Robotics Foundation. *RQT*. URL: *http://wiki.ros.org/rqt*.

[19] ROBOTIS. *Turtlebot3 manual*. URL: *https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/*.