Robotyka: techniki, funkcje, rola społeczna Cz. 1. Techniczne podstawy inteligencji i bezpieczeństwa robotów

Cezary Zieliński
Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, Instytut Automatyki i Informatyki Stosowanej, Nowowiejska 15/19, 00-665 Warszawa

Streszczenie: Aby ocenić, jaki wpływ będą miały roboty na społeczeństwo, należy skrupulatnie przeanalizować obecny stan wiedzy, a w szczególności wskazać fundamentalne problemy, które jeszcze nie zostały rozwiązane, mające istotne znaczenie dla potencjalnych zmian społecznych powodowanych rozwojem robotyki. Wspomniany wpływ zależy od inteligencji robotów, więc ten aspekt dominuje w przedstawionej tu analizie. Rozważania zostały podzielone na trzy części: 1) analizę czynników technicznych wpływających na inteligencję i bezpieczeństwo robotów, 2) analizę obecnych możliwości robotów, 3) analizę przewidywań dotyczących rozwoju robotyki, a w konsekwencji poglądów na skutki tego rozwoju dla społeczeństwa. Niniejszy artykuł poświęcony jest pierwszemu z wymienionych tu trzech zagadnień.

Słowa kluczowe: robot, system robotyczny, sztuczna inteligencja, cyberbezpieczeństwo robotów

1. Motywacja

Ostatnio nasiliły się w prasie i opracowaniach o bardziej naukowym charakterze katastroficzne doniesienia przestrzegające przed rozwojem sztucznej inteligencji i robotyki. W wersji skrajnej wieszczą one, iż ludzie staną się zbędni, a maszyny przejmą władzę nad Światem. W istocie, słowo robot pojawiło się po raz pierwszy w sztuce Karela Čapeka o tytule "R.U.R – Rossumovi Univerzální Roboti" [145] napisanej w 1920 r. Jej prapremiera odbyła się w 1921 r. Już w tym pierwszym dziele poświęconym robotom zawładnęły one naszą planetą. Początkowo posłuszne maszyny obdarzono uczuciami, co doprowadziło je do wniosku, że ludzie, którzy wskutek pojawienia się robotów stali się bezproduktywni, a więc są niepotrzebni. Bunt robotów doprowadził niemal do wyplenienia ludzi. Ten scenariusz oparty jest na przeświadczeniu, że roboty mogą się stać bardziej inteligentne od ludzi, a co więcej mogą podejmować decyzje bez uwzględniania ludzkich zasad moralnych. Wprawdzie wyeliminowanie ludzi przez roboty wydaje się mało prawdopodobne, ale z faktu, że są stosowane w coraz to większej liczbie dziedzin gospodarki wynika, iż będą miały istotny wpływ na społeczeństwo. Motywacją do napisania niniejszego artykułu była próba rozstrzygnięcia, na ile roboty zmienią zasady funkcjonowania społeczeństw.

Odpowiedź na tak postawione pytanie wymaga określenia aktualnego stanu robotyki, a w szczególności pokazania, jaki jest stan podstawowych czynników kształtujących zarówno inteligencję jak i możliwości ruchowe tych urządzeń. Istotne jest też tempo zachodzących zmian technologicznych. Na podstawie literatury dotyczącej oddziaływania maszyn na zachowanie ludzi, należy przewidzieć, jaki wpływ będą miały roboty na społeczeństwo. W dużej mierze jest to uwarunkowane stopniem akceptacji tych urządzeń przez ludzi, tworzonym prawem oraz prawami ekonomii. Zawarta tu analiza jest wynikiem studiów literaturowych i streszcza główne nurty w debacie toczącej się nad tym problemem od lat. Jak każda prognoza, obarczona jest niepewnością, tym większą im dłuższy jest horyzont przewidywania.

Ta część artykułu koncentruje się przede wszystkim na tych aspektach robotyki, które związane są z inteligencją i sterowaniem robotów. Oczywiście inteligencja nie jest oderwana od możliwości ruchowych i percepcyjnych tych maszyn, więc i te zagadnienia będą pojawiać się w tym tekście.

Autor korespondujący:
Cezary Zieliński, cezary.zielinski@pw.edu.pl

Artykuł recenzowany
nadesłany 06.11.2022 r., przyjęty do druku 21.11.2022 r.

2. Podstawowe czynniki kształtujące inteligencję robotów

Aby ocenić, jaki wpływ na społeczeństwo będą mieć roboty, trzeba przyjrzeć się nieco bardziej szczegółowo, jak obecnie próbuje się wytworzyć inteligentne zachowania tych urzą-

dzeń. Ponadto należy dokonać projekcji, do czego będą zdolne w przyszłości, a to można jedynie zrobić analizując obecne trendy rozwojowe. Niemniej jednak wpierw trzeba skierować snop światła na fundamentalne czynniki wpływające na inteligencję robotów. By uporządkować dyskusję, na wstępie należy nieco lepiej scharakteryzować przedmiot zainteresowań.

2.1. Robot

Przystoi na wstępie dobrze określić przedmiot zainteresowania artykułu. Niestety nie ma jednej ogólnie przyjętej definicji robota. Większość definicji cierpi albo na zbytnią szczegółowość, wykluczającą urządzenia powszechnie uważane za roboty, albo na zbytnią ogólnikowość, umożliwiającą zaliczenie do robotów urządzeń, które trudno jest uznać za takowe. Joseph Engelberger, który wspólnie z Georgem Devolem założył firmę Unimation, w której wytworzono pierwsze roboty przemysłowe [103], gdy został zapytany o definicję robota, wyrażając swą frustrację brakiem adekwatnego określenia, powiedział: I can't define a robot, but I know one when I see one. (Nie potrafię zdefiniować robota, ale rozpoznaję go, gdy go widzę.) [34].

W tym artykule przyjęto bardzo szeroką definicję robota, niemniej jednak wyklucza ona oprogramowanie nieoddziałujące na środowisko fizyczne. Tak więc, robot stanowi system, który wpływa na otoczenie fizyczne przez swoje efektory (manipulatory, chwytaki, nogi, koła etc.), określa stan środowiska za pomocą swych receptorów (czujników) oraz obdarzony jest imperatywem wewnętrznym do realizacji zadań zdefiniowanych albo przez jego projektanta albo przekazywanych pośrednio lub bezpośrednio przez człowieka. System sterowania robota na podstawie danych uzyskanych z receptorów oraz definicji zadania, stanowiącego wspomniany imperatyw do działania, steruje efektorami tak, aby owo zadanie zrealizować. Działanie robota w środowisku fizycznym oczywiście nie wyklucza jego interakcji z cyberprzestrzenią. System robotyczny jest pojęciem bardziej ogólnym, składając się z co najmniej jednego robota oraz ewentualnie z dodatkowych urządzeń współpracujących.

2.2. Inteligencja robotów

Już w latach 80. XX w. przewidywano, iż sztuczna inteligencja będzie miała kluczowe znaczenie dla robotyki. Michael Brady, jeden z prekursorów robotyki, określił ją, jako dziedzinę badającą połączenie akcji z percepcją [21]. Wskazał ponadto, że jeżeli roboty mają się zachowywać inteligentnie, to centralną rolę odegra w tym połączeniu sztuczna inteligencja. Niestety jednoznaczne określenie czym jest sztuczna inteligencja jest bodaj jeszcze trudniejsze niż zdefiniowanie robota. George Luger i William Stubblefield przyjmują, że jest to gałąź informatyki zajmująca się automatyzacją inteligentnego zachowania [93]. Natomiast Stuart Russell i Peter Norvig, w klasycznym dziele poświęconym sztucznej inteligencji [122], pokusili się o podział jej definicji na kategorie. Z ich analizy wynikało, że istnieją cztery kategorie systemów korzystających ze sztucznej inteligencji. Są to systemy: 1) myślące, jak ludzie, 2) działające, jak ludzie, 3) myślące racjonalnie i 4) działające racjonalnie. Pierwsze dwie kategorie definicji odnoszą się do sposobu funkcjonowania ludzi, a więc wykorzystywane są przez kognitywistykę, czyli naukę o obserwacji i analizie działania oraz modelowaniu umysłu i mózgu, natomiast dwie ostatnie jedynie wymagając racjonalności, dotyczą systemów osiągających dobre efekty swego działania w kontekście posiadanej wiedzy, a więc dobrze nadają się do wykorzystania przez nauki inżynierskie, w szczególności do optymalizacji działania systemów.

Aby oderwać rozważania na temat inteligencji od kontekstu ludzkiego Alan Turing opracował eponimiczny test [143]. Test ten jest swoistą grą polegającą na zgadnięciu przez człowieka, na podstawie prowadzonej konwersacji z niewidocznym interlokutorem, czy jest to człowiek czy maszyna. Jeżeli jest to maszyna, a testujący nie jest w stanie stwierdzić tego z pewnością, to mamy do czynienia ze sztuczną inteligencją. Test ten nie jest powszechnie wykorzystywany, ze względu na subiektywność osądu, czy obserwowane działanie jest inteligentne czy nie.

Yann LeCun uważa, że istotą inteligencji jest zdolność do przewidywania. Czasami wyraża to jako zdolność do uczenia się przewidywania, a więc podstawą inteligencji są zdolność przewidywania oraz umiejętność uczenia się. Wszakże należy pamiętać, że te zdolności potrzebują odpowiednich zasobów wiedzy (zgromadzonej lub dostępnej do zdobycia ze środowiska), aby rozumowanie przebiegało szybko zasoby te muszą być odpowiednio zorganizowane. W szczególności maszyny potrzebują wiedzy zdroworozsądkowej, którą ludzie zdobywają w dzieciństwie, i na podstawie której są w stanie przewidywać konsekwencje swoich czynów. LeCun rysuje interesującą koncepcję badawczą mająca prowadzić do budowy maszyn inteligentnych w swojej wizji [90]. Wizja ta dotyczy badań w celu stworzenia inteligentnych maszyn, które uczą się, jak zwierzęta, które potrafią wnioskować i planować, i których zachowanie jest określone ogólnymi celami, a nie na sztywno zaprogramowane bądź zewnętrznie nadzorowane albo motywowane otrzymywaniem zewnętrznych nagród. LeCun spodziewa się, że rozwiązaniem problemu będzie architektura systemu złożona z wielu głębokich sieci neuronowych imitujących poszczególne obszary ludzkiego mózgu. Istotną rolę w jego podejściu odgrywa model świata, który zazwyczaj był ignorowany przy wykorzystywaniu sieci neuronowych. Pozwala on przewidywać rezultaty działania robota, a więc ma fundamentalne znaczenie dla planowania działań.

2.3. Kategoryzacja robotów

Pomimo że istnieje wiele różnych kategorii robotów, to nie dopracowano się ścisłej ich klasyfikacji. Jako zgrubne kryterium można zastosować stopień strukturyzacji (uporządkowania) środowiska, w którym dany robot działa. Wtedy można wyróżnić trzy główne kategorie robotów: przemysłowe, usługowe oraz terenowe. Dwie ostanie kategorie w literaturze bywają wspólnie określane jako roboty zaawansowane (ang. *advanced robots*) [132]. Roboty przemysłowe działają w otoczeniu w pełni strukturalnym, gdzie położenia wszystkich obiektów są ściśle określone, więc roboty tej kategorii nie muszą mieć receptorów albo mają ich niewiele, bo w pełni mogą polegać na swojej wiedzy o pozycjach przedmiotów znajdujących się w środowisku. Roboty terenowe działają w otoczeniu, którego strukturę trudno określić, np. las, sad albo pole. Te roboty mogą zrealizować swoje zadania tylko przy wykorzystaniu różnorodnych receptorów. Roboty usługowe działają w środowisku o częściowo określonej strukturze, np. w pomieszczeniach mieszkalnych, biurowych czy szpitalnych lub gastronomicznych. Roboty tej kategorii również potrzebują receptorów do realizacji zadania. Ta podstawowa kategoryzacja robotów może być dalej rozbudowywana poprzez zastosowanie dodatkowych kryteriów klasyfikacji, szczególnie że wymienione wyżej podstawowe kryterium jest ciągłe, a nie dyskretne, a więc roboty przemysłowe i terenowe jawią się jako skrajne kategorie robotów usługowych. Dlatego, z jednej strony, roboty usługowe, działające w środowisku częściowo strukturalnym, obejmują szczególnie szeroką rodzinę robotów, a z drugiej, obecne roboty przemysłowe nie muszą działać w otoczeniu w pełni strukturalnym, a terenowe mogą realizować swoje zadania w środowisku częściowo strukturalnym, np. na drogach publicznych. Wśród robotów usługowych wymienia się takie podkategorie jak roboty osobiste (ang. *personal*), stosowane w domach, roboty profesjonalne (ang. *professional*), stosowane w biurach lub szpitalach, roboty asystujące (ang. *assistive*) lub kompani (ang. *companions*) wspomagający ludzi niepełnosprawnych i starszych, czy koboty (ang. *collaborative robots*, *co-robots*, *cobots*) [161]. Te ostatnie są robotami bezpośrednio współpracują-

cymi z ludźmi przy produkcji, a więc można je zakwalifikować zarówno do kategorii robotów przemysłowych jak i usługowych. Powstały one wskutek rozwoju technologii potrzebnych do konstrukcji robotów usługowych, a technologie te dopiero wtórnie zostały zastosowane w przemyśle. Stało się tak, ponieważ uznano, że są na tyle bezpieczne, że dzięki nim roboty mogą wchodzić w bezpośrednie interakcje z ludźmi, a więc standardowo stosowane zewnętrzne bariery i ogrodzenia odgradzające roboty przemysłowe od pracowników w tym przypadku nie są konieczne. Jak widać, ścisłe rozgraniczenie kategorii robotów jest trudne, bo jak określić ścisłe kryteria podziału, które nie zachodzą wzajemnie na siebie. Dlatego podany podział będzie tu stosowany jedynie dla ogólnej orientacji.

Robot musi być w stanie zrealizować powierzone mu zadanie, więc sposób przekazania mu tego zadania staje się kluczowy. Stąd musimy zająć się zarówno sposobami programowania robotów, jak i architekturami tych systemów.

2.4. Programowanie robotów

Istnieją dwa poglądy na temat programowania robotów. Pierwszy, węższy, definiuje programowanie robotów jako sposób określania zadania, które robot ma wykonać. Drugi, szerszy, postuluje, że programowanie robotów jest tożsame z wytwarzaniem oprogramowania dla robotów, a więc obejmuje także tworzenie ich sterowników. Jak zwykle w przypadku robotyki, nie jest to podział dychotomiczny, ponieważ wiele robotów przeznaczonych jest do autonomicznego wykonywania zadań, które zostały określone na etapie tworzenia ich sterowników, a więc trudno jest rozróżnić część odpowiedzialną za wykonanie zadania od pozostałych części, które sterują urządzeniem. Dlatego lepiej jest przyjąć tę szerszą definicję.

Od momentu powstania pierwszych robotów korzystających z serwomechanizmów przemysł preferował uczenie robotów, podczas gdy ośrodki akademickie pracowały nad specjalizowanymi językami programowania robotów. W tym przypadku uczenie rozumiane jest jako przemieszczanie przez operatora ramienia robota przemysłowego do pozycji, które muszą być osiągnięte, by zrealizować zadanie, i zapamiętywanie ich. Zapamiętane pozycje stanowią program, który jest odtwarzany. Program nauczony w ten sposób może być wielokrotnie odtwarzany w celu powtarzania realizacji zadania. Istnieją dwie formy uczenia: PTP (point to point) oraz CP (ang. *continuous path*). Pierwsza forma polega na doprowadzaniu końcówki manipulatora do odległych od siebie pozycji specyficznych dla wykonywanego zadania, by je zapamiętać. W trakcie odtwarzania programu sterownik dokonuje interpolacji trajektorii. Druga forma zmusza sterownik do automatycznego zapamiętywania pozycji na trajektorii co bardzo krótki czas, dzięki czemu unika się interpolacji. Pierwsza forma wykorzystywana jest do obsługi maszyn, transportu obiektów lub montażu, natomiast druga np. do malowania natryskowego. Zaletą uczenia jest prostota programowania, która nie wymaga zbyt wysokich kwalifikacji od programisty. Niestety modyfikacja fragmentu zadania bądź wykorzystanie odczytów czujników stanowi tu niebagatelny problem, nie wspominając o braku czytelnej dokumentacji stworzonego programu.

Wad tych nie mają programy zapisane w specjalizowanych językach programowania robotów, ale tutaj kompetencje programisty muszą być o wiele wyższe, by zapisać program w formie tekstowej. Co więcej, roboty nawet tego samego typu nieznacznie się różnią między sobą, więc ich końcówki będą osiągały pozycje zadane z różną dokładnością. To powoduje konieczność kalibracji albo mechanizmu ramienia albo programu. W toku rozwoju robotów przemysłowych oba sposoby programowania zostały zintegrowane [163]. Obecnie kluczowe, ale nieliczne, pozycje są uczone, a pozostałe określane są w tekście programu względem tych nauczonych.

Specjalizowane języki programowania robotów ponadto oferują wszelkie możliwości, które posiadają proceduralne uniwersalne języki programowania. Fakt ten spowodował odejście od języków specjalizowanych na korzyść języków uniwersalnych, takich jak C, C++ czy Pascal. Struktura programu zapisywana była w języku uniwersalnym, natomiast elementy specyficzne dla robota, takie jak instrukcje ruchu czy obliczenia geometryczne, realizowane były przez biblioteki, zapisane w tym samym języku uniwersalnym (np. PASRO (ang. *Pascal for Robots*) [17, 18] lub RCCL (ang. *Robot Control C Library*) [67]). Wprawdzie zmniejszyło to czytelność programów w stosunku do tych zapisanych w językach specjalizowanych, ale likwidowało konieczność tworzenia kompilatorów bądź interpreterów tych języków, i zmniejszało koszt wprowadzania modyfikacji do języka przy inkorporacji nowych urządzeń (np. czujników) do systemu. Niestety biblioteki nie dawały wskazówek, jak tworzyć oprogramowanie o odpowiedniej architekturze. Tej niedogodności miały zaradzić programowe struktury ramowe (ang. *programming frameworks*), np. Player [40, 60, 144], OROCOS (ang. *Open Robot Control Software*) [31, 32], ROS (ang. *Robot Operating System*) [120], YARP (ang. *Yet Another Robot Platform*) [57, 102], ORCA [22, 23], MRROC++ (ang. *Multi-Robot Research Oriented Controller based on C++*) [156, 158–160, 168] i inne [166]. Oferowały one wzorce użycia, co stanowiło pewną wskazówkę dla programistów, jak poprawnie konstruować programy. Większość struktur ramowych skupiała się na tzw. idiomach, a więc wzorcach dotyczących kodu w konkretnym języku programowania. Nieliczne odnosiły się do wzorców projektowych lub architektonicznych [77].

Zarówno biblioteki, jak i struktury ramowe koncentrowały się na fazie implementacji w budowie oprogramowania, ignorując specyfikację. Natomiast dobre praktyki wytworzone na gruncie inżynierii oprogramowania podkreślają konieczność tworzenia specyfikacji przed przystąpieniem do implementacji [125]. By temu problemowi zaradzić powrócono do języków specjalizowanych, ale w innej postaci. Języki zorientowane na konkretną dziedzinę DSL (ang. *Domain-Specific Language*) wywodzą się z ogólnego podejścia do projektowania zwanego inżynierią opartą na modelu MDE (ang. *Model Driven Engineering*) [30, 108]. Model uwypukla najistotniejsze właściwości systemu, pomijając szczegóły implementacyjne. Zastosowanie MDE do tworzenia oprogramowania prowadzi do automatycznej generacji kodu tworzonego systemu – przynajmniej częściowo. W robotyce stosuje się zarówno języki modelowania ogólnego przeznaczenia, takie jak UML [117], jak i języki specjalnie opracowane do modelowania systemów robotycznych [30]. Ponieważ różne aspekty systemu tworzone są za pomocą różnych języków dziedzinowych, potrzeba wielu narzędzi, by przetworzyć i zweryfikować model, aby w rezultacie powstał kod pożądanego systemu. Wskazane jest, by zestaw narzędzi (ang. *toolchain*) był jak najmniej liczny, bo w przeciwnym razie trudność jego opanowania powoduje niechęć programistów do jego używania. W tym nurcie stworzono również sparametryzowany meta-model, który może być przekształcony w program sterownika dowolnego robota poprzez określenie funkcyjnych parametrów oraz struktury hierarchicznej sieci Petriego [55].

Chcąc uczynić programowanie robotów, rozumiane jako przekazywanie im zadań do wykonania, jeszcze bardziej zbliżone do tego, jak ludzie przekazują sobie polecenia, zaczęto pracować nad programowaniem przez demonstrację (ang. *programming by demonstration*). Wykorzystując kamery, mikrofony oraz inne czujniki robot przygląda się pracy człowieka oraz słucha jego komentarzy, w ten sposób jest instruowany, jak ma wykonać zadanie [14, 15]. Dzięki temu kwalifikacje człowieka w zakresie programowania robotów stają się zbędne, natomiast ludzki nauczyciel musi umieć wykonać czynności, by samodzielnie zrealizować zadanie, które później robot ma powtórzyć. Osiągnięcie takiej formy programowania jest bardzo trudne, więc

proponowane są rozwiązania prostsze. W skrajnym podejściu za programowanie przez demonstrację można uznać uczenie typu PTP lub CP, ale nie o to tu chodzi. Dlatego bada się zastosowanie czujników sił i dotyku oraz kamer do wspomagania demonstracji [112]. Próbuje się też stosować rozszerzoną rzeczywistość w trakcie programowania robota. Ponadto stara się definiować zadania poprzez ograniczony zestaw sparametryzowanych umiejętności [136] określonych na podstawie standardowych procedur operacyjnych dla robotników produkcyjnych [116]. Podstawowymi problemami są [14]:

— kiedy i kogo robot ma imitować? – usunięcie z prezentacji wykonania zadania czynników nieistotnych jest trudnym zagadnieniem,
— jak ma być imitowany ruch człowieka, skoro struktury kinematyczne oraz parametry odnoszące się do wymiarów robota i człowieka są różne,
— roboty mają słabo rozwinięty zmysł dotyku, więc bardzo źle rozpoznają fakturę i temperaturę przedmiotów.

Do tego zestawu podstawowego należałoby dodać zdolność do uogólniania swych obserwacji, by na podstawie jednej prezentacji można było wykonywać podobne zadania – byłby to krok w kierunku sztucznej ogólnej inteligencji (AGI). Pomimo istotnego postępu w metodach programowania robotów, zlecanie zadań robotowi w taki sposób, jak to robią ludzie między sobą, jest jeszcze dość odległą przyszłością.

2.5. Architektury systemów robotycznych

Aby ocenić do czego zdolne są roboty, trzeba nieco więcej powiedzieć, jak są one sterowane i jak wyglądają ich systemy sterowania. Dlatego skoncentrujemy się na architekturach systemów robotycznych. Pożądanymi cechami robotów współdziałającymi z ludźmi są: inteligencja, autonomia, zdolność do komunikacji, wnioskowania i uczenia się. Aby uzyskać te cechy trzeba zaprojektować złożony sterownik robota lub bardziej ogólnie, systemu robotycznego. Jest to zadanie nietrywialne, stąd wiele jest propozycji jego rozwiązania.

Pomimo liczności opracowywanych systemów robotycznych nie istnieje jedna metoda ani przedstawiania ich architektur, ani klasyfikacji ich struktur (np. [33, 37, 47, 49, 52, 96, 97, 114, 150]). W pracach [43, 83], poświęconych architekturom systemów robotycznych, wskazano, że do ich opisu trzeba określić:

— strukturę architektoniczną, czyli prezentację podsystemów i ich połączeń,
— styl architektoniczny, czyli opis wykorzystywanych konceptów obliczeniowych i komunikacyjnych, określających aktywność lub funkcje systemu – zapewne lepiej byłoby to nazwać funkcjami architektury, niż stylem architektonicznym.

Niemniej jednak praca [83] wskazuje, że dla większości stworzonych systemów trudno określić zarówno ich strukturę jak i styl (funkcje). Pomimo prowadzenia prac dotyczących formalnej specyfikacji oprogramowania sterującego robotami [7, 94, 95, 157, 165] i jego formalnej weryfikacji [81, 82] niestety takie ścisłe podejście do procesu wytwarzania sterownika nie uzyskało szerszej akceptacji środowiska robotyków. Zazwyczaj architektury przedstawiane są za pomocą nieformalnych opisów tekstowych oraz diagramów blokowych o bardzo zróżnicowanym poziomie szczegółowości, przy zastosowaniu różnych środków graficznych. Prowadzi to do poważnych problemów w implementacji i dokumentacji tak opisanych systemów, nie wspominając kłopotów przy próbach modyfikacji ich działania.

Najpopularniejszą formą architektoniczną w robotyce jest struktura warstwowa. Podział na warstwy jest albo tworzony na podstawie częstotliwości powtarzania zachowania komponentów systemu [50] albo przez dekompozycję zadania na podzadania. Dominują architektury dwu- [106] i trójwarstwowe, np.: czuj–planuj–działaj SPA (ang. *sense–plan–act*), reaktywna uogólniająca (subsumption) [25–27], hybrydowe planująco–reaktywne [6], hierarchiczne [4, 59]), inspirowane biologicznie [94, 95, 138], wykorzystujące przekonanie–pragnienie–intencję BDI (ang. *belief–desire–intension*) [111, 129].

Jedną z najstarszych architektur jest architektura SPA, używana na przykład przez robota Shakey już w latach sześćdziesiątych XX w. [107]. W tej architekturze moduły: percepcji, planowania i wykonawczy połączone były kaskadowo, co wprowadzało duże opóźnienie w reagowaniu na zmiany zachodzące w środowisku. Model środowiska, niezbędny do planowania działań, zapisywany był w języku STRIPS, o którym więcej napisano w sekcji 2.9.

Ponieważ planowanie jest czasochłonnym zadaniem, R. Brooks zaproponował alternatywną architekturę -– architekturę uogólniającą, w której zachowania zostały zdefiniowane jako: reakcje na bodźce sensoryczne lub producenci aktywności. Jeśli reakcje dolnej warstwy nie były w stanie poradzić sobie z zadaniem, reakcje górnej warstwy hamowały lub tłumiły działania dolnej warstwy, przejmując kontrolę nad systemem, a tym samym uogólniając ich działania. W takim systemie reprezentacja środowiska nie występuje. Brooks twierdził, że najlepszym modelem środowiska jest ono samo, więc wystarczy jedynie odczytywać bodźce z niego pochodzące [27]. Ponieważ przetwarzanie informacji w takich systemach odbywa się równolegle (wszystkie zachowania działają jednocześnie, a ostateczny wynik ich działania jest superpozycją propozycji generowanych przez te zachowania), to reakcje robota są prędkie. W systemach o architekturze SPA dekompozycja na moduły następuje według funkcji, które poszczególne moduły mają realizować w potoku przetwarzania informacji, poczynając od percepcji a kończąc na akcji, natomiast w architekturze uogólniającej dekompozycja dokonywana jest na zachowania, z których każde samodzielnie może realizować pewne proste zadanie – zachowania działają wspólnie i równolegle, by zrealizować zadanie złożone. Architektura uogólniająca może być budowana stopniowo, gdzie każda nowa warstwa nadaje więcej kompetencji działaniom systemu. Niemniej jednak na każdym etapie system jest w stanie działać w środowisku, być może słabo, ale jednak zachowywać się w miarę sensownie. Należy zwrócić uwagę, że w architekturze SPA podsystem percepcyjny działa na wszystkich dostępnych sygnałach pochodzących ze środowiska i na tej podstawie buduje aktualny stan modelu tego środowiska, by w konsekwencji podjąć decyzję, co do dalszych swoich działań. Natomiast w architekturze uogólniającej poszczególne moduły (zachowania) działają tylko na części odczytów czujników, więc agregując je uzyskują częściową reprezentację środowiska. Reprezentacja całości jest rozproszona i, co więcej, jest cechą emergentną takiego systemu. Prawdopodobnie podobnie działają systemy nerwowe u zwierząt, stąd wiara Brooksa, że tym sposobem można wytworzyć inteligencję ssaków, a może i ludzką – niestety nie udało się tego jeszcze osiągnąć. Wszakże ewolucja potrzebowała na to kilkaset milionów lat.

Ponieważ architektura SPA wydaje się być zbyt powolna, a architektura uogólniająca, choć bardzo reaktywna, nie jest dalekowzroczna, pojawiły się architektury hybrydowe. Były one zwykle projektowane jako systemy trójwarstwowe, realizujące: planowanie (warstwa deliberatywna), sekwencjonowanie (zwykle implementowane jako automat skończony przełączający zachowania) i sterowanie urządzeniami (czyli warstwa odpowiedzialna za sterowanie efektorami i agregację danych z receptorów – w sumie tworząc warstwę reaktywną).

Istnieje wiele odmian architektury BDI, różniących się szczegółami. Cechą wspólną jest wyróżnienie stanów umysłu (ang. *mental states*), które są emanacją pracy układu sterującego. W literaturze (np. [83, 111, 129]) przyjmuje się, że są to: 1) przekonania, czyli posiadana informacja, 2) pragnienia, czyli

motywacja do działania i 3) intencje systemu, czyli wybrany plan działania. Tak naprawdę, to te trzy atrybuty razem określają aktualny stan systemu. Przekonania odpowiadają wiedzy, jednak różnią się tym, że wiedza jest z pewnością prawdziwa, podczas gdy przekonania mogą takie nie być. Przekonania obejmują zbiór informacji dotyczących samego agenta i jego środowiska, w tym innych czynników, a także zasad wnioskowania do tworzenia nowych przekonań. Pragnienia reprezentują cele, które agent chce osiągnąć albo poprzez własne działania, albo które mogą być wymuszone przez pewne czynniki zewnętrzne. Cele są pragnieniami, do realizacji których agent aktywnie dąży. Intencje to pragnienia, do których zobowiązał się agent, wybierając i realizując plan, który prowadzi do ich spełnienia. Reaktywne zachowanie agenta jest spowodowane zdarzeniami, zarówno generowanymi zewnętrznie, jak i wewnętrznie. Architektura BDI opiera się na interpreterze, który jest wyzwalany przez zdarzenia. Aktualizuje on przekonania i pragnienia, a w konsekwencji realizuje nowe intencje.

2.6. Agent upostaciowiony

Istotny wpływ na architektury systemów robotycznych miało wprowadzenie pojęcia agenta upostaciowionego. Koncepcja agenta upostaciowionego została spopularyzowana przez Rodneya Brooksa [26], podczas debaty nad problemem, czy roboty potrzebują modeli otoczenia oraz planowania do inteligentnego działania w złożonym środowisku. Tradycjonaliści, zwani zwolennikami GOFAI (ang. *Good Old Fashioned Artificial Intelligence*), uważali, że robot powinien działać w architekturze SPA, a więc wpierw pozyskać informacje ze środowiska, następnie, na podstawie uaktualnionego stanu modelu, zaplanować swoje czynności, by w końcu je wykonać. Brooks twierdził, iż wykonanie takiego cyklu obliczeń trwa na tyle długo, że w dynamicznie zmieniającym się środowisku, w momencie, gdy dochodzi do wykonania zaplanowanych czynności, stan otoczenia jest już na tyle zmieniony, że te czynności są nieadekwatne do sytuacji. Dlatego twierdził, że robota należy traktować jako agenta upostaciowionego, tj. agenta mającego materialne ciało, który powinien być usytuowany, tj. działać bezpośrednio w rzeczywistym środowisku, a nie używać uproszczonego jego modelu do planowania swych działań [24, 27, 28]. Robot powinien być reaktywny, a inteligencja pojawi się w wyniku interakcji agenta z rzeczywistym środowiskiem. Przetwarzanie sygnałów pochodzących ze środowiska wymaga inteligencji, więc ją stymuluje. Innymi słowy, inteligencja jest tu traktowana jako efekt emergentny oddziaływania robota i środowiska kształtującego jego zachowanie.

Ani zwolennicy GOFAI, ani Brooks nie udowodnili, że ich podejścia są jedynie słuszne, dlatego pojawiły się agenty hybrydowe, mające właściwości zarówno reaktywne, jak i deliberatywne [6]. Debata ta pobudziła wielu autorów do skoncentrowania się na koncepcji agenta. Klasyczny tekst o sztucznej inteligencji [122] traktuje agenta jako coś, co działa. Taka wszechogarniająca definicja jest zbyt ogólna, stąd podręcznik ten nadaje agentom określone cechy: autonomię, czyli działanie bez pomocy człowieka, umiejętność postrzegania środowiska, wytrwałość, adaptację do zmian zachodzących w środowisku, zdolność do podejmowania zadań innych agentów. Dodatkowo różnicuje agenty wprowadzając pojęcia:

- agenta racjonalnego, który wytwarza optymalny wynik swoich działań lub optymalny oczekiwany rezultat, jeśli działa w niedeterministycznym środowisku.
- agenta uczącego się, który doskonali swoje możliwości dzięki zdobytemu doświadczeniu.

Agent upostaciowiony wyewolułował z pojęcia agenta programowego używanego w informatyce. Niestety większość tekstów poświęconych agentom (np. [109]) wskazuje, że nie ma ścisłej definicji tego pojęcia. Według Hyacintha Nwany [110] jednym z podstawowych kryteriów klasyfikacji agentów programowych jest to, czy są one stacjonarne czy mobilne, co określane jest na podstawie tego, czy są one umiejscowione na jednym komputerze, czy poruszają się po sieci. Ponadto agenty mogą być: reaktywne, deliberatywne lub hybrydowe. Wyróżnia się szczególną klasę agentów – agenty interfejsujące, które umożliwiają użytkownikom interakcję z różnymi systemami.

Michael Wooldridge [147] definiuje agenta jako system komputerowy ulokowany w określonym środowisku, w którym może działać autonomicznie, aby osiągnąć określony cel. Definicja ta jest podstawą tekstu [111], który wymienia pożądane właściwości agenta. Agent powinien być reaktywny, tj. reagować na zmiany zachodzące w otoczeniu, proaktywny, tj. dążyć do celu, społeczny, tj. zdolny do interakcji z innymi agentami, elastyczny, tj. zdolny do osiągnięcia celu na różne sposoby, odporny, tj. zdolny radzić sobie z ewentualnymi błędami.

Niewątpliwie istnieje wiele podobieństw między agentem programowym i upostaciowionym, ale podstawową różnicą jest to, że ten pierwszy funkcjonuje wewnątrz komputera lub sieci komputerowej, a więc umownie rzecz ujmując w cyberprzestrzeni, natomiast ten drugi ma ciało materialne i funkcjonuje w środowisku rzeczywistym. Należy też zwrócić uwagę, że agent upostaciowiony nie powinien być utożsamiany z robotem, gdyż jednego robota może tworzyć wiele agentów upostaciowionych, ale nie na odwrót [162]. Istotnym jest to, że w przypadku agentów upostaciowionych ich interakcja z otoczeniem ma fundamentalne znaczenie dla powstania i rozwoju inteligencji systemu z nich złożonego.

2.7. Autonomia

Obecnie dąży się do tego, aby roboty realizowały swoje zadania autonomicznie, a więc bez pomocy człowieka. Rozróżnia się dwa ogólne typy autonomii: energetyczną i decyzyjną. Ten pierwszy typ determinuje, czy źródło zasilania robota w energię znajduje się na jego pokładzie czy nie. Ten drugi wskazuje, czy przy realizacji zadania potrzebny jest współudział człowieka. W tym przypadku raczej mówi się o stopniu lub poziomie autonomii, w zależności od wpływu operatora na wykonywanie czynności prostych. Możemy tu mieć do czynienia z urządzeniami teleoperowanymi, gdy człowiek jest odpowiedzialny za sterowanie ruchem urządzenia lub o dużym stopniu autonomii, gdy całe zadanie jest realizowane bez udziału operatora. Zarówno robot chirurgiczny da Vinci [19], jak i łazik marsjański Sojourner [41] są urządzeniami teleoperowanymi. W przypadku urządzeń w pełni autonomicznych, operujących w złożonym i zmiennym środowisku, inteligencja operatora musi zostać zastąpiona tą sztuczną.

Międzynarodowe stowarzyszenie Society of Automotive Engineers (SAE), zajmujące się m.in. standaryzacją, zdefiniowało sześć poziomów autonomii samochodów, numerując je od 0 do 5 [92]. W tym przypadku pojazdy traktowane są jako roboty mobilne. Samochody przypisane poziomowi 0 są pod pełną kontrolą kierowcy i co najwyżej informują go o niebezpieczeństwie. Na poziomie 1 człowiek i układ sterowania współkierują pojazdem, np. kierownicą zawiaduje człowiek, a prędkość regulowana jest przez automat. Na poziomie 2 układ sterowania w pełni kieruje pojazdem, ale człowiek stale powinien czuwać nad sytuacją panującą na drodze i natychmiast przejąć sterowanie, o ile zauważy, że dzieje się coś niepokojącego. Dopiero poziom 3 autonomii zezwala kierowcy odwrócić uwagę od sytuacji na drodze, bo w tym przypadku to samochód jest odpowiedzialny za wykrywanie sytuacji niepokojących i wtedy wzywa kierowcę, by przejął kontrolę nad pojazdem. Ten poziom autonomii umożliwia, przykładowo, automatyczne kierowanie w korku lub na autostradzie. Na poziomie 4, gdy spełnione są odpowiednie warunki drogowe, udział człowieka w kierowaniu nie jest wymagany, więc może się on zająć dowolną inną aktywnością. Samochód może wezwać kierowcę do interwencji, ale jeżeli ten nie zareaguje, to pojazd może zacząć dążyć do zapar-

kowania. Taka sytuacja może powstać, gdy warunki pogodowe zmienią się drastycznie. Pojazdy sklasyfikowane na poziomie 5 nie wymagają uczestnictwa kierowcy w sterowaniu pojazdem niezależnie od warunków drogowych. Wyższe poziomy autonomii obecnie są jedynie przedmiotem badań. IEEE pracuje nad standardem dotyczącym bezpieczeństwa pojazdów autonomicznych (P2846 Draft Standard for Assumptions for Models in Safety-Related Automated Vehicle Behavior). Zarys standardu P2846 określa zbiór założeń dla dających się przewidzieć scenariuszy, które należy brać pod uwagę przy tworzeniu modeli związanych z bezpieczeństwem pojazdów sterowanych automatycznie. Standard jest tworzony, aby wszystkie firmy tworzące autonomiczne samochody, niezależnie od użytych technologii, tak samo rozumiały bezpieczne zachowanie pojazdu na drodze. Z punktu widzenia konsumentów standard tworzy sytuację, w której kupujący samochód nie będzie musiał zastanawiać się, który wybrać z punktu widzenia reguł bezpieczeństwa.

2.8. Ontologie

Cel, który robot ma zrealizować, może być określony w fazie projektowania robota – zazwyczaj tak się dzieje w przypadku robotów specjalizowanych wykonujących złożone zadanie. Zadania mogą też być formułowane i zmieniane w trakcie działania robota. Wtedy układ sterowania robota musi zawierać interpreter poleceń. Zazwyczaj lista poleceń wyrażana jest jako język programowania robota. Jest to szczególnie często stosowana metoda programowania robotów przemysłowych [163]. Interakcja między operatorem i nie w pełni autonomicznym robotem usługowym bądź terenowym odbywa się albo poprzez dedykowaną konsolę (np. [41]) albo głosem w ograniczonym języku naturalnym (np. [164]). W obu przypadkach musi istnieć wewnętrzny język sterownika robota służący reprezentacji poleceń.

Niezależnie od tego, w jaki sposób robotowi zostanie przekazane zadanie do wykonania, istotne są kategorie pojęciowe służące określeniu tego zadania. Te kategorie tworzą ontologię [66]. Złożoność ontologii, z której korzysta robot, determinuje poziom jego inteligencji. Ontologia określa zarówno byty, jak i relacje między nimi. Podstawowym celem jej stosowania jest umożliwienie wnioskowania dotyczącego realizacji zadania, a więc planowanie. Szczegółowość planowania zależna jest od ilości wiedzy zdroworozsądkowej (ang. *commonsense knowledge*), którą dysponuje robot. Zasób wiedzy zdroworozsądkowej posiadany przez robota w istotny sposób determinuje zdolność do prawidłowego zrealizowania zadania. Przykładowo, jeżeli w ontologii robota nie występuje pojęcie pola grawitacyjnego, to pozostawienie przedmiotu w powietrzu, bez podparcia, będzie uznane za rozsądne rozwiązanie, ale w rzeczywistości spowoduje katastrofę.

Wyróżnia się różne poziomy ontologiczne. Dla robotów manipulacyjnych zwykło się definiować cztery poziomy. Na najniższym poziomie zadanie definiuje się określając kolejne położenia lub trajektorie ruchu końcówki manipulatora za pomocą wartości zmiennych złączowych, a więc kątów lub przesunięć między członami łańcucha kinematycznego. Jest to dla operatora dość niedogodny sposób definiowania zadania, gdyż faktyczny cel działania robota nie zostaje wyjawiony – co ma być wykonane, wynika z sekwencji zapisanych położeń złączy. Kolejny poziom dotyczy określenia pozycji końcówki, z którą związano kartezjański układ odniesienia. Definiując jego położenie i orientację wskazuje się pozycje docelowe końcówki. W tym przypadku nadal cel działania nie jest określony jawnie, ale sposób definicji trajektorii jest o wiele bardziej intuicyjny. Ontologie dwóch wspomnianych poziomów definiują zadanie w kategoriach ruchu robota pomiędzy pozycjami zadanymi wyrażonymi w różnych przestrzeniach: konfiguracyjnej (złączy) i operacyjnej (końcówki). Człowiek jednak, nawet jeżeli wydaje mu się dość szczegółowe polecenia, jest przyzwyczajony do definiowania zadania poprzez wskazywanie przedmiotów i określanie związanych z nimi czynności. W ontologii tego poziomu pojawiają się nazwy kategorii obiektów oraz relacji między nimi [154, 155]. Relacje odgrywają fundamentalną rolę, gdyż, przykładowo, postawienie filiżanki na spodku i przemieszczenie spodka, spowoduje zmianę pozycji filiżanki, ale nie na odwrót. Przy sklejeniu dwóch obiektów relacja staje się symetryczna, bo przemieszczenie dowolnego z obiektów przesuwa drugi z nich. Stosując ontologie tego poziomu czynności przestają dotyczyć bezpośrednio robota, a odnoszą się do potencjalnie nieskończonej liczby obiektów znajdujących się w środowisku. Sytuacja staje się bardzo skomplikowana. Jedynie obiekty uchwycone przez robota mogą być przemieszczane. Ponadto obiekty mogą być pozostawione jedynie w miejscu, w którym będą podparte stabilnie, a więc trzeba uwzględnić zarówno grawitację jak i tarcie. Nawet jeżeli uwzględni się pełny model dynamiki środowiska, co oczywiście nie jest możliwe ze względu na brak precyzyjnej znajomości wszystkich jego parametrów, to nadal zadanie formułowane jest w kategoriach szczegółowych przemieszczeń – wprawdzie tym razem obiektów, a nie robota bezpośrednio. Należy zwrócić uwagę, że ludzie komunikując się ze sobą zakładają, że ich interlokutorzy mają zdroworozsądkową wiedzę o zachowaniach obiektów i w związku z tym wystarczy by polecenie odnosiło się jedynie do celu, który ma być osiągnięty. Plan realizacji zadania musi zostać wygenerowany przez osobę, której wydano polecenie. Planowanie wymaga kolejnego poziomu ontologicznego. Najbardziej pożądanym byłoby przekazywanie robotowi poleceń w języku naturalnym. Niestety język naturalny potrafi być wieloznaczny, a ponadto jego rozumienie wymaga od interlokutorów nie tylko posiadania wiedzy zdroworozsądkowej, ale również znajomości kodów kulturowych. Dla porozumienia się ludzi między sobą, a więc i dla komunikowania się z robotami, obie strony dialogu muszą wykorzystywać tę samą ontologię. Ponadto dla naturalnej współpracy z ludźmi roboty same muszą wyrażać emocje i odpowiednio reagować na emocje tych, z którymi wchodzą w interakcje.

Ontologie stanowią podstawę do tworzenia baz wiedzy. Jakość bazy wiedzy ocenia się określając, co można wywnioskować za jej pomocą, na jakie zapytania jest w stanie odpowiedzieć i ile czasu zajmuje znalezienie odpowiedzi. Przykładowo, KnowRob [139] to baza wiedzy przeznaczona do użytku przez roboty. KnowRob był wzorowany na ontologii ogólnego przeznaczenia Cyc [98], która wszakże nie zawierała pojęć związanych z robotyką. KnowRob zawiera pojęcia zaczerpnięte z robotyki i używane do opisu gospodarstw domowych, czyli: zdarzenia, informacje czasowe, akcje, zadania złożone, parametry akcji, obiekty, informacje czasoprzestrzenne, procesy, komponenty robota i opis jego możliwości. Interakcja z bazą KnowRob odbywa się za pomocą zapytań. KnowRob jest zanurzony w programowej strukturze ramowej CRAM (ang. *Cognitive Robot Abstract Machine*) [11]. W skład systemu wchodzą: bazy wiedzy (reprezentacja wiedzy), moduł dysponujący metodami wnioskowania, moduł akwizycji wiedzy (interfejs do podsystemu percepcji oraz sieci WWW), moduł sterowania robotem oraz modułu interakcji z człowiekiem za pomocą mowy. CRAM wykorzystuje procedury przetwarzające dane z podsystemów percepcji oraz sterowania efektorami, aby uzyskać dane, które są przekształcane w symboliczną reprezentację, na której można wykonywać wnioskowanie. Maszyna CRAM wykorzystuje język CPL (ang. *CRAM Plan Language*) do planowania działań robota. Baza wiedzy KnowRob służy do wnioskowania, jakie działania powinien wykonać robot (tryb zapytań) oraz do przechowywania informacji zebranych przez system percepcji (tryb powiedz). Wykorzystuje koncepcję wirtualnych baz wiedzy, które nie przechowują informacji, ale obliczają je na żądanie przy użyciu najbardziej aktualnych danych dostarczanych przez podsystem percepcji lub

przy użyciu innych już dostępnych danych obecnych w bazie wiedzy. KnowRob wykorzystuje zarówno ogólne, jak i specjalne metody wnioskowania. Preferuje szybkość wnioskowania nad jego ogólność. KnowRob korzysta także z OWL (ang. *Web Ontology Language*), który przechowuje formuły zapisane w logice opisowej DL (ang. *Description Logic*) w pliku zapisanym w formacie XML stworzonym do reprezentacji wiedzy w sieci semantycznej (ang. *Semantic Web*). Instrukcje OWL są wewnętrznie reprezentowane w KnowRob jako predykaty języka Prolog, więc można użyć mechanizmów wnioskowania Prologu. Wszakże należy zwrócić uwagę, że Prologowi brakuje mechanizmów wnioskowania zajmujących się niepewnością, właściwościami przestrzennymi czy czasowymi.

Człowiek zlecając wykonanie zadania zakłada, że jego wykonawca posiada wiedzę zdroworozsądkową, więc w poleceniu nie są zawarte oczywistości. Ponieważ roboty wiedzy zdroworozsądkowej nie mają, to muszą ją wywnioskować na podstawie zasobów zawartych w swej bazie wiedzy lub mogą ją pozyskać z sieci. RoboEarth [140, 141] ma być zasobem internetowym dla robotów, tj. World Wide Web dla robotów. Generowanie treści przez użytkowników jest podstawą WWW. RoboEarth chce przenieść to podejście na roboty. Roboty mogą wymieniać informacje bez interwencji człowieka. Platforma RoboEarth jest dostarczana jako zestaw pakietów ROS. Wiedza uzyskana za pomocą RoboEarth jest reprezentowana w kategoriach aksjomatów logicznych. Formalnym językiem przedstawiania wiedzy dla robotów do wykorzystania i wnioskowania jest język SRDL (ang. *Semantic Robot Description Language*), który umożliwia opis: akcji i ich parametrów, pozycji obiektów i modeli potrzebnych do ich rozpoznawania, informacji dodatkowych o danych (np. sposób reprezentacja pozycji, jednostek przypisanych wielkościom liczbowym, formatów plików), wymagań nałożonych na robota w zakresie wykorzystania dostępnych informacji, modeli własnych robota i obiektów oraz metod dopasowywania wymagań zadań do możliwości robota. Modele obiektów zawierają informacje o tym, jak je rozpoznać (rysunki i obrazy CAD) oraz jak je poruszać. Modele rozpoznawania można pobrać z sieci WWW. Jeśli robot nie ma modelu obiektu, którego potrzebuje, w swojej lokalnej bazie wiedzy, to może zażądać jego pozyskania poprzez RoboEarth. Środowiska są reprezentowane jako mapy. Niestety siatki zajętości (ang. *occupancy grid*) nie mogą być używane do wnioskowania, więc w tym celu wykorzystywane są mapy semantyczne. SRDL jest realizowany jako rozszerzenie ontologii KnowRob.

Na marginesie dyskusji na temat roli ontologii w robotyce należy zauważyć, że komunikujące się ze sobą roboty muszą korzystać nie tylko z tego samego języka, ale również z tej samej ontologii [105]. W przeciwnym razie nie będą się mogły porozumieć. Co więcej, często sterowniki robotów konstruowane są jako systemy składające się z agentów. Foundation for Intelligent Physical Agents (FIPA), jedno ze stowarzyszeń standaryzacyjnych IEEE, określiło standard komunikacji międzyagentowej, który zakłada, że wiadomości przesyłane między agentami zawierają informacje o języku komunikacji oraz wykorzystywanej ontologii [115]. Tak więc ontologie są istotne dla sposobu komunikacji wewnątrz sterownika systemu robotycznego oraz sposobu jego komunikowania się ze światem zewnętrznym, a przez to także dla Internetu Rzeczy oraz Przemysłu 4.0.

2.9. Planowanie

Podejmowano próby proceduralnego określenia jednoczesnego działania robota w środowisku rzeczywistym i jego modelu, np. przy implementacji języka TORBOL (ang. *Transformation Of Relations Between Objects Language*) [154]. Interpreter tego języka wybierał właściwy wariant wykonania akcji (dokładniej realizacji relacji między obiektami) na podstawie stanu modelu środowiska. Niestety, gdy możliwych wariantów realizacji zadania jest nadmiernie wiele, a w szczególności nie jesteśmy w stanie przewidzieć, jak wiele, to napisanie wariantowego programu sterującego robotem staje się niemożliwe. Mnogość wariantów wynika z tego, że środowisko, w którym operuje robot może przyjmować bardzo wiele stanów, a co więcej cel wykonania zadania wpływa na wygenerowany plan. Przykładowo, jeżeli zadaniem jest wyjęcie butelki ze skrzynki, to najlepiej złapać ją za szyjkę, a jeżeli chcemy z niej nalać sok, to trzeba ją schwycić z boku. Jeżeli to drugie jest niemożliwe, to trzeba wyjąć butelkę ze skrzynki trzymając ją za szyjkę, odstawić gdzieś i zmienić chwyt na dostosowany do nalewania soku. Przenoszenie otwartej butelki z sokiem może się odbywać jedynie w pionie. Natomiast jej orientacja przestrzenna może być dowolna, gdy butelka jest pusta albo zamknięta. Widać, że nawet takie trywialne zadanie ma multum wariantów. W takich sytuacjach trzeba uciec się do planowania opartego na wnioskowaniu prowadzonym na podstawie zgromadzonych faktów. Fakty te pochodzą z własnej bazy wiedzy robota oraz z jego podsystemu percepcyjnego. Wykorzystanie tego zasobu wiedzy umożliwia wyrażenie zadania w niezwykle zwięzły sposób. Zamiast wielce rozgałęzionego programu mającego instrukcje dla każdego z możliwych wariantów, mamy ogólną, zwięzłą specyfikację zadania, a program wnioskujący, biorąc pod uwagę zgromadzoną wiedzę, musi opracować szczegółowy plan działań dla robota, ale dla tej konkretnej sytuacji.

Jednym z przejawów inteligencji jest zdolność do planowania swych działań. Im horyzont planowania dłuższy, a postawione zadanie trudniejsze lub bardziej abstrakcyjne, tym inteligencja musi być wyższa. W robotyce wyróżnia się trzy formy planowania. Planowanie ścieżki robota określa drogę, po której robot ma się przemieścić. Planowanie trajektorii wskazuje, gdzie na ścieżce robot ma się znaleźć w określonym czasie. Wreszcie planowanie zadań określa czynności (akcje), jakie robot powinien wykonać, by zrealizować zadanie. Ta ostatnia forma planowania jest najtrudniejsza, ale też i najciekawsza. Pierwsze dwie, nazywane zbiorczo planowaniem ruchu [80], sprowadzają się do wyznaczenia matematycznej funkcji określającej drogę od aktualnie zajmowanej pozycji do celu, z ominięciem przeszkód. Zazwyczaj wykorzystuje się do tego metody optymalizacyjne albo interpolację.

Do planowania zadań potrzebny jest opis środowiska (świata) wyrażony w pewnym języku oraz program wnioskujący, który na podstawie tego opisu wygeneruje plan realizujący zadanie. Planowanie zadań odbywa się w przestrzeni stanów, która modeluje świat, w którym działa robot. Owa przestrzeń określa, jakie są możliwe stany środowiska (obiektów znajdujących się w nim), a czasem również stany robota (np. jego położenie, orientacja i prędkość). Do planowania wymagany jest opis stanu początkowego i pożądanego stanu końcowego, a ponadto potrzebny jest algorytm, który znajdzie sekwencję akcji zmieniających stan początkowy w końcowy [88, 122]. Akcje przekształcające stan świata mogą być traktowane jako funkcje przejścia dla stanów tego świata. Dodatkowo mogą być narzucone ograniczenia na stany pośrednie bądź może być sformułowane kryterium optymalności planu, tzn. funkcjonał określający pod jakim względem plan ma być najlepszy.

Najbardziej znanym planerem (i jednocześnie językiem opisu) jest STRIPS [56, 122]. Sposób opisu stanu, który zastosowano w STRIPS, wykorzystywany był przez wiele innych planerów. Celem STRIPS było przekształcenie początkowego stanu świata w końcowy, za pomocą dostępnych akcji. Akcja mogła być wykonana jedynie, gdy były spełnione warunki początkowe (ang. *preconditions*) dla niej, i zmieniała stan świata zgodnie z definiującymi ją efektami zwanymi warunkami końcowymi (ang. *postconditions*). Stany zadania, początkowy i końcowy, były określane za pomocą zbioru warunków, które w tych stanach powinny być spełnione. Stany były opisywane za pomocą rachunku predykatów pierwszego rzędu [122], natomiast robot

operował w tzw. świecie klocków, a więc w wielce uproszczonym modelu środowiska rzeczywistego. Co więcej, środowisko to było deterministyczne, w pełni obserwowalne i nie działały w nim inne agenty prócz robota. Założono zamknięty model świata, tzn. wszystkie warunki podstawowe, które nie zostały explicite wymienione jako prawdziwe uznawano jako fałszywe. W planerze ADL zrezygnowano z tego założenia – wartość logiczna niewymienionych warunków podstawowych traktowana była jako nieznana [122]. Później te ograniczenia zostały zniesione, a model świata stawał się coraz bardziej zbliżony do rzeczywistego.

STRIPS jest warty wspomnienia nie tylko ze względów historycznych, ale przede wszystkim dlatego, że większość późniejszych języków planowania bazowała na tym języku. W szczególności obecnie najbardziej popularna rodzina języków mających w swych nazwach akronim PDDL (ang. *Planning Domain Description Language*) wywodzi się od STRIPS. PDDL [61] stał się rodzajem standardu w dziedzinie planowania zadań. Języki te dzielą zadanie na dwie części: 1) opis dziedziny oraz 2) opis problemu. Opis dziedziny jest rodzajem ontologii określającej hierarchię typów obiektów i obiektów, sparametryzowane akcje, które mogą być realizowane, oraz predykaty określające fakty dotyczące dziedziny. Natomiast opis problemu zawiera obiekty znajdujące się w świecie robota, stan początkowy świata oraz opis stanu końcowego. Planery, które korzystają z opisu zadania w języku PDDL wytwarzają albo w pełni uporządkowane plany TOP (ang. *totally ordered plan*) albo częściowo uporządkowane POP (ang. *partially ordered plan*). W pierwszym z tych przypadków kolejność zaplanowanych akcji jest uporządkowana liniowo i stanowi sekwencję, w drugim, kolejność niektórych czynności nie jest wskazana, w szczególności mogłyby być wykonane równolegle. Późniejsze modyfikacje języka PDDL wprowadzały możliwość występowania zdarzeń, które zostały wywołane przez inne agenty działające w świecie. Ponadto prócz akcji wywołujących dyskretne zmiany stanu wprowadzono takie, które działały z czasem ciągłym. Pojawiły się też probabilistyczne warianty PDDL (PPDDL), które przypisywały wynikom akcji prawdopodobieństwo sukcesu. Język RDDL (ang. *Relational Dynamic influence Diagram Language*) wprowadził do planowania częściową obserwowalność stanu. W języku tym można wyrazić częściowo obserwowalne decyzyjne procesy Markova POMDP (ang. *Partially Observable Markov Decision Processes*). Zastosowanie procesów decyzyjnych Markova do podejmowania decyzji w obliczu niepewności jest trudne ze względu na olbrzymią liczbę możliwych stanów, więc wprowadzane są ograniczenia, które umożliwiają znalezienie planu w skończonym czasie [54]. Powyższe rozszerzenia podstawowego języka PDDL zwiększają skuteczność wygenerowanych planów – innymi słowy, poprawne wykonanie planu w rzeczywistym środowisku staje się coraz bardziej prawdopodobne.

Podstawowymi problemami w planowaniu jest duża liczba możliwości, które pojawiają się zarówno przy zwiększaniu liczby akcji możliwych do zastosowania, jak i wariantów zaistniałych sytuacji, oraz fakt, że pełny stan może nie być obserwowalny dla robota. Próbowano podejścia z wariantowością (ang. *contingency planning*), gdzie generowano plany na wszelkie ewentualności, natomiast w trakcie realizacji planu uzyskiwano dodatkową informację z układu percepcyjnego i na tej podstawie decydowano, który wariant wybrać. Obecnie raczej stosuje się przeplanowywanie. Jeżeli w trakcie wykonania planu zastana sytuacja odbiega od tej oczekiwanej planuje się ponownie uwzględniając nową sytuację. Niestety planowanie jest obliczeniochłonne, więc zajmuje dużo czasu. Oznacza to, że system powoli reaguje na zdarzenia zewnętrzne. Aby tego uniknąć, do planowania stosuje się algorytmy przerywalne w dowolnym momencie (ang. *anytime algorithm*) [72]. One zwracają aktualną, w momencie przerwania, wersję planu. Wersja ta nie będzie doskonała. Gdyby algorytm miał więcej czasu, to by ją ulepszył. Niemniej jednak dłuższe planowanie oznaczałoby, że perfekcyjny plan będzie nieaktualny. Lepszy jest słaby plan na czas, niż najlepszy, ale za późno. Algorytmy tego typu stosowane są do planowania, w miarę zdobywania nowej wiedzy o środowisku [69]. Więcej wiedzy oznacza lepszy plan, ale nowej wiedzy nie można pozyskiwać w nieskończoność.

Ponadto prowadzone są pracę nad planowaniem hierarchicznym, w którym akcje wyższego rzędu mogą być dekomponowane na akcje niższego rzędu [71]. Hierarchiczne sieci zadań HTN (ang. *Hierarchic Task Network*) powodują, że planowanie nie dotyczy jedynie przekształcania stanu świata, ale również dekompozycji akcji złożonych na proste. Stąd określa się to jako planowanie w przestrzeni zadań. Istnieje ponadto technika transformacyjnego tworzenia planów. Polega ona na budowaniu wstępnego planu realizacji zadania z bibliotecznych schematów. Następnie taki plan jest przekształcany i korygowany, aż uzyska ostateczną formę [53, 104].

Gdy określony jest już plan w postaci sekwencji akcji, które prowadzą do realizacji zadania, trzeba jeszcze wyznaczyć trajektorie ruchu robota. Wtedy stosowane jest planowanie trajektorii. Niestety na tym etapie może się okazać, że dla wybranej akcji nie istnieje dopuszczalna trajektoria ruchu, a więc ta akcja jest niemożliwa do wykonania. Wtedy trzeba plan zmodyfikować. Dlatego już w fazie generacji planu weryfikuje się wykonalność wybranych akcji. Ten rodzaj planowania nazywany jest łącznym planowaniem wykonania zadania i ruchu robota TAMP (ang. *task and motion planning*) [76, 151]. Należy zauważyć, że dochodzi tu do połączenia planowania dyskretnego, związanego z wyborem akcji w kolejnych chwilach, z generacją ciągłej w czasie trajektorii ruchu, a więc połączenia planowania dyskretnego i ciągłego.

Gdy już plan zostanie wygenerowany i rozpocznie się jego wykonanie, może się zdarzyć, że robot otrzyma kolejne polecenie, o wyższym priorytecie wykonania. Niestety w tej sytuacji metody znane z informatyki, polegające na odłożeniu stanu procesu na stos i wykonaniu procedury obsługi przerwania, nie mogą być zastosowane [51, 62]. Przykładowo, robot sprząta po posiłku, i w momencie gdy niesie talerz do zmywarki, ktoś dzwoni do drzwi. Aby gość nie odszedł, trzeba przerwać sprzątanie i otworzyć drzwi. Pojawienie się robota u drzwi z talerzem w ręku, uniemożliwi ich otwarcie. Wpierw trzeba odstawić gdzieś talerz. Po obsłużeniu gościa trzeba odzyskać talerz, by kontynuować sprzątanie. Widać tu, że potrzebne są dodatkowe czynności zawieszające realizację poprzedniego zadania, a potem jego kontynuację. Potrzebne jest dodatkowe planowanie. Co więcej, łańcuch przerwań kolejno wszczynanych zadań może być długi. Rozwiązanie tego problemu jest otwartym zagadnieniem badawczym.

Języki typu STRIPS, ADL czy PDDL umożliwiają określenie faktów dotyczących zarówno środowiska jak i zadania, które ma być zrealizowane – służą do reprezentacji wiedzy dla celów wnioskowania. Ta wiedza musi być przetworzona przez program wnioskujący, czasami zwany maszyną wnioskującą lub generatorem planów. Maszyna wnioskująca przetwarza specyfikację wyrażoną w jednym z wymienionych języków na plan [142]. Osobnym problemem jest to, że robot może nie być jedynym agentem działającym w środowisku, a więc wtedy jego system percepcyjny musi być w stanie przetworzyć odczyty z czujników na symboliczną reprezentację stanu świata – nazywa się to problemem kotwiczenia (ang. *anchoring problem*) [42]. Najczęściej po wykryciu takiej niespodziewanej zmiany stanu świata plan musi być uaktualniony.

Metody wnioskowania używane w robotyce niekiedy dzielone są na dwie szerokie kategorie: wnioskowanie logiczne oraz wnioskowanie probabilistyczne [10]. To pierwsze zazwyczaj oparte jest na rachunku predykatów pierwszego rzędu, natomiast to drugie na bayesowskiej teorii prawdopodobieństwa.

Ponieważ do wnioskowania zawsze używa się jakiejś logiki, ściślejszym jest podział na: wnioskowanie oparte na ścisłych faktach i regułach, wnioskowanie probabilistyczne oraz wnioskowanie posybilistyczne. Te dwie ostanie formy wnioskowania są zbiorczo zwane rozumowaniem w warunkach niepewności [74]. W przypadku wnioskowania probabilistycznego mamy do czynienia z wielkościami przyjmującymi pewne wartości z różnym prawdopodobieństwem, które określone jest na podstawie częstości występowania, natomiast gdy wnioskowanie odbywa się na podstawie informacji o nieprecyzyjnym znaczeniu mamy do czynienie z rozumowaniem przybliżonym [74]. Brak precyzji może dotyczyć zarówno faktów (np. nieprecyzyjne pomiary lub nieprecyzyjny opis sytuacji) lub reguł (np. zależności między zmiennymi nie są pewne). Należy zwrócić uwagę, że we wnioskowaniu probabilistycznym zdarzenie, które może wystąpić, zdefiniowane jest precyzyjnie, jedynie jego wystąpienie jest niepewne. Natomiast w rozumowaniu posybilistycznym samo zdarzenie jest określone w sposób nieprecyzyjny. Do tego dochodzi jeszcze możliwa niekompletność informacji niezbędnej do wnioskowania, nie wspominając o możliwości uzyskiwania informacji sprzecznych.

Nawet dla wnioskowania opartego na ścisłych faktach i regułach podstawowym problemem jest to, że implementacja tych form wnioskowania w pełnym zakresie czyni przetwarzanie danych nieefektywnym, więc potrzebne są pewne zawężenia, by wnioskowanie stało się wydajne. Rachunek predykatów pierwszego rzędu jest nierozstrzygalny (ang. *undecidable*), czyli maszyna przetwarzająca napisy nie jest w stanie rozstrzygnąć, czy dowolne napisy przetwarzanego języka są prawdziwe czy nie. Dlatego często korzysta się z języków o ograniczonych możliwościach, takich ja wymienione już STRIPS, ADL czy PDDL. W matematyce istnieje rodzina języków reprezentowana zbiorczą nazwą logik opisowych DL (ang. *Description Logic*). Wprawdzie języki te ograniczają reprezentację świata, ale są rozstrzygalne i obliczeniowo do zaakceptowania. Jeżeli liczba obiektów w rozpatrywanym świecie jest skończona, to można zastąpić logikę pierwszego rzędu logiką zdań, jednak kosztem wprowadzenia olbrzymiej liczby zdań opisujących ten świat. Niemniej jednak logika zdań jest rozstrzygalna, więc w skończonym czasie można dzięki niej rozstrzygnąć rozpatrywany problem.

Jednym z podstawowych problemów przy wnioskowaniu jest zapis zmian zachodzących w świecie przy wykonaniu pojedynczej akcji. Zwane jest to problemem tła (ang. *frame problem*) [122]. Przykładowo, jeżeli robot wyjął z szafki butelkę, to pozostałe przedmioty znajdujące się w tej szafce nie zmieniły swoich położeń – oczywiście przy założeniu, że robot manipulując jednymi przedmiotami nie strąca innych. Język opisujący zmianę stanu na skutek wykonania pojedynczej akcji powinien wymagać jedynie zapisu zmian, które nastąpiły w świecie, a nie ponownego opisu wszystkich relacji, które w tym świecie zachodzą. Na to się nakładają rozbieżności między posiadaną przez robota bazą wiedzy a stanem faktycznym występującym w środowisku, np. wskutek działań innych agentów. Ponadto logiki dwuwartościowe nie uwzględniają stopnia zachodzenia pewnych relacji. Tym zajmuje się logika rozmyta [48, 74], która jest jedną z logik wielowartościowych. Inne logiki tego typu uwzględniają, że fakty mogą być prawdziwe lub nie, ale również nieznane. Powoduje to niemożność stosowania reguły wyłączonego środka.

Wnioskowanie na podstawie wiedzy niepełnej jest możliwe, ale im ta niepewność jest większa, tym trudniej jest to robić jedynie na podstawie reguł logiki opartej na faktach pewnych. By zwiększyć użyteczność wnioskowania, w takich sytuacja lepiej zastosować metody probabilistyczne (np. podejścia bayesowskie, teorię konfirmacji bądź teorię Dempstera-Shafera) lub posybilistyczne (np. wnioskowanie rozmyte).

Prawdopodobieństwo odnosi się tu do tego, czy dany fakt jest prawdziwy. Stosując sieci bayesowskie bądź procesy decyzyjne Markova można wywnioskować prawdopodobieństwo zdarzenia. Sieci bayesowskie powstają przez narysowanie grafu acyklicznego, którego węzły reprezentują zmienne losowe. Łuki skierowane łączą węzły reprezentujące zmienne zależne. Jeżeli wybrana zmienna jest warunkowo zależna od innych zmiennych, to węzły reprezentujące te zmienne stanowią poprzedniki w grafie węzła odpowiadającego wybranej zmiennej. Dzięki takiemu przedstawieniu zależności łatwiej jest operować na wycinkach grafu, by ustalić prawdopodobieństwo wynikowe. Sieci takie mogą być stosowane zarówno do wyjaśniania przyczyn obserwowanych efektów (diagnostyka) jak i określenia prawdopodobieństwa powstania efektów gdy znane są przyczyny (określanie związków przyczynowych). Dla zmiennych zależnych od czasu, opisujących stan procesu, posiadających własność Markowa, powstają dynamiczne sieci bayesowskie. Własność Markowa wymaga, by stan następny w czasie zależał od stanu aktualnego, ale nie od jego poprzedników, czyli stan następny jest warunkowo niezależny od stanów poprzedzających stan aktualny. W takich sieciach wnioskowanie przebiega szybko.

Predykaty opisujące stan przestrzenny środowiska uzupełniane są o zależności czasowe. Przykładowo może być potrzeba, by zaznaczyć, że jedna akcja musi poprzedzać inną lub że dwie sytuacje muszą zachodzić jednocześnie. Taki opis umożliwia algebra interwałowa (ang. *interval algebra*). W tym przypadku planowanie polega na poszukiwaniu takiego usytuowania akcji w czasie, by spełnić narzucone ograniczenia czasowe. Tego typu problemy rozwiązywane są przez sprowadzenie tak postawionego zagadnienia do problemu spełniania ograniczeń CSP (ang. *Constraint Satisfaction Problem*), a następnie stosuje się standardowe algorytmy rozwiązujące takie problemy [122]. Jeżeli wymagane jest ustalenie konkretnego czasu rozpoczęcia i zakończenia poszczególnych akcji do planowania stosuje się harmonogramowanie [29]. Istnieją też formalizmy łączące przedziały czasowe i punkty na osi czasu [73]. Niemniej jednak planowanie z zastosowaniem tak skomplikowanych opisów świata jest zagadnieniem obliczeniowo trudnym.

Ponieważ zazwyczaj stan środowiska jest tylko częściowo obserwowalny, natomiast proces decyzyjny jest sekwencyjny, to planowanie można przedstawić jako częściowo obserwowalny decyzyjny proces Markowa POMDP (ang. *Partially Observable Markov Decision Process*) [75]. Jest to probabilistyczny automat skończony, w którym każdej trójce (stan aktualny, akcja, stan następny) przyporządkowano wartość liczbową nagrody. Z przejściem między stanem aktualnym i następnym, pod wpływem konkretnej akcji, związane jest jego prawdopodobieństwo. Planowanie polega na maksymalizacji sumy nagród wskutek wykonywania sekwencji akcji. Ponieważ stan środowiska nie jest w pełni obserwowalny, to musi być odtwarzany na podstawie obserwacji. Związek miedzy stanem i obserwacją też jest probabilistyczny. Niestety to w istotny sposób komplikuje wnioskowanie, czyniąc go nieefektywnym. Dlatego stosuje się uproszczenia tego modelu [76].

W przypadku stosowania teorii zbiorów rozmytych opis zaistniałych sytuacji jest niepewny [48, 74]. Dana sytuacja zachodzi do pewnego stopnia (np. zmierzona temperatura do pewnego stopnia określana jest jako ciepła, a do pewnego stopnia jako gorąca – może to wynikać z subiektywnych odczuć obserwatora lub eksperta). Dlatego w teorii zbiorów rozmytych przypisuje się stopień przynależności elementu (np. zmierzonej temperatury) do danego zbioru – stopień ten jest zawarty w przedziale od 0 do 1, ale nie wynika on z częstości zjawiska, a więc nie ma charakteru probabilistycznego. Baza reguł łączy rozmyte przesłanki z rozmytymi wnioskami – zazwyczaj konstruowana jest przez ekspertów na podstawie ich intuicji popartej bogatym doświadczeniem, stąd brak precyzji określeń. Najczęściej

wnioskowanie rozmyte składa się z matematycznych reguł dotyczących:
- rozmywania (ang. *fuzzyfication*), czyli określenia stopnia przynależności elementu do poszczególnych zbiorów rozmytych,
- agregacji, czyli wyznaczania na podstawie stopni przynależności elementów należących do różnych uniwersów siły działania (ang. *firing*) poszczególnych reguł znajdujących się w bazie,
- skalowania lub przycinania, by wyznaczyć siłę wniosku,
- akumulacji tak zmodyfikowanych wniosków wynikających z poszczególnych reguł oraz
- wyznaczenia ostatecznej wartości ostrej (ang. *defuzzyfiction*), w sumie tworzących przepis na wyznaczenie ostatecznego wniosku – najczęściej decyzji bądź sterowania.

Pomimo prób integracji różnych podejść do planowania [64] nie istnieje jedna zbiorcza forma opisu uwzględniająca wszystkie powyższe zagadnienia. Ponadto jednym z podstawowych problemów przy wykonaniu planu jest rozstrzygnięcie, czy obserwowany obiekt jest tożsamy z tym występującym w bazie wiedzy, a więc i w planie. W takiej sytuacji tworzona jest reprezentacja stanów odzwierciedlających przekonania (ang. *belief*) robota o stanie świata. Przykładowo, jeżeli robot pozostawił zielony kubek w szafce w kuchni, a teraz widzi zielony kubek na stole w pokoju, to musi rozstrzygnąć, czy oba kubki są tożsame, czy to są dwa różne, acz podobne egzemplarze. W tej sytuacji próbuje się odwoływać do planowania probabilistycznego [16]. Ponadto nie wszystkie obiekty są zawsze widoczne, a jeżeli nawet są w zasięgu wzroku robota, to niekoniecznie są dla niego osiągalne. Kolejnym problemem jest rozróżnienie tego, co jest istotne, od tego, co nie. Nieograniczone gromadzenie wiedzy doprowadzi do przepełnienia pamięci oraz nieefektywnego przeszukiwania ogromnej bazy wiedzy. Stąd powinna być gromadzona wiedza potrzebna jedynie do rozwiązania aktualnego zadania, ale to czyni robota i jego sposób planowania mało ogólnym. Niestety wszystkie te problemy oraz wiele innych, o których nie było tu mowy, oznaczają konieczność prowadzenia dalszych badań, by stworzyć inteligentnego robota potrafiącego rozumować stosując wiedzę symboliczną.

2.10. Uczenie

Prócz wnioskowania dotyczącego realizowanego zadania, roboty powinny doskonalić wykonanie powtarzalnych czynności. Stąd intensywne prace nad uczeniem maszynowym [38]. Historycznie oprogramowanie komputerów wykorzystywało algorytmy, które określały kolejne kroki przetwarzania danych. Zadaniem programisty było stworzenie precyzyjnego algorytmu determinującego, co komputer ma zrobić i w jakiej kolejności. Uczenie maszynowe umożliwia odejście od tego żmudnego sposobu tworzenia oprogramowania. Tutaj algorytm uczący, na podstawie olbrzymiej liczby przykładów, uczy się wykonywania pożądanego zadania przez modyfikowanie wag połączeń w sieci. Tak więc, ustalanie sekwencji czynności komputera zostało zastąpione strojeniem wag. Uczenie maszynowe wykorzystuje sieci, które się samodoskonalą w wykonywaniu postawionego im zadania na podstawie dostarczanych danych. Dane te służą określeniu modelu, a dokładniej związku między danymi a rezultatem. Zazwyczaj można wyróżnić dwa etapy pracy sieci: uczenie i wnioskowanie. W pierwszym na podstawie danych treningowych sieć uczy się wykonywania swego zadania, a drugim realizuje to zadanie. Zasoby potrzebne do uczenia nie są potrzebne w trakcie realizacji zadania.

Wyróżnia się kilka sposobów uczenia: nadzorowane (ang. *supervised*), nienadzorowane (ang. *unsupervised*), częściowo nadzorowane (ang. *semi-supervised*), ze wzmacnianiem (ang. *reinforcement learning*) [152]. Jeżeli dane uczące są zaetykietowane, a więc znane jest, jaki rezultat sieć powinna wytworzyć dla tych danych, to mamy do czynienia z uczeniem nadzorowanym. Jeżeli rezultat nie jest z góry znany, to takie uczenie nazywamy nienadzorowanym. Jeżeli tylko część danych uczących jest zaetykietowana, to mamy do czynienia z uczeniem częściowo nadzorowanym, a więc metodą pośrednią między poprzednio wymienionymi metodami skrajnymi. Niezależnie od zastosowanej metody uczenia sieci neuronowe różnią się architekturami [3, 123]. Ich liczność uniemożliwia dokładne omówienie tego zagadnienia w tym artykule.

Uczenie nadzorowane wykorzystywane jest do klasyfikacji (rozpoznawania obiektów) lub przewidywania wartości funkcji na podstawie jej argumentów – zazwyczaj bardzo wielu. Natomiast uczenie nienadzorowane używane jest do poszukiwania struktury danych wejściowych. Sieć wyodrębnia cechy danych. Na podstawie wartości tych cech dokonuje klasyfikacji danych. W związku z tym uczenie nienadzorowane używane jest do klasteryzacji danych, wykrywania anomalii, asocjacji lub filtrowania. W tym ostatnim przypadku stosuje się autoenkodery, które dokonują wpierw kompresji danych wejściowych, by uzyskać zbiór cech, na podstawie którego odtwarzany jest zbiór danych pierwotnych – oczywiście w tym procesie odtworzenie nigdy nie jest akuratne.

Ponieważ etykietowanie jest zarówno czasochłonne, jak i kosztowne, a co więcej nie zawsze znany jest pożądany rezultat działania sieci, metody uczenia nienadzorowanego są często stosowane. By poprawić ich działanie i jednocześnie nie ponosić zbyt wielkich kosztów procesu uczenia, stosuje się uczenie częściowo nadzorowane. Przykładem sieci, która wykorzystuje tę formę uczenia jest sieć GAN (ang. *Generative Adversarial Network*), w której generator produkuje nowe dane przypominające dane oryginalne, natomiast dyskryminator próbuje ocenić, czy ma do czynienia z danymi oryginalnymi czy wygenerowanymi. W każdej iteracji dzięki dodatniemu sprzężeniu zwrotnemu sieci generatora i dyskryminatora się doskonalą w wykonywanych przez siebie zadaniach. Dzięki temu sieci GAN mogą być wykorzystywane do detekcji anomalii [148].

W przypadku sieci neuronowych uczenie nadzorowane prowadzone jest przez wzmacnianie wag połączeń proporcjonalnie do różnicy między pożądaną aktywacją i aktualną. Różnica ta zwana jest błędem. Dokonuje się tego na podstawie iteracyjnej optymalizacji funkcji celu zbudowanej na bazie wspomnianych różnic. Zazwyczaj stosuje się wsteczną propagację błędu (ang. *backpropagation*) zaczynając od warstwy wyjściowej sieci, a kończąc na warstwie wejściowej. Natomiast uczenie nienadzorowane wykorzystuje regułę Hebba [68], która mówi, iż waga połączenia neuronów, których działanie jest skorelowane, powinna ulegać wzrostowi, jeżeli zachodzi relacja przyczynowo-skutkowa między aktywnością tych neuronów. Oczywiście brak takiej korelacji prowadzi do osłabienia połączenia. Uczenie nienadzorowane działa na podstawie przyjętej metryki podobieństwa danych.

Uczenie ze wzmacnianiem, gdzie dane uczące nie są zaetykietowane, wykorzystuje znajomość nagrody związanej z wykonaniem czynności na podstawie tych danych wejściowych. Algorytm uczenia dąży do maksymalizacji skumulowanych nagród. Takie algorytmy są często używane w robotyce do uczenia się podejmowania decyzji w miarę eksploracji środowiska, np. nawigacji w nieznanym otoczeniu. Uczenie ze wzmacnianiem uczy sieć przewidywać kolejny krok działania w taki sposób, by końcowa nagroda była najwyższa. W trakcie działania robot uczy się na podstawie przeszłych swoich działań oraz eksperymentuje z nowymi strategiami, dzięki czemu może się doskonalić.

Rozwój badań nad sieciami neuronowymi miał swoje wzloty i upadki. Od mniej więcej 2012 r. badania te znów przeżywają rozkwit. Spowodowane jest to opanowaniem technologii zrównoleglania obliczeń. Odkąd pojawiły się procesory graficzne GPU (ang. *Graphic Processing Unit*) prowadzenie dostatecznie

szybko obliczeń, które w przypadku sieci neuronowych daje się łatwo zrównoleglić, umożliwiło tworzenie ogromnych sieci nazywanych głębokimi (ang. *Deep Learning Neural Networks*). Sieci te uczą się na milionach danych, dzięki czemu są w stanie z dostateczną pewnością realizować postawione przed nimi zadania. Typowym przykładem jest sieć AlphaGo, która rozgrywając partie go ze swoją repliką, nauczyła się grać tak dobrze, że była w stanie pokonać arcymistrza [133].

Dotychczas głównym polem zastosowania uczenia maszynowego w robotyce była percepcja, a w szczególności rozpoznawanie obrazów oraz sygnałów mowy [79], ale powoli się to zmienia. Obecnie pracuje się także nad samodoskonalącymi się robotami, które dostosowują swoje zachowania do środowiska, w którym działają. Firma Google stworzyła Brain Team [2], zespół którego zadaniem jest zbadanie wpływu uczenia maszynowego na robotykę oraz robotyki na uczenie maszynowe. W tym celu wyposażono duże laboratorium w tzw. farmę robotów. Roboty te uczyły się chwytania przedmiotów i przekazywały sobie nawzajem wiedzę, którą zdobyły na temat przemieszczania i chwytania obiektów widzianych przez siebie [91]. Każdy z robotów wykorzystywał jedną kamerę umieszczoną na statywie obok manipulatora. Konwolucyjna sieć neuronowa uczyła się chwytania obiektów. Założono, że układ oko-ręka nie będzie kalibrowany, więc dostosowanie ruchów do uzyskiwanego obrazu także podlegało uczeniu. Uczenie prowadzono w ciągu dwóch miesięcy wykorzystując od 6 do 14 robotów, dzięki czemu dane uczące składały się z 800 tysięcy prób chwycenia obiektów. W drugim eksperymencie wykorzystano 9 robotów innego typu, które wykonały 900 tysięcy prób, ale im przekazano wiedzę zdobytą w pierwszym eksperymencie. Okazało się, że transfer wiedzy wspomaga uczenie, ale nadal potrzeba olbrzymiej ilości dodatkowych danych. Wprawdzie w obu eksperymentach wykorzystywano różne manipulatory, ale w obu przypadkach miały one po 7 stopni swobody oraz chwytaki dwupalczaste. Obiekty, których różnych typów użyto 1100, wyjmowane były z pojemnika, w którym zgromadzone były w sposób nieuporządkowany, tak że częściowo się przesłaniały, a co więcej ich chwytanie wymagało przesuwania innych przedmiotów. Wykorzystano uczenie bez nadzoru. To, czy próba uchwycenia się powiodła, ustalane było na podstawie stopnia zamknięcia chwytaka oraz różnicy w wyglądzie obrazu pojemnika po uchwyceniu przedmiotu i ponownym jego wrzuceniu przez robota do pojemnika. Początkowo robot nie miał wiedzy na temat obiektów, które będzie widział. Przeprowadzone eksperymenty wskazują, że wiedza zdobyta przez robota jednego typu w umiarkowany sposób wspomaga uczenie podobnego robota, a co więcej ilość danych, które są potrzebne do uczenia chwytania bez nadzoru musi być ogromna. Zebranie tych danych jest procesem niezmiernie czasochłonnym. Niemniej jednak ten niealgorytmiczny sposób podejścia do chwytania wykazał, że roboty nauczyły się różnych strategii chwytania obiektów sztywnych i miękkich. Obiekty sztywne były chwytane poprzez ustawienie palców chwytaka po przeciwnych stronach obiektu, natomiast w przypadku obiektów miękkich chwytak był wgniatany w obiekt i chwytał go przez uszczypnięcie.

W sztucznej inteligencji istnieją dwa podstawowe nurty. Pierwszy bazujący na wnioskowaniu symbolicznym i drugi opierający się na sztucznych sieciach neuronowych. Ten drugi, obecnie jest niezmiernie popularny, ponieważ wykazał swoją wyższość przy zastosowaniu do zadań, z którymi ten pierwszy miał istotne kłopoty, np. rozpoznawanie obrazów bądź mowy. Niemniej jednak należy zwrócić uwagę, że sieci neuronowe, a w szczególności głębokie sieci neuronowe, mają istotne wady. Szerzej rzecz ujmując, sztuczne sieci neuronowe doskonale się sprawdzają w klasyfikacji, gdzie dominuje porównanie ze wzorcem, natomiast dużo gorzej im idzie, gdy muszą znaleźć korelacje między nieprzystającymi do siebie zdarzeniami, np. w diagnostyce uszkodzeń. Ponadto wytrenowana sieć rozwiązuje jedynie zadanie, którego ją nauczono. Zmiana zadania zazwyczaj wymaga ponownego uczenia od początku, a często również zmiany struktury sieci. Koszt trenowania jest wysoki, ponieważ do uczenia potrzeba olbrzymich ilości różnorodnych danych przykładowych, a co więcej przy uczeniu nadzorowanym muszą to być dane zaetykietowane, a proces etykietowania jest dość pracochłonny. Próby douczania takich sieci dla nowych danych często kończą się pogorszeniem wykonywania zadania dla poprzednich kategorii danych. Uczenie głębokie (ang. *deep learning*) niewątpliwie odniosło wielki sukces w wielu dziedzinach. Niemniej jednak uważa się, że w zastosowaniach krytycznych jego użycie jest ryzykowne. Podstawowym problemem jest brak wytłumaczenia (ang. *explainability*), dlaczego sieć daje takie rezultaty, jakie daje. W tej sytuacji nasza pewność, co do działania systemu, przestaje być oparta na rozumieniu, a zaczyna wynikać z wiary. Co więcej, w trakcie douczania sieci wykonywania nowego zadania może ją dotknąć gwałtowana amnezja – zapomni tego, czego się nauczyła wcześniej [119]. Próbą rozwiązania tego problemu jest stworzenie wyjaśnialnej sztucznej inteligencji (ang. *explainable AI*).

Wymienione problemy są obecnie przedmiotem intensywnych badań. Rozwiązanie tych problemów upatruje się w ściślejszym odwzorowywaniu sposobu działania mózgu niż to ma miejsce w sztucznych sieciach neuronowych powszechnie stosowanych. Jednym z podejść jest zastosowanie sieci typu ART (ang. *adaptive resonance theory*) [35, 65], które kategoryzują informacje jednocześnie na podstawie oczekiwanego rezultatu obserwacji oraz danych sensorycznych, tak jak to się dzieje w ludzkim umyśle, który stara się dopasować to co widzimy lub słyszymy do naszych oczekiwań. Sieci ART umożliwiają zachowanie odpowiedniej równowagi między zdobywaniem nowej wiedzy i stabilnością starej.

2.11. Algorytmy ewolucyjne

Algorytmy ewolucyjne dążą do wyboru najlepszego rozwiązania postawionego zagadnienia naśladując procesy ewolucyjne, stąd i stosowana terminologia zaczerpnięta jest z tej używanej do opisu doboru naturalnego [5]. Algorytmy ewolucyjne można uznać za jedną z technik optymalizacji albo za metodę stopniowego doskonalenia rozwiązania. Każde rozwiązanie zagadnienia stanowi osobnika. Ów osobnik reprezentowany jest chromozomem składającym się z genów. Algorytm ewolucyjny działa na populacji osobników. Dla każdego osobnika można obliczyć stopień jego przystosowania do środowiska. Służy temu funkcja przystosowania, która dobierana jest do rozpatrywanego zagadnienia. Algorytm ewolucyjny z pokolenia na pokolenie dokonuje reprodukcji populacji, czyli wybiera osobniki, które będą stanowiły podstawę konstrukcji kolejnego pokolenia osobnikó. Następnie wykonuje operacje genetyczne na zreprodukowanej populacji: mutacje, czyli modyfikację pojedynczych genów w chromosomie, i krzyżowanie, czyli sklejanie części chromozomów uzyskanych z różnych osobników. Dalej dokonuje oceny osobników tak zmodyfikowanej populacji, po czym następuje sukcesja. Ocena osobników i sukcesja służą selekcji. Selekcja jest prowadzona tak, by osobniki lepiej przystosowane przechodziły do kolejnego pokolenia z wyższym prawdopodobieństwem niż te gorzej przystosowane. Niemniej jednak i te ostatnie mogą być reprodukowane, co umożliwia wyrwanie się rozwiązania z ekstremum lokalnego. To powoduje, że algorytmy genetyczne uznawane są za metody optymalizacji globalnej. Po etapie selekcji powyższe kroki są powtarzane znów od reprodukcji, aż spełnione zostanie kryterium stopu, kończące wykonanie algorytmu. Najbardziej wartościowy osobnik uznawany jest za rozwiązanie postawionego zagadnienia.

Ponieważ w naturze osobniki silniejsze lub inteligentniejsze są lepiej dostosowane do życia w środowisku, a to ulega ciągłym zmianom, to mówi się tu o adaptacji. Zdolność do

adaptacji jest jednym z przejawów inteligencji. Algorytmy ewolucyjne były stosowane do rozwiązywania bardzo różnych problemów, w szczególności wyłoniło się programowanie genetyczne, które stosowane było do ewolucyjnego tworzenia programów zamiast ręcznego ich pisania [85, 137]. Algorytm ewolucyjny rozpoczynał pracę od populacji osobników (programów) słabo rozwiązujących postawiony problem, a następnie sprawdzał, jak osobniki kolejnych populacji dają sobie z tym problemem radę. Swego czasu pokładano w tym sposobie programowania wielkie nadzieje i prorokowano zmierzch klasycznej informatyki. Tak się jednak nie stało – nadal programy pisane są przez programistów.

2.12. Inteligencja roju – systemy złożone

Inteligencja roju jest cechą emergentną, czyli wyłaniającą się na skutek samoorganizowania, współdziałania i lokalnego pośredniego bądź bezpośredniego komunikowania się wielu prostych autonomicznych agentów wchodzących w skład zdecentralizowanych systemów [12, 13]. Samoorganizacja jest procesem, w którym wewnętrzna organizacja systemu złożonego z autonomicznych agentów wchodzących w interakcje ze swym otoczeniem samoistnie wzrasta, bez udziału koordynatora. Istotą jest kolektywne działanie prostych jednostek [113]. Symulacyjne badania nad tego typu systemami początkowo były prowadzone pod szyldem sztucznego życia (ang. *artificial life*). Sam termin „sztuczne życie" został spopularyzowany przez Christophera Langtona [86] i odnosi się do życia stworzonego przez człowieka, a nie przez naturę [87]. Wprawdzie prace nad tą dziedziną dotyczyły również związków chemicznych, ale większość badań poświęconych było aspektom matematycznym i symulacjom komputerowym wyimaginowanych systemów składających się z prostych automatów. W szczególności usiłowano odtworzyć systemy biologiczne. Do chwili obecnej nie udało się stworzyć niczego rzeczywiście żywego w sensie biologicznym. Nie wiadomo, czy to dobrze czy źle, ale sam Langton miał wątpliwości, czy nie stworzy to nowych niebezpieczeństw.

W 1970 r. John Conway opracował grę symulacyjną nazwaną Life [58], w której na dwuwymiarowej nieskończonej płaszczyźnie podzielonej na kwadratowe komórki umieszcza się pewną liczbę komórek żywych. Komórki tworzą rodzaj szachownicy, a więc każda z nich ma osiem sąsiadek. Komórka z dwoma lub trzema bezpośrednimi sąsiadkami pozostaje żywa, komórka martwa z trzema żywymi sąsiadkami staje się żywa (reprodukcja), natomiast wszystkie pozostałe komórki żywe umierają (wskutek przeludnienia). Okazało się, że przy tych trywialnych regułach rządzących zachowaniem komórek, zależnie od stanu początkowego, można obserwować wielką różnorodność zachowań organizmów złożonych z komórek żywych – mogą one się rozrastać, oscylować, ginąć lub przemieszczać w różnych formacjach. Komórki stanowią niezwykle prosty model automatu – automatu komórkowego.

Prace nad samopowielającymi się automatami zostały jednak zainicjowane wcześniej, bo już pod koniec lat 40. ubiegłego wieku, przez Stanisława Ulama oraz Johna von Nemanna. Von Nemann zauważył, że jeżeli jeden automat ma stworzyć drugi, to ten pierwszy powinien być bardziej skomplikowany niż ten drugi, ponieważ powinien dysponować dodatkowymi środkami na tworzenie automatów [146]. Niemniej jednak natura tak nie działa. W toku ewolucji organizmy prostsze reprodukują się w taki sposób, że w konsekwencji powstają organizmy coraz bardziej skomplikowane. Rozwiązaniem tego problemu było odwołanie się do Maszyny Turinga [143], która na podstawie prostych inskrypcji jest w stanie tworzyć inskrypcje bardziej złożone. Wprawdzie z matematycznego punktu widzenia te prace są niezmiernie interesujące, ale dotychczas nie dały odpowiedzi na pytanie, jak skonstruować fizyczny automat samopowielający się, a w szczególności jak stworzyć takiego inteligentnego robota.

Inspirację do badań nad inteligencją rojów, pomijając kwestię reprodukcji, stanowi obserwacja owadów społecznych (ang. *social insects*), np. mrówek [63], termitów czy pszczół, które dysponują bardzo prostymi układami nerwowymi, ale zdolne są do współpracy, a w szczególności do tworzenia bardzo złożonych struktur, takich jak mrowiska, termitiery czy gniazda. Przykładowo, termitiery powstające na sawannach, gdzie latem jest niezmiernie gorąco, są tak skonstruowane, że ich wentylacja grawitacyjna zapewnia temperaturę i wilgotność wewnątrz kopca zapobiegającą usmażeniu się owadów zamieszkujących termitierę. Żaden z termitów nie dysponuje całościowym planem budowy kopca, niemniej jednak powstaje struktura spełniająca globalne oczekiwanie. Co więcej, termitiery są cały czas modyfikowane (poprawiane), by mikroklimat w nich panujący był odpowiedni. Należy zwrócić uwagę, że większość owadów społecznych porozumiewa się pośrednio przez stygmergię, czyli pozostawianie innym insektom znaków w środowisku (np. feromonów) [20]. Budowa termitiery stymulowana jest sygnałami pozyskiwanymi ze środowiska, stygmergią i oczywiście imperatywem wewnętrznym do wykonywania tej pracy.

Zainteresowanie rojami powodowane jest ich zaletami, które chcielibyśmy umieć odtworzyć w systemach technicznych. Zalety te są następujące:
- każdy owad wykazuje proste zachowanie, co wymaga jedynie prostego układu sterowania, ale rój wytwarza złożone struktury bądź zachowania,
- każdy owad ma ograniczone możliwości wpływania na środowisko,
- każdy owad ma ograniczone możliwości percepcyjne i poznawcze (kognitywne),
- łatwo się adaptują do zmian zachodzących w środowisku,
- system jest odporny – eliminacja lub awaria niektórych jednostek nie naraża na fiasko funkcjonowania całego systemu.

Niestety mają one również poważne wady:
- rezultat działania systemu jest emergentny, a więc jest trudny do przewidzenia lub do zaprogramowania,
- wynik bazuje nie tylko na zachowaniach jednostki, ale także na interakcjach z innymi osobnikami i środowskiem, przez co jest trudny do przewidzenia,
- brak wiedzy o globalnym celu może prowadzić do stagnacji lub zakleszczenia.

Z punktu widzenia realizacji technicznej systemy o strukturze roju mają następujące cechy:
- nie wymagają centralnego planowania – nie potrzebują koordynatora,
- mają rozproszoną strukturę,
- jednostki komunikują się przez środowisko (stygmergia) – komunikacja bezpośrednia rodzi problem ze skalowalnością, bo przy wielu robotach trudno jest zorganizować komunikację wielu-do-wielu,
- ma proste reguły zachowań indywidualnych,
- wytwarza złożone zachowanie adaptacyjne systemu,
- są skalowalne – struktura układu sterowania każdego robota jest taka sama, niezależnie od liczby robotów,
- są elastyczne – roboty mogą być dodawane lub zabierane ze środowiska bez konieczności modyfikacji specyfikacji zadania,
- są odporne – nie tylko ze względu na redundancję jednostek, ale także ze względu na minimalistyczny projekt jednostki.

Gdyby udało się określić sposób konstrukcji prostych robotów, które poprzez interakcje z otoczeniem oraz wzajemną komunikację byłyby w stanie tworzyć pożyteczne struktury złożone, to byłby to bardzo atrakcyjny sposób tworzenia np. budowli – same mogłyby zespalać się w celu utworzenia np. mostu nad rzeką. Ciekawą próbą, w tym kierunku, było

zaprojektowanie ponad tysiąca robotów, które przemieszczając się po płaszczyźnie stale stykając się krawędziami były w stanie tworzyć konkretne kształty dwuwymiarowe [121]. Niestety, mimo licznych prac na ten temat, jedynie udało się opracować bardzo szczególne systemy zdolne do bardzo ograniczonych zachowań realizujących globalne oczekiwania, np. wspólne pchanie pudła [169], tworzenie formacji, wspólne patrolowanie lub agregowanie się w zespoły robotów w celu realizacji konkretnego celu. Tworzenie takich systemów zazwyczaj sprowadza się do konstrukcji wielu prostych robotów, a następnie weryfikowania, czy uda im się osiągnąć globalny cel. Jeżeli nie, to roboty są modyfikowane, a następnie powtarzany jest krok weryfikacyjny. Innymi słowy, takie systemy zazwyczaj tworzone są metodą prób i błędów.

Często w przypadku zadań złożonych i heterogenicznym składzie grupy wprowadzany jest element rozdzielający zadania elementarne między poszczególne roboty. Taki system zawierający roboty obserwacyjne, manipulujące i przemieszczające został stworzony w ramach projektu europejskiego Swarmanoid [46]. Aby wykonać zadanie polegające na przyniesieniu książki znajdującej się na półce w regale, roboty obserwacyjne musiały latając po pomieszczeniach znaleźć regał z książką, następnie dwa roboty przemieszczające musiały zanieść robota manipulującego do regału, by ten wspiął się na niego i zdjął książkę. Każdy z robotów miał niewielki zasięg swojej komunikacji radiowej, więc dodatkowo roboty musiały sobie stworzyć sieć komunikacyjną. Takie zadanie wymaga zaplanowanie jego wykonania i przydzielenie zadań elementarnych poszczególnym robotom. Przy rozdziale zadań do wykonania, można mieć wątpliwości, czy nadal mamy do czynienia z rojem. Nie zmienia to faktu, że mamy tu do czynienia z dużą grupą współpracujących robotów przy realizacji złożonego zadania, ale jest ono zadane z góry, a nie wyłania się samoistnie z interakcji między robotami dysponującymi prostymi algorytmami postępowania.

2.13. Elektroniczne interfejsy z mózgiem

Interfejsy te konstruowane są ze względów terapeutycznych, ale także poznawczych. W tym drugim przypadku przede wszystkim chodzi o odkrycie tajników działania mózgu, ale niektórzy spodziewają się, że dzięki tym badaniom będzie można usprawnić działanie mózgu. Inni obawiają się, że doprowadzi to do transhumanizmu.

Od lat 60. ubiegłego wieku badane są elektroniczne interfejsy z mózgiem BMI (ang. *Brain Machine Interfaces*). Celem takiego interfejsu jest wytworzenie wzajemnego powiązania między mózgiem zwierzęcia lub człowieka oraz napędem manipulatora, sztucznej nogi lub obrazu na ekranie komputera, by w pętli sprzężenia zwrotnego wykorzystać wolicjonalną aktywność elektryczną mózgu do sterowania ruchami tego napędu [89]. Zwierzę lub człowiek korzysta zarówno z własnych receptorów, np. zmysłu wzroku, jak i stymulacji bezpośredniej poprzez elektrody wytwarzające wrażenie, że np. dotyka twardego lub miękkiego przedmiotu. W niektórych eksperymentach zamiast sterować jakimś urządzeniem technicznym pacjent może sterować własna kończyną, z którą wskutek wypadku utracił łączność przez własny obwodowy układ nerwowy. Intensywność tych badań stale zwiększa się od lat 90. XX wieku. Celem tych badań jest:

— ustalenie i wykorzystanie zasad działania i właściwości plastycznych mózgu oraz
— tworzenie nowych terapii przywracających mobilność i czucie pacjentom z ciężkimi niepełnosprawnościami.

Z punktu widzenia poznawczego głównym celem jest zbadanie właściwości fizjologicznych mózgu, w tym zdolność grup neuronów w korze czuciowo-ruchowej do kodowania informacji i wykazywania plastyczności podczas uczenia się zwierząt nowych zadań motorycznych. Niemniej jednak cel terapeutyczny był pierwotny. Z powodzeniem wszczepia się obecnie implanty ślimakowe połączone z mikrofonem, by przywrócić słuch. W celu przywrócenia wzroku, zbadano pozytywnie, czy implanty umieszczane w korze wzrokowej są w stanie przywrócić zdolność do wykrywania światła. Stosowano również substytucję jednego zmysłu drugim. Elektrody umieszczone na plecach pacjenta były pobudzane przetworzonym obrazem z kamery, dzięki czemu pacjenci po 10 godzinach uczenia się korzystania z urządzenia byli w stanie poprzez czucie dotyku wykrywać obiekty znajdujące się w pobliżu. Istotny postęp w miniaturyzacji implantów pozwolił zwiększyć rozdzielczość implantów umożliwiając jednoczesne pobieranie i wysyłanie sygnałów z/do wielu neuronów. Dzięki temu zwierzęta były w stanie jedynie za pomocą aktywności swoich mózgów włączać urządzenia, a w niektórych przypadkach poruszać np. ramieniem robota. Na pacjentach, którzy musieli być poddani operacyjnemu leczeniu choroby Parkinsona, wykazano, że te same techniki, które były stosowane na zwierzętach, można stosować na ludziach, by uzyskać wzorce ruchowe na podstawie sygnałów podkorowych. Niestety istnieją problemy z biokompatybilnością wielokanałowych implantów mózgowych. Niemniej jednak wykazano, że za pomocą takich implantów można ustalić wzorce ruchowe i je później odtworzyć, co umożliwia sterowanie urządzeniami za pomocą myśli. Co więcej można też pobudzać odpowiednie rejony mózgu, by wytworzyć wrażenia haptyczne.

Prócz technik inwazyjnych stosuje się też interfejsy nieinwazyjne wykorzystujące sygnały EEG. W ten sposób, np. za pomocą otwierania i zamykania oczu można sterować prostym urządzeniem. Ta technika była też stosowana do sterowania wózkami inwalidzkimi oraz egzoszkieletami montowanymi na nogach. Interfejsy nieinwazyjne nie wymagały stosowania procedur chirurgicznych i nie są związane z ryzykiem zakażenia lub tworzenia się blizn wokół elektrod umieszczonych w mózgu. Niestety ich rozdzielczość jest niska, a jakość uzyskiwanych sygnałów niezbyt wysoka. Niemniej jednak techniki nieinwazyjne są preferowane, bo nie zagrażają pacjentowi. Istnieje też technika pośrednia, która wprawdzie wymaga otwarcia czaszki, ale elektrody umieszczane są na powierzchni mózgu, a nie są wprowadzane do niego. Istnieją też interfejsy oparte na technikach obrazowania wykorzystujących np.:

— funkcjonalny MRI, gdzie mierzy się odpowiedzi chemodynamiczne mózgu,
— magnetoencefalografię, która mierzy pola magnetyczne indukowane przez elektryczną aktywność neuronów,
— pomiar wykorzystujący promieniowanie świetlne w bliskiej podczerwieni, by mierzyć koncentrację we krwi oksyhemoglobiny i deokshemoglobiny.

Niestety te metody pomiarowe wymagają specjalnych stanowisk laboratoryjnych.

2.14. Koordynacja ręka-oko

Ponieważ roboty muszą działać w środowisku naturalnym istotna jest prędkość reakcji na bodźce pochodzące z otoczenia. Podobnie, jak u zwierząt wyższych, w robotach kluczową rolę odgrywa koordynacja kończyn ze wzrokiem. W tym celu konstruuje się serwomechanizmy wizyjne [36, 135]. Największym problemem jest szybkie rozpoznanie obrazu – detekcja i lokalizacja pożądanego obiektu. Układ składający się z kamery, procesora i oprogramowania przetwarzającego obraz zazwyczaj działa zbyt powoli, by reagować na szybko poruszające się przedmioty. Tu postępy spodziewane są raczej dzięki ulepszonym urządzeniom przetwarzającym obrazy, a nie doskonalszym algorytmom. Ostatnio opracowana fotoniczna głęboka sieć neuronowa może analizować obrazy bez potrzeby stosowania modułów pamięci, przetwornika analogowo-cyfrowego oraz zegara taktującego, czego wymagają procesory elektroniczne [8]. Może rozpoznać obraz w mniej niż 570 pikosekund. Roz-

miar układu fotonicznego to 9,3 mm^2. Klasyfikację liter p i d w zbiorze 216 liter sieć wykonała z dokładnością 93,8 %, natomiast liter a, t, p i d w zbiorze 432 liter dokonała z dokładnością 89,8 %. Konwencjonalna 190-neuronowa sieć dokonywała tego z dokładnością 96 % – oczywiście zabierało jej to dużo więcej czasu. Ta fotoniczna sieć musi zostać udoskonalona, by radzić sobie z bardziej skomplikowanymi obrazami, z jakimi mają do czynienia roboty.

2.15. Cyberbezpieczeństwo

Wizja Przemysłu 4.0 zakłada wykorzystanie Internetu Rzeczy (IoT), inteligentnych urządzeń oraz chmur obliczeniowych do produkcji wyrobów dostosowanych do indywidualnych potrzeb klienta. Ta wizja rozszerza się w naturalny sposób także na świadczenie usług zarówno publicznych jak i przez urządzenia zainstalowane w domach. Chmura obliczeniowa może wspomagać roboty w przetwarzaniu i gromadzeniu informacji, szczególnie jeżeli ilość informacji oraz złożoność algorytmów przetwarzania danych jest duża. Pomysł zintegrowania robotów z siecią IoT doprowadził do powstania koncepcji Internetu rzeczy robotycznych IoRT (ang. *Internet-of-Robotic-Things*) [134]. Inkorporacja robotów do Internetu rzeczy zwiększa możliwości ich współdziałania z innymi urządzeniami znajdującymi się w ich otoczeniu, poprzez zrozumienie, jakie są ich cele, oraz wykorzystanie czujników znajdujących się w środowisku, a przyłączonych do sieci. Z punktu widzenia IoT dodanie robotów umożliwia wykorzystanie aktywnego czucia, czyli wykorzystanie ruchomości robota do zebrania lepszych odczytów czujników. Roboty, urządzenia produkcyjne i produkty mają komunikować się przez sieć, dzięki czemu następuje efektywniejsze wykorzystanie zasobów produkcyjnych do zindywidualizowanego wytwarzania. Ponieważ u podłoża tej koncepcji tkwi wymiana informacji między rozproszonymi urządzeniami poprzez sieć, niezbędne jest zapewnienie odpowiedniego poziomu cyberbezpieczeństwa. Istotnym elementem tej wizji jest zastosowanie sztucznej inteligencji. Między innymi zakłada się samodoskonalenie działania systemu wytwarzania lub świadczącego usługi dzięki jego uczeniu się. Inkorporowanie robotów do sieci, prócz zalet, niestety ma też wady – naraża je na różne formy ataków cybernetycznych.

Przyczynami podatności systemów robotycznych na cyber-ataki mogą być [149]: połączenia z sieciami przewodowymi lub bezprzewodowymi, korzystanie z platform programowych podatnych na ataki (np. systemów operacyjnych zawierających dziury lub tylne wejścia (ang. *backdoors*)), stosowanie oprogramowania sterującego stworzonego bez uwzględnienia zabezpieczeń przeciwko atakom, adopcja niewłaściwych środków zabezpieczających przeciwko atakom, brak odpowiednich procedur zabezpieczających w trakcie interakcji personelu z systemem robotycznym (np. autentyfikacja, kodowanie wiadomości, autoryzacja, szkolenie operatorów). Ataki mogą być prowadzone przez: blokowanie łącz komunikacyjnych (np. przez ich zakłócanie lub spowodowanie odmowy świadczenia usług wywołane nadmiarowym ich żądaniem DoS (ang. *denial of service*)), przechwytywanie komunikacji, wstrzykiwanie do oprogramowania systemu wrogiego oprogramowania (np. wirusy, trojany), nieuprawnione zdobywanie informacji o sposobach dostępu do systemu, modyfikowanie przesyłanych informacji, nadmierne zużycie zasobów robota (np. wyczerpanie akumulatorów lub przeciążenie komputera sterującego), fizyczne uszkodzenie sprzętu wykorzystywanego przez system robotyczny lub zakłócanie działania jego czujników. Powodem ataków zazwyczaj jest albo chęć przejęcia kontroli nad systemem w celu upośledzenia jego funkcji lub pozyskiwanie w sposób nieuprawniony danych wrażliwych. Upośledzenie funkcji może polegać np. na zakodowaniu danych w celu uzyskania okupu w zamian za odkodowanie danych.

Współczesne roboty są sterowane komputerowo, więc ich oprogramowanie, jak widać z tego, co zostało napisane powyżej, podlega w dużej mierze tym samym zagrożeniom, co oprogramowanie innych systemów informatycznych. Oczywiście istnieją różnice, ale duża część zagrożeń jest wspólna, dlatego wiele metod ochrony systemów informatycznych przed atakami jest wykorzystywanych również przez robotyków. Modelowanie zagrożeń polega na stworzeniu abstrakcyjnego modelu oprogramowania w celu zidentyfikowania możliwości, celów oraz motywacji atakującego, tak aby w efekcie utworzyć katalog możliwych zagrożeń [131]. Proces modelowania zagrożeń ma prowadzić do identyfikacji i oceny zagrożeń dla firmy wykorzystującej oprogramowanie oraz określenia skutków ewentualnego ataku i do znalezienia środków zaradczych w przypadku ich wykrycia.

W pracy [127] podkreślono, że analiza zagrożeń wymaga stworzenia abstrakcyjnego modelu systemu oraz profili agresorów, biorąc pod uwagę zarówno ich cele jaki i metody działania. Aby taki model był przydatny do analizy zagrożeń bezpieczeństwa, powinien obejmować zarówno oprogramowanie, jak i osoby z niego korzystające [130]. Brak modelu użytkownika w istotny sposób utrudnia analizę problemów związanych z autentyfikacją, a w szczególności czyni kłopotliwym wskazanie możliwych metod przeprowadzenia ataku. Prace przeglądowe [78, 101, 127, 128] przedstawiają najbardziej popularne metody projektowania systemów informatycznych biorące pod uwagę zapewnienie cyberbezpieczeństwa. Większość przedstawianych metod koncentruje się na fazie formułowania wymagań dla projektowanego systemu oraz na jego specyfikacji.

Metoda modelowania zagrożeń STRIDE [78], [127] (nazwana tak od angielskich słów: *Spoofing*, *Tampering*, *Repudiation*, *Information disclosure*, *Denial of Service*, *Elevation of privilege*), opracowania w 1999 r., używana przez wiele lat przez firmę Microsoft, bierze pod uwagę szeroki repertuar możliwych zagrożeń. Są to:

- udawanie użytkownika uprawnionego do korzystania z systemu, dzięki naruszeniu procedur sprawdzających autentyczność (autentyfikacja),
- modyfikacja danych, czyli naruszanie ich integralności,
- wypieranie się swoich działań (zazwyczaj jest to związane z modyfikacją logów),
- naruszanie poufności danych,
- doprowadzanie do odmowy dostarczania usług przez wyczerpywanie zasobów (DoS) oraz
- zwiększanie swoich przywilejów w celu uzyskania nieuprawnionego dostępu do zastrzeżonych zasobów.

Analiza systemu prowadzona jest przez jego dekompozycję na podsystemy. Najczęściej stosuje się w tym celu diagramy przepływu danych DFD (ang. *Data Flow Diagram*). Diagramy te tworzone są z czterech typów pojęć: procesów (funkcji), magazynów danych, przepływu danych, jednostek zewnętrznych (terminatorów), a w przypadku analizy bezpieczeństwa uzupełniane są przez zaznaczenie obszarów zaufania (ich granice wyznaczają zmianę niezbędnego stopnia uprzywilejowania). DFD wybrano do przeprowadzenia analizy cyberbezpieczeństwa, ponieważ większość ataków dotyczy danych i ich przetwarzania [130]. Każdy z elementów takiego diagramu analizowany jest pod kątem wymienionych tu zagrożeń. Dla ułatwienia analizy stosuje się przygotowane szablony zawierające listy pytań, na które należy udzielić odpowiedzi. Wadą metody STRIDE jest to, że dla dużych systemów, składających się z wielu modułów, z których każdy może podlegać wielu zagrożeniom, określenie środków zaradczych jest żmudne. Ponadto niezbędna jest lista potencjalnych zagrożeń. Na podstawie wymienionych tam zagrożeń oraz dzięki znajomości struktury systemu formułowane są środki zaradcze.

Dlatego firma Microsoft zmodyfikowała metodę STRIDE, a w konsekwencji powstała metoda DREAD (ang. *Damage potential*, *Reproducibility*, *Exploitability*, *Affected users*, *Discoverability*) [70, 130], która po kilku kolejnych latach uległa dalszym modyfikacjom. W metodzie DREAD ocenia się:
– jak wielką szkodę może wyrządzić atak?
– jak łatwo jest odtworzyć atak?
– jak wiele pracy trzeba włożyć w przeprowadzenie ataku?
– jak wielu ludzi będzie dotkniętych skutkami ataku?
– jak trudno jest odkryć atak?

Przy stosowaniu DREAD ewaluacji liczbowej podlega ryzyko wystąpienia każdego z wymienionych zagrożeń. Pierwsze trzy zagrożenia oceniane są w skali trójstopniowej, a ostanie dwa w czterostopniowej.

W 2012 r. opracowano zestaw narzędzi o akronimie PASTA (ang. *Process for Attack Simulation and Threat Analysis* – proces symulacji ataku oraz analizy zagrożeń, czyli niepożądanych zdarzeń), służący do stworzenia modelu zagrożeń z punktu widzenia atakującego system. Siedmiokrokowy proces modelowania określa ryzyko wystąpienia zagrożeń. Pierwszy krok polega na zdefiniowaniu swoich celów biznesowych oraz określeniu partnerów, którzy będą mieli styczność z tworzonym oprogramowaniem oraz przepisów rządzących dziedziną, której oprogramowanie dotyczy. W szczególności należy określić, jakie wartościowe zasoby mogą być zagrożone, by je odpowiednio chronić. W drugim kroku określa się techniczne zasoby (oprogramowanie i sprzęt) i jakie są współzależności między poszczególnymi komponentami. Innymi słowy, określa się kontekst, w którym będzie funkcjonowało tworzone lub badane oprogramowanie. Trzeci krok zajmuje się dekompozycją systemu. Określa przypadki użycia, role, usługi, źródła danych, przywileje użytkowników. W tym kroku tworzy się diagramy przepływu danych DFD oraz określa obszary zaufania. W czwartym kroku następuje analiza zagrożeń, polegająca na budowie katalogu zagrożeń z punktu widzenia dziedziny zastosowań tworzonego oprogramowania oraz probabilistyczna analiza możliwych scenariuszy ataków. W tym kroku bada się logi, jeżeli istnieją, by określić przeszłe wektory ataków. Piąty krok to detekcja słabości na podstawie drzew zagrożeń. Krok szósty polega na modelowaniu ataków i analizie ich skutków. W tym celu budowane są drzewa ataków, a w efekcie określane są drogi i sposoby prowadzenia możliwych ataków. Istotnym elementem jest wskazanie, jak przypadki użycia mogą zostać przetransformowane w przypadki nadużycia. Wreszcie w kroku siódmym przeprowadzana jest analiza ryzyka pojawienia się poszczególnych ataków i określane są sposoby przeciwdziałania im. Do działań związanych z poszczególnymi krokami powstały narzędzia programistyczne wspomagające pracę analityków.

Koncepcja drzew ataków została opracowana w 1999 r. [126]. Korzeniem drzewa jest cel ataku, gałęźmi są cele cząstkowe, natomiast jego liśćmi są metody osiągnięcia celu. Dla analizowanego systemu konstruuje się wiele takich drzew, bo różne mogą być cele ataku. Gałęzie wychodzące z węzłów drzewa mogą być w relacji OR lub AND. Zazwyczaj określa się też koszt osiągnięcia tego celu przypisując wartości celom cząstkowym. Cenę ataku określa się poruszając od liści do korzenia. Koszt ataku może być określany biorąc pod uwagę różne kryteria, np. czy dana czynność jest łatwa do wykonania czy trudna, czy wymaga specjalistycznego sprzętu czy nie, jest legalna czy nie. Można też oszacować koszt monetarny wykonania czynności lub prawdopodobieństwo jej sukcesu, przez co można oszacować całkowity koszt lub prawdopodobieństwo ataku. Drzewa ataku konstruuje się albo dla całego systemu albo dla poszczególnych jego elementów. Powinny być skojarzone z wiedzą o umiejętnościach atakujących. Istnieją opracowane wskazówki, jak powinno się konstruować takie drzewa [126].

Metoda modelowania zagrożeń PnG (ang. *Persona non Grata*) koncentruje się na motywacjach i umiejętnościach atakujących [39]. Atakujących definiuje się jako osoby wchodzące w interakcje z systemem w sposób niedopuszczalny. Analitycy stosując tę metodę patrzą na system z punktu widzenia osoby atakującej. Przyglądają się każdej z wymaganych cech systemu przez pryzmat tego, jak może ona być nadużyta. W ten sposób określa się, jak oprogramowanie zareaguje na przypadek niewłaściwego użycia.

Karty bezpieczeństwa (ang. *Security Cards*) są techniką badania bezpieczeństwa za pomocą zestawu kart, które zawierają pytania i przykłady ataków [45]. Karty te używa się w trakcie sesji burzy mózgów, jako ułatwienie do generacji pomysłów dotyczących ataków możliwych do przeprowadzenia. Karty podzielone są na cztery zestawy ułatwiające udzielanie odpowiedzi na następujące grupy pytań:
– kto lub co (np. klimat) może ponieść stratę wskutek ataku?
– jakie są motywacje atakującego?
– jakie zasoby posiada atakujący?
– jakich używa metod ataku?

Metoda SQUARE (ang. *Security Quality Requirements Engineering Method*) została opracowana w 2005 r. przez Software Engineering Institute Carnegie Mellon University [100]. Celem opracowanej metody było wprowadzenie do projektowania systemów informatycznych, już w początkowej fazie ich tworzenia, elementów mających zapewnić bezpieczeństwo produktu finalnego. Jeżeli wymagania nałożone na projektowany system nie są poprawnie wyrażone, powstanie system nieodpowiedni lub niskiej jakości. Korekty na etapie wdrażania są bardzo kosztowne i prowadzą do projektów, które znacznie przekraczają budżet i trwają dłużej niż zakłada harmonogram. Metoda wymaga wykonania dziewięciu następujących kroków. Krok pierwszy prowadzi do uzgodnienia definicji pojęć używanych przez wszystkich udziałowców projektu – tu najczęściej korzysta się z definicji ujętych w standardach. Krok drugi polega na sformułowaniu wymagań dotyczących bezpieczeństwa systemu (identyfikacji celów). Trzeci krok dotyczy wskazania narzędzi, które zostaną wykorzystane do pracy nad stworzeniem bezpiecznego systemu, np. scenariuszy działania, przypadków użycia i nadużycia, formularzy. Krok czwarty zajmuje się oceną ryzyka związanego z poszczególnymi zagrożeniami. W kroku piątym dokonuje się wyboru technik pozyskiwania informacji od udziałowców o zagrożeniach. W kroku szóstym wybrane technik stosuje się do pozyskania niezbędnych informacji od udziałowców. Krok siódmy polega na kategoryzacji wymagań dotyczących bezpieczeństwa. W kroku ósmym przypisuje się priorytety poszczególnym kategoriom zagrożeń. W ostatnim kroku dokonuje się ostatecznej inspekcji efektów pracy i tworzy ostateczną dokumentację przeprowadzonego procesu decyzyjnego.

Ponieważ wiele ze wcześniej przedstawionych podejść do modelowania zagrożeń koncentruje się jedynie na pewnych aspektach dziedziny, stworzono również metody hybrydowe wykorzystujące kilka z wcześniej powstałych narzędzi. Jedną z takich metod jest hTMM (ang. *Hybrid Threat Modeling Method*) stworzona przez Software Engineering Institute Carnegie Mellon University w 2018 r. [101]. Została ona zainspirowana przez STRIDE. Łączy ona w sobie: SQUARE, Security Cards i PnG. Oto główne kroki metody. Wpierw należy określić system, dla którego ma być przeprowadzona analiza zagrożeń. Następnie należy zorganizować burzę mózgów z zastosowaniem kart bezpieczeństwa. W burzy mózgów powinni wziąć udział użytkownicy (obecni lub przyszli) oraz nabywcy systemu, jego projektanci i wdrożeniowcy oraz eksperci zajmujących się cyberbezpieczeństwem. Kolejnym krokiem jest usunięcie z zestawu osób atakujących mało prawdopodobnych PnG, tj. takich, dla których wektory ataku są nierealistyczne. Po tym

następuje podsumowanie wyników za pomocą narzędzi wspomagających. Te działania powinny doprowadzić do skonkretyzowania odpowiedzi na następujące pytania:
- kto lub co może zainicjować atak?
- jaka jest motywacja atakującego?
- co jest celem ataku?
- biorąc pod uwagę zasoby i umiejętności atakującego, jaki może być wektor ataku?
- jakie mogą być efekty ataku?
- jak dotkliwe mogą być efekty ataku?
- z jakim typem ataku mamy do czynienia (np. DoS, udawanie użytkownika)?

Na tej podstawie przeprowadza się formalną ocenę ryzyka, przykładowo używając metody SQUARE.

Metoda CVSS (ang. *Common Vulnerability Scoring System*), opracowana przez NIST (ang. *National Institute of Standards and Technology*), a obecnie utrzymywana przez stowarzyszenie FIRST, przypisuje wartość numeryczną podatności na zagrożenie [1]. Zastosowano trzy metryki: podstawową (ang. *base*), czasową (ang. *temporal*) oraz środowiskową (ang. *environmental*). Pierwsza zajmuje się zagrożeniami niezależnymi od czasu i środowiska, w którym działa system, druga zajmuje się zagrożeniami zależnymi od czasu, a trzecia tymi zależnymi od środowiska, które wykorzystuje użytkownik. Wpierw obliczana jest wartość metryki podstawowej, a następnie jest ona modyfikowana przez wartości wynikające z określenia wartości dwóch pozostałych metryk. Wartość metryki podstawowej określana jest przez organizację utrzymującą oprogramowanie, natomiast dwie pozostałe metryki określane są przez konkretnych użytkowników, którzy znają warunki, w których użytkują system. Przykładowo, zmiana zagrożenia w czasie może być spowodowana przez aktualizację systemu operacyjnego, natomiast zagrożenia wynikające z uwarunkowań środowiskowych zmieniają się w zależności od zabezpieczeń dodatkowych stosowanych przez konkretnego użytkownika. Metryka podstawowa zbudowana jest z dwóch zestawów wartości. Pierwszy dotyczy możliwości wykorzystania podatności na atak (ang. *exploitability*), a druga konsekwencji ataku (ang. *impact*). Pierwszy zestaw wskazuje na łatwość dokonania ataku na podatny komponent oraz niezbędne środki techniczne do jego przeprowadzenia. Drugi zestaw kwantyfikuje skutki pomyślnie przeprowadzonego ataku na ten komponent. Metryka bazowa daje wynik w zakresie 1–10 i jest uzupełniana przez wektor liczb wskazujący na wpływ powyżej wymienionych ocen cząstkowych.

Pełna ocena zagrożeń, którym może podlegać system robotyczny, możliwa jest jedynie w przypadku dogłębnej znajomości struktury i sposobu działania tego systemu [84]. Aby ocenić zagrożenia, wymienia się tu następujące wektory ataku (drogi, którymi ataki są przeprowadzane): sprzęt, sieć, system operacyjny i oprogramowanie wbudowane, a ponadto oprogramowanie aplikacyjne. Stwierdzono tam, że cyberbezpieczeństwo systemu wymaga: konfidencjonalności, czyli uniemożliwienia nieautoryzowanego ujawnienia danych, zachowania integralności, czyli zapobieżenia modyfikacji bądź skasowania danych, oraz dostępności, czyli zapewnienia dostępu do danych w sposób terminowy i niezawodny. Prócz danych zagrożone mogą być również inne zasoby, w szczególności urządzenia, i dostarczane usługi, a dokładniej kod programów je świadczące.

Analiza cyberbezpieczeństwa dotyczy zarówno sposobu konstrukcji systemów jak i ich użytkowania. W przypadku systemów informatycznych powszechnego użytku agresorzy mogą w celu zmylenia użytkownika podstawić nieprawdziwą stronę internetową, ale ten sposób jest powszechnie znany. W przypadku robotów należy zwrócić uwagę, że ludzie inaczej zachowują się wobec innych ludzi, a inaczej wobec robotów. Ludzie mają dużo większe zaufanie do robotów społecznych niż do innych ludzi. Nie przypuszczają, że roboty te mogą posłużyć do spowodowania szkód. Przykładowo, o ile nie byliby skłonni wpuścić do strefy zastrzeżonej obcej osoby, to zbadano, że nie mieli takich oporów w przypadku robotów [84]. Prócz problemów związanych z konstrukcją bezpiecznego oprogramowania rozważana jest kwestia, analizy istniejącego oprogramowania, które nie było stworzone według postulowanych zasad, a teraz należy określić stopień jego odporności na ataki. W tym celu prowadzone są prace badawcze mające na celu automatyczną konwersję takiego oprogramowania w modele standardowo używane do analizy odporności na ataki (np. DFD) [130].

Ostatecznym wynikiem powyżej opisanych metod analizy jest wskazanie środków zaradczych zapobiegających skutkom ataków. Przykładowymi środkami obrony są:
- Ściany ogniowe (firewall), służące blokowaniu niechcianych wiadomości nadchodzących z sieci, jednocześnie przepuszczające te pożądane,
- Systemy wykrywania i zapobiegania włamaniom (ang. *Intrusion Detection System, Intrusion Prevention System*), które porównują dane w przychodzących pakietach ze znanymi wzorcami ataków i je albo jedynie wykrywają albo również neutralizują ich szkodliwe działanie,
- Pułapki ściągające na siebie ataki (honeypot), umożliwiające zdobycie informacji o atakującym, dzięki stworzeniu sztucznego środowiska przypominającego rzeczywiste będące owocem pożądania intruza.

Należy zwrócić uwagę, że w przypadku robotów, nie tylko połączenie z siecią może być źródłem zagrożenia. Roboty korzystają z czujników do pozyskiwania informacji o stanie swego otoczenia. Atak może nastąpić poprzez wrogie oddziaływanie na czujniki. Taki atak będzie tworzył iluzję, którą układ sterowania robota potraktuje, jako rzeczywistość i zacznie podejmować niewłaściwe decyzje. To z kolei może prowadzić do jego uszkodzenia lub zniszczeń w jego otoczeniu. Jest to szczególnie niebezpieczne w przypadku robotów przeznaczonych do celów militarnych, które nierzadko są uzbrojone.

W przypadku robotów kompanów, lub szerzej patrząc robotów usługowych, dąży się do tego, by interakcja między człowiekiem a robotem odbywała się w języku naturalnym z użyciem mikrofonów i głośników, a także kamer w celu analizy wyrazu twarzy oraz gestykulacji. Nie każdy człowiek powinien mieć możliwość wydawania poleceń takiemu robotowi. W takim przypadku potrzebne są zabezpieczenia biometryczne, np. bazujące na rozpoznawaniu głosu, bądź twarzy [9], tęczówki [44], a nawet odcisków palców [118]. Ponadto każdy sterownik robota powinien być wyposażony w oprogramowanie monitorujące zachowanie tego robota, w celu detekcji anomalii. Minimalnym wymaganiem jest sygnalizacja wykrycia takiego nienormalnego zachowania, natomiast pożądanym jest by sterownik był w stanie dokonać korekty swego działania w celu osiągnięcia stanu normalnego (samonaprawa, ang. *self-healing*). Jeżeli to nie jest możliwe, to element systemu, który uległ atakowi powinien być izolowany. Jeżeli system jest w stanie jedynie wykryć atak, to powinien przechowywać logi rejestrujące jego aktywność, by możliwa była analiza prowadząca do wykrycia, co konkretnie się stało i skąd nastąpił atak. Ponadto należy podkreślić, że szyfrowanie komunikacji w istotny sposób utrudnia prowadzenie ataków.

Prócz ataków na wyprodukowanego i funkcjonującego już robota rozpatruje się również ataki prowadzone w trakcie jego wytwarzania lub projektowania – może dojść do sabotażu, wynikiem którego będzie celowe wstawienie do oprogramowania tylnego wejścia, przez które osoby niepowołane będą mogły w przyszłości dostać się do wnętrza systemu. W przypadku robotów wykorzystujących systemy sterowania o zmiennej strukturze (np. [167]), gdzie na potrzeby realizacji dużej różnorodności zlecanych zadań, robot musi wymieniać moduły

wykonujące te zadania, względnie łatwo jest przesłać z repozytorium umieszczonego w chmurze zainfekowany moduł. Roboty też mogą korzystać z usług świadczonych przez chmury obliczeniowe, np. przekazując obliczeniochłonne zadania do realizacji zdalnej. Przekazywanie danych i odbieranie wyników podatne jest na cyberatak. Co więcej, roboty działają w czasie rzeczywistym, więc spowolnienie otrzymania wyników przez atak typu DoS może w istotny sposób zaburzyć funkcjonowanie systemu.

Niestety nie ma jeszcze spójnego zestawu rekomendacji, jak konstruować cyberbezpieczne systemy robotyczne. Istnieją jedynie ogólne rekomendacje, głównie zaczerpnięte z inżynierii oprogramowania oraz zalecenia dotyczące wybranych form ataków na systemy konkretnego typu. Widać, że potrzebny tu jest duży wysiłek badawczy, by stworzyć odpowiednie standardy. Oczywiście standaryzacja jest wskazana, ale należy zwrócić uwagę, że badacze zajmujący się cyberbezpieczeństwem nadal poszukują rozwiązań niekonwencjonalnych. W szczególności szukają inspiracji w naturze [99, 124]. Zwracają uwagę, że konwencjonalny atak, składający się z:

1) instalacji wrogiego oprogramowania (ang. *malware*) – najczęściej dzięki oszukaniu użytkownika,
2) wprowadzenie zainfekowanego komputera do botnetu, dzięki czemu atakujący przejmuje kontrolę nad tym komputerem,
3) zainicjowanie kolejnego ataku z użyciem tego komputera,

ma swoje odpowiedniki w biologii. Przykładowo niektóre pająki zastawiają pułapki feromonowe, które powodują, że owady są wabione do ich pajęczyn, tak jak wabieni są internauci na niebezpieczne strony. Do przekazywania informacji w botnecie wykorzystuje się steganografię [153], czyli ukrywanie tajnej wiadomości w publicznie dostępnych danych, np. w obrazach, modyfikując je niezauważalnie dla ludzkiego oka. Podobnie działają niektóre grzyby, które infekują mrówki, powoli je trawiąc. W pewnym momencie dochodzi do przejęcia kontroli nad zachowaniem mrówki. Mrówka jest zmuszana do wędrówki do miejsca optymalnego dla rozwoju kolejnego pokolenia zarodników. Studiowanie zachowania roślin, grzybów i zwierząt pokazuje, jakie mogą być wektory ataku, ale również jak się przed nimi bronić.

Ponadto należy zwrócić uwagę, że cyberbezpieczeństwo, traktowane jako nauka, zajmuje się wykrywaniem oraz ochroną przed: wrogim przejęciem, modyfikacją zachowania lub zniszczeniem robota, ale zupełnie pomija możliwość samoistnej zmiany jego działania na potencjalnie groźne dla ludzi, a to jest przedmiotem troski wielu futurystów. Przez samoistną zmianę rozumiem tu transformację działania robota wskutek „świadomej" decyzji sztucznej inteligencji wbudowanej w jego sterownik.

3. Podsumowanie

Przegląd podstawowych czynników wpływających na inteligencję robotów, zawarty w tej części artykułu, wskazuje na następujące istotne fakty:

- opracowanie każdego z wymienionych tu czynników wymagało dużego wysiłku badawczego,
- w każdym z tych przypadków osiągnięto duży postęp,
- stworzenie inteligentnego robota wymaga inkorporacji wszystkich opisanych tu czynników do jego sterownika,
- patrząc na każdy z tych czynników oddzielnie, niestety osiągnięty stan jest nadal niewystarczający, by roboty mogły dorównać ludzkiej inteligencji, nie wspominając już o jej przewyższeniu.

W drugiej części tego artykułu pokazane będzie, na ile wybrane tu kluczowe technologie, niezbędne do wytworzenia inteligentnego zachowania robotów, w istocie umożliwiły stworzenie takich maszyn.

Bibliografia

1. Common Vulnerability Scoring System version 3.1: Specification Document CVSS Version 3.1 Release. https://www.first.org/cvss/ specification-document, 2019.
2. Google Brain Team. https://research.google/teams/brain/robotics/, 2022.
3. Aggarwal C.C., *Neural Networks and Deep Learning: A Textbook*. Springer, 2018, DOI: 10.1007/978-3-319-94463-0.
4. Alami R., Chatila R., Fleury S., Ghallab M.M., Ingrand F., *An architecture for autonomy*. The International Journal of Robotics Research, Vol. 17, No. 4, 1998, 315–337, DOI: 10.1177/027836499801700402.
5. Arabas J., *Wykłady z algorytmów ewolucyjnych*. Wydawnictwa Naukowo-Techniczne, Warszawa 2001.
6. Arkin R.C., *Behavior-Based Robotics*. MIT Press, 1998.
7. Armbrust C., Kiekbusch L., Ropertz T., Berns K., *Soft robot control with a behaviour-based architecture. Soft Robotics*, 81–91, Springer, 2015, DOI: 10.1007/978-3-662-44506-8_8.
8. Ashtiani F., Geers A., Aflatouni F., *An on-chip photonic deep neural network for image classification*. „Nature", No. 606, 2022, 501–506, DOI: 10.1038/s41586-022-04714-0.
9. Azimi M., Pacut A., *Investigation into the reliability of facial recognition systems under the simultaneous influences of mood variation and makeup*. „Computers & Electrical Engineering", Vol. 85, 2020, DOI: 10.1016/j.compeleceng.2020.106662.
10. Beetz M., Chatila R., Hertzberg J., Pecora F., *AI Reasoning Methods for Robotics*, [In:] Siciliano B., Khatib O. (red.), *Springer Handbook of Robotics*, Springer, 2016, 329–356, DOI: 10.1007/978-3-319-32552-1_14.
11. Beetz M., Mösenlechner L., Tenorth M., *CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments*. „IEEE/RSJ International Conference on Intelligent Robots and Systems", IROS, Taipei, Taiwan, 1012–1017, DOI: 10.1109/IROS.2010.5650146.
12. Beni G., *From swarm intelligence to swarm robotics*. „Lecture Notes in Computer Science", Vol. 3342, 2004, DOI: 10.1007/978-3-540-30552-1_1.
13. Beni G., Wang J., *Swarm intelligence in cellular robotic systems*. „Robots and Biological Systems: Towards a New Bionics", Springer, 1993, 703–712, DOI: 10.1007/978-3-642-58069-7_38.
14. Billard A., Calinon S., Dillmann R., *Learning from humans*, [In:] Siciliano B., Khatib O. (red.), *Springer Handbook of Robotics*, Springer, 2016, 1995–2014, DOI: 10.1007/978-3-319-32552-1_74.
15. Billard A., Calinon S., Dillmann R., Schaal S., *Robot programming by demonstration*. [In:] Siciliano B., Khatib O. (red.), *Springer Handbook of Robotics*, Springer, 2008, 1371–1394, DOI: 10.1007/978-3-540-30301-5_60.
16. Blodow N., Jain D., Marton Z., Beetz M., *Perception and probabilistic anchoring for dynamic world state logging*. 10th IEEE-RAS International Conference on Humanoid Robots, 2010, 160–166, DOI: 10.1109/ICHR.2010.5686341.
17. Blume C., Jakob W., *PASRO: Pascal for Robots*. Springer–Verlag, 1985.
18. Blume C., Jakob W., *Programming Languages for Industrial Robots*. Springer–Verlag, 1986.
19. Bodner J., Augustin F., Wykypiel H., Fish J., Muehlmann G., Wetscher G., Schmid T., *The da Vinci robotic system for general surgical applications: a critical interim appraisal*. "Swiss Medical Weekly", Vol. 135, 2005, 674–678, DOI: 10.4414/smw.2005.11022.

20. Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, Oxford, 1999.
21. Brady M., *Artificial intelligence and robotics*. "Artificial Intelligence", Vol. 26, No. 1, 1985, 79–121.
22. Brooks A., Kaupp T., Makarenko A., Williams S., Orebäck A., *Towards component-based robotics*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05), 2005, 163–168, DOI: 10.1109/IROS.2005.1545523.
23. Brooks A., Kaupp T., Makarenko A., Williams S., Orebäck A., *Orca: A component model and repository*. [In:] Brugali D. (red.), *Software Engineering for Experimental Robotics*, Vol. 30 serii Springer Tracts in Advanced Robotics, 2007, 231–251, DOI: 10.1007/978-3-540-68951-5_13.
24. Brooks R.A., *Elephants don't play chess*. "Robotics and autonomous systems", Vol. 6, No. 1–2, 1990, 3–15, DOI: 10.1016/S0921-8890(05)80025-9.
25. Brooks R.A., A robust layered control system for a mobile robot, "IEEE Journal of Robotics and Automation", Vol. 2, No. 1, 1986, 14–23, DOI: 10.1109/JRA.1986.1087032.
26. Brooks R.A., *Intelligence without reason*, Artificial Intelligence: critical concepts, 3:107–163, 1991.
27. Brooks R.A., *Intelligence without representation*. "Artificial Intelligence", Vol. 47, No. 1–3, 1991, 139–159, DOI: 10.1016/0004-3702(91)90053-M.
28. Brooks R.A., *New approaches to robotics*, "Science", Vol. 253, No. 5025, 1991, 1227–1232, DOI: 10.1126/science.253.5025.1227.
29. Brucker P., *Scheduling Algorithms*. Springer-Verlag, Berlin, Heidelberg, wyd. 5, 2007.
30. Brugali D., *Model-driven software engineering in robotics: Models Are Designed to Use the Relevant Things, Thereby Reducing the Complexity and Cost in the Field of Robotics*, "IEEE Robotics Automation Magazine", Vol. 22, No. 3, 2015, 155–166, DOI: 10.1109/MRA.2015.2452201.
31. Bruyninckx H., *Open robot control software: The OROCOS project*. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Vol. 3, 2001, 2523–2528, DOI: 10.1109/ROBOT.2001.933002.
32. Bruyninckx H., *The real-time motion control core of the OROCOS project*. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2003, 2766–2771, DOI: 10.1109/ROBOT.2003.1242011.
33. Bulter Z., Rizzi A., *Distributed and cellular robots*. [In:] Siciliano B., Khatib O. (red.), *Springer Handbook of Robotics*, Springer, 2008, 911–920, DOI: 10.1007/978-3-540-30301-5_40.
34. Carlisle B., *Robot mechanisms*. Proceedings of the 2000 ICRA Millenium IEEE International Conference on Robotics and Automation, 2000, DOI: 10.1109/ROBOT.2000.844134.
35. Carpenter G., Grossberg S., *Adaptive resonance theory*. [In:] Sammut C., Webb G. (red.), *Encyclopedia of Machine Learning*, Springer, 2010, 22–35, DOI: 10.1007/978-0-387-30164-8_11.
36. Chaumette F., Hutchinson S., Corke P., *Visual Servoing*. [In:] *The Handbook of Robotics*, Springer, 2016, 841–866.
37. Chibani A., Amirat Y., Mohammed S., Matson E., Hagita N., Barreto M., *Ubiquitous robotics: Recent challenges and future trends*. "Robotics and Autonomous Systems", Vol. 61, No. 11, 2013, 1162–1172, DOI: 10.1016/j.robot.2013.04.003.
38. Cichosz P., *Systemy uczące się*. Wydawnictwa Naukowo-Techniczne, Warszawa 2007.
39. Cleland-Huang, J., *How well do you know your personae non gratae*? "IEEE Software", Vol. 31, No. 4, 2014, 28–31, DOI: 10.1109/MS.2014.85.
40. Collett T., MacDonald B., Gerkey B., *Player 2.0: Toward a practical robot programming framework*. Proceedings of the Australasian Conference on Robotics and Automation (ACRA), December 2005.
41. Cooper B., *Driving on the surface of mars using the rover control workstation*. Proceedings of the International Conference on Space Operations, SpaceOps '98, June 1–5 1998.
42. Coradeschi S., Saffiotti A., *An introduction to the anchoring problem*. "Robotics and Autonomous Systems", Vol. 43, No. 2-3, 2003, 85–96, DOI: 10.1016/S0921-8890(03)00021-6.
43. Coste-Maniere E., Simmons R., *Architecture, the backbone of robotic systems*. Proc. IEEE International Conference on Robotics and Automation ICRA '00, Vol. 1, 2000, 67–72.
44. Czajka A., Kasprzak W., Wilkowski A., *Verification of iris image authenticity using fragile watermarking*. "Bulletin of the Polish Academy of Sciences: Technical Sciences", Vol. 64, No. 4, 2016, 807–819, DOI: 10.1515/bpasts-2016-0090.
45. Denning T., Friedman B., Kohno T., *The security cards – a security threat brainstorming toolkit*. https: //securitycards.cs.washington.edu/, 2013.
46. Dorigo M., Floreano D., Gambardella L., Mondada F., Nolfi S., Baaboura T., Birattari M., Bonani M., Brambilla M., Brutschy A., Burnier D., Campo A., Christensen A., Decugniere A., Di Caro G., Ducatelle F., Ferrante E., Forster A., Gonzales J., Guzzi J., Longchamp V., Magnenat S., Mathews N., Montes de Oca M., O'Grady R., Pinciroli C., Pini G., Retornaz P., Roberts J., Sperati V., Stirling T., Stranieri A., Stutzle T., Trianni V., Tuci E., Turgut A.E., Vaussard F., *Swarmanoid: A novel concept for the study of heterogeneous robotic swarms*. "IEEE Robotics & Automation Magazine", Vol. 20, No. 4, 2013, 60–71, DOI: 10.1109/MRA.2013.2252996.
47. Doriya R., Mishra S., Gupta S., *A brief survey and analysis of multi-robot communication and coordination*, 2015 International Conference on Computing, Communication Automation (ICCCA), 1014–1021, DOI 10.1109/CCAA.2015.7148524.
48. Driankov D., Hellendoorn H., Reinfrank M., *An Introduction to Fuzzy Control*. Springer-Verlag, Berlin, Heidelberg, 1993.
49. Dudek G., Jenkin M.R.M., Milios E., Wilkes D., *A taxonomy for multi-agent robotics*. "Autonomous Robots", Vol. 3, No. 41996, 375–397, DOI: 10.1007/BF00240651.
50. Dudek W., Szynkiewicz W., Winiarski T., *Nao Robot Navigation System Structure Development in an Agent-Based Architecture of the RAPP Platform*. [In:] R. Szewczyk, C. Zieliński, M. Kaliczyńska (red.), *Recent Advances in Automation, Robotics and Measuring Techniques*, Vol. 440 serii Advances in Intelligent Systems and Computing (AISC), Springer, 2016, 623–633, DOI: 10.1007/978-3-319-29357-8_54.
51. Dudek W., Winiarski T., *Scheduling of a Robot's Tasks with the TaskER Framework*. "IEEE Access", Vol. 8, 2020, 161449–161471, DOI: 10.1109/ACCESS.2020.3020265.
52. Farinelli A., Iocchi L., Nardi D., *Multirobot systems: a classification focused on coordination*. "IEEE Transactions on Systems, Man, and Cybernetics", Part B (Cybernetics), Vol. 34, No. 52004, 2015–2028, DOI: 10.1109/TSMCB.2004.832155.
53. Fedrizzi A., Mosenlechner L., Stulp F., Beetz M., *Transformational planning for mobile manipulation based on action-related places*. 14th International Conference on Advanced Robotics, ICAR, Munich, Germany, June 22–26 2009.
54. Ferrer-Mestres J., Dietterich T., Buffet O., Chadès I., *Solving k-mdps*. Proceedings of the International Conference

on Automated Planning and Scheduling, Vol. 30, 2020, 110–118, DOI: 10.1609/icaps.v30i1.6651.
55. Figat M., Zieliński C., *Parameterised robotic system metamodel expressed by Hierarchical Petri nets*. "Robotics and Autonomous Systems", Vol. 150, 2022, DOI: 10.1016/j.robot.2021.103987.
56. Fikes R., Nilsson N., *Strips: A new approach to the application of theorem proving to problem solving*. "Artificial intelligence", Vol. 2, No. 3–4, 1971, 189–208, DOI: 10.1016/0004-3702(71)90010-5.
57. Fitzpatrick P., Metta G., Natale L., *Towards long-lived robot genes*. "Robotics and Autonomous Systems", Vol. 56, No. 1, 2008, 29–45, DOI: 10.1016/j.robot.2007.09.014.
58. Gardner M., *Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life"*. "Scientific American", Vol. 223, 1970, 120–123, DOI: 10.1038/scientificamerican1070-120.
59. Gat E., *On three-layer architectures*. D. Kortenkamp, R. P. Bonnasso, R. Murphy (red.), Artificial Intelligence and Mobile Robots, 195–210. AAAI Press Cambridge, MA, 1998.
60. Gerkey B.P., Vaughan R.T., Støy K., Howard A., Sukhatme G.S., Mataric M.J., *Most Valuable Player: A Robot Device Server for Distributed Control*. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2001, 1226–1231, DOI: 10.1109/IROS.2001.977150.
61. Ghallab M., Howe A., Knoblock C., Mcdermott D., Ram A., Veloso M., Weld D., Wilkins D., *PDDL—The Planning Domain Definition Language*, 1998.
62. Gierse G., Niemueller T., Claßen J., Lakemeyer G., *Interruptible Task Execution with Resumption in Golog*. G. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen, (red.), Twenty-Second European Conference on Artificial Intelligence (ECAI), 2016, 1265–1273, Den Haag, The Netherlands, IOS Press, DOI: 10.3233/978-1-61499-672-9-1265.
63. Gordon D., *Ant Encounters: Interaction, Networks and Colony Behavior. Primers in Complex Systems*. Princeton University Press, 2010.
64. Gregory P., Long D., Fox M., Beck C., *Planning modulo theories: Extending the planning paradigm*. Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 22, No. 1, 2012, 65–73, DOI: 10.1609/icaps.v22i1.13505.
65. Grossberg S., *Conscious Mind, Resonant Brain: How Each Brain Makes a Mind*. Oxford University Press, 2021.
66. Guarino N., Oberle D., Staab S., *What is an ontology*? S. Staab, R. Studer (red.), *Handbook on Ontologies*, 1–20. Springer-Verlag, Berlin, 2009.
67. Hayward V., Paul R.P., *Robot manipulator control under Unix RCCL: A robot control C library*. "International Journal of Robotics Research", Vol. 5, No. 4, 1986, 94–111, DOI: 10.1177/027836498600500407.
68. Hebb D., *The Organization of Behavior*. Wiley & Sons, New York, 1949.
69. Horsch M., Poole D., *An anytime algorithm for decision making under uncertainty*. XIV Conference on Uncertainty in Artificial Intelligence, UAI'98, 1998, 246–255, DOI: 10.5555/2074094.2074123.
70. Howard M., Lipner S., *The Security Development Lifecycle*. Microsoft Press, 2006.
71. Höller D., Behnke G., Bercher P., Biundo S., Fiorino H., Pellier D., Alford R., *HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems*. Vol. 34, No. 6, 2020, 9883–9891, DOI: 10.1609/aaai.v34i06.6542.
72. Jesus A., Liefooghe A., Derbel B., Paquete L., *Algorithm selection of anytime algorithms*. Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 850–858, DOI: 10.1145/3377930.3390185.
73. Jonsson P., Krokhin A., *Complexity classification in qualitative temporal constraint reasoning*. "Artificial Intelligence", Vol. 160, No. 1-2, 2004, 35–51, DOI: 10.1016/j.artint.2004.05.010.
74. Kacprzyk J., *Rozumowanie w warunkach niepewności*. [In:] W. Traczyk (red.), *Problemy sztucznej inteligencji*, 77–114, Warszawa, 1995. Wiedza i Życie.
75. Kaelbling L., Littman M., Cassandra A., *Planning and acting in partially observable stochastic domains*. "Artificial Intelligence", Vol. 101, No. 1-2, 1998, 99–134, DOI: 10.1016/S0004-3702(98)00023-X.
76. Kaelbling L., Lozano-Perez T., *Integrated task and motion planning in belief space*. "The International Journal of Robotics Research", Vol. 32, No. 9-10, 2013, 1194–1227, DOI: 10.1177/0278364913484072.
77. Kaisler S., *Software Paradigms*. Wiley Interscience, Hoboken, 2005.
78. Kaneko T., *Threat analysis using STRIDE with STAMP/STPA*. Proceedings of the International Workshop on Evidence-based Security and Privacy in the Wild and the 1st International Workshop on Machine Learning Systems Engineering, wolumen CEUR-2809, Nara, Japan, 2018.
79. Kasprzak W., *Rozpoznawanie obrazów i sygnałów mowy*. Oficyna Wydawnicza Politechniki Warszawskiej, 2009.
80. Kavraki L., LaValle S., *The handbook of robotics. Motion Planning*, 139–161. Springer, 2016.
81. Kiekbusch L., Armbrust C., Berns K., *Formal verification of behaviour networks including sensor failures*. "Robotics and Autonomous Systems", Vol. 74, 2015, 331–339, DOI: 10.1016/j.robot.2015.08.002.
82. Kiekbusch L., Armbrust C., Berns K., *Formal verification of behaviour networks including hardware failures*. Vol. 302, 2016, 571–1582, DOI: 10.1007/978-3-319-08338-4_113.
83. Kortenkamp D., Simmons R., Brugali D., *Robotic systems architectures and programming*. B. Siciliano, O. Khatib (red.), Springer Handbook of Robotics, 2nd Edition, 283–306, Springer, 2016, DOI: 10.1007/978-3-319-32552-1_12.
84. Lacava G., Marotta A., Martinelli F., Saracino A., La Marra A., Gil-Uriarte E., Mayoral-Vilches V., *Cybersecurity issues in robotics*. "Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications", Vol. 12, No. 3, 2021, 1–28, DOI: 10.22667/JOWUA.2021.09.30.001.
85. Langdon W., Poli R., McPhee N., Koza J., *Genetic programming: An introduction and tutorial, with a survey of techniques and applications*. J. Fulcher, L. Jain (red.), Computational Intelligence: A Compendium, Vol. 115, 2008, 927–1028, Berlin, Heidelberg, DOI: 10.1007/978-3-540-78293-3_22.
86. Langton C., *Self-reproduction in cellular automata*. "Physica D: Nonlinear Phenomena", Vol. 10, No. 1, 1984, 135–144, DOI: 10.1016/0167-2789(84)90256-2.
87. Langton C., *Artificial Life : An Overview*. MIT Press, Cambridge Mass., 1995.
88. LaValle S., *Planning Algorithms*. Cambridge University Press, 2006.
89. Lebedev M., Nicolelis M., *Brain-machine interfaces: From basic science to neuroprostheses and neurorehabilitation*. "Physiological Reviews", Vol. 97, No. 2, 2017, 767–837, DOI: 10.1152/physrev.00027.2016.
90. LeCun Y., *A path towards autonomous machine intelligence*. https://openreview.net/pdf?id=BZ5a1rkVsf, 2022.
91. Levine S., Pastor P.S., Krizhevsky A., Ibarz J., Quillen D., *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*. "Internatio-

nal Journal of Robotics Research", Vol. 37, No. 4–5, 2018, 421–436, DOI: 10.1177/0278364917710318.
92. Litman T., *Autonomous vehicle implementation predictions implications for transport planning*. Victoria Transport Policy Institute, November 2018.
93. Luger G.F., Stubblefield W.A., *Artificial Intelligence and the Design of Expert Systems*. Benjamin-Cummings, Redwood, 1989.
94. Lyons D.M., *Prerational intelligence*, Vol. 2: Adaptive behavior and intelligent systems without symbols and logic serii Studies in cognitive systems, [In:] A Schema-Theory Approach to Specifying and Analysing the Behavior of Robotic Systems, 51–70. Kluwer Academic, 2001.
95. Lyons D.M., Arbib M.A., *A formal model of computation for sensory-based robotics*. "IEEE Transactions on Robotics and Automation", Vol. 5, No. 3, 1989, 280–293, DOI: 10.1109/70.34764.
96. Matarić M.J., Michaud F., *The Handbook of Robotics, Behavior-Based Systems*, 2008, 891– 909. Springer, DOI: 10.1007/978-3-540-30301-5_39.
97. Matarić M.J., *Issues and approaches in the design of collective autonomous agents*. "Robotics and Autonomous Systems", Vol. 16, No. 2-4, 1995, 321–331, DOI: 10.1016/0921-8890(95)00053-4.
98. Matuszek C., Cabral J., Witbrock M., DeOliveira J., *An Introduction to the Syntax and Content of Cyc*. Proc. 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, 44–49, Stanford, California, March 2006.
99. Mazurczyk W., Rzeszutko E., *Security–a perpetual war: Lessons from nature*. "IEEE IT Professional", Vol. 17, 2015, 16–22, DOI: 10.1109/MITP.2015.14.
100. Mead N., Hough E., Stehney T., *Security quality requirements engineering* (*SQUARE*). Raport instytutowy CMU/SEI-2005-TR-009, Carnegie Mellon University, Software Engineering Institute, November 2005.
101. Mead N., Shull F., Vemuru K., Villadsen O., *A hybrid threat modeling method*. Raport instytutowy CMU/SEI-2018-TN-002, Carnegie Mellon University, Software Engineering Institute, March 2018.
102. Metta G., Fitzpatrick P., Natale L., *YARP: Yet Another Robot Platform*. "International Journal on Advanced Robotics Systems", Vol. 3, No. 1, 2006, 43–48, DOI: 10.5772/5761.
103. Munson G., *The rise and fall of Unimation Inc. – story of robotics innovation & triumph that changed the world*! "Robot Magazine", December 2010.
104. Müller A., Kirsch A., Beetz M., *Transformational planning for everyday activity*. 17th International Conference on Automated Planning and Scheduling, ICAPS, 248–255, Providence, Rhode Island, September 22–26 2007.
105. Nalepa G., *From Good Robots to Useful Intelligent Agents*. [In:] K. Tchoń, W. Gasparski, (red.), *A Treatise on Good Robots*, 159–170. Transaction Publishers, 2014.
106. Nesnas I., *The CLARAty project: Coping with hardware and software heterogenity*. [In:] D. Brugali (red.), *Software Engineering for Experimental Robotics*, 31–70. Springer–Verlag, 2007.
107. Nilsson N., *Shakey the robot*. SRI International Technical note 323, 1984.
108. Nordmann A., Hochgeschwender N., Wigand D., Wrede S., *A survey on domain-specific modeling and languages in robotics*. "Journal of Software Engineering for Robotics", Vol. 7, 2016, 75–99.
109. Nwana H., *Software agents: an overview*. "The Knowledge Engineering Review", Vol. 11, No. 3, 1996, 205–244, DOI: 10.1017/S026988890000789X.
110. Nwana H.S., Ndumu D.T., *A Brief Introduction to Software Agent Technology*, 29–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
111. Padgham L., Winikoff M., *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons, 2004.
112. Pan Z., Polden J., Larkin N., Duin S.V., Norrish J., *Recent progress on programming methods for industrial robots*. "Robotics and Computer-Integrated Manufacturing", Vol. 28, No. 2, 2012, 87–94, DOI: 10.1016/j.rcim.2011.08.004.
113. Parker L., Rus D., Sukhatme G., *The handbook of robotics. Multiple Mobile Robot Systems*, 1335–1380. Springer, 2016.
114. Parker L.E. *Multiple mobile robot systems*. [In:] O. Khatib, B. Siciliano (red.), *Springer Handbook of Robotics*, 921–941. Springer, June 2008.
115. Pałka P., *Wieloagentowe systemy decyzyjne*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2019.
116. Pedersen M.R., Nalpantidis L., Andersen R.S., Schou C., Bøgh S., Krüger V., Madsen O., *Robot skills for manufacturing: From concept to industrial deployment*. "Robotics and Computer-Integrated Manufacturing", Vol. 37, 2016, 282–291, DOI: 10.1016/j.rcim.2015.04.002.
117. Pilone D., Pitman N., *UML 2.0 in a Nutshell*. O'Reilly, 2005.
118. Pinto A., Pedrini H., Krumdick M., Becker B., Czajka A., Bowyer K., Rocha A., *Counteracting presentation attacks in face, fingerprint, and iris recognition*. [In:] M. Vatsa, R. Singh, A. Majumdar (red.), *Deep Learning in Biometrics*. CRC Press, 2018.
119. Pretz K., *Don't Trust Deep Learning, Says Brain Modeler – Stephen Grossberg maintains his approach to intelligence is better*. "IEEE Spectrum", 54–55, June 2022.
120. Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R., Ng A., *ROS: an open-source Robot Operating System*. Proceedings of the Open-Source Software Workshop at the International Conference on Robotics and Automation (ICRA), 2009.
121. Rubenstein M., Cornejo A., Nagpal R., *Programmable self-assembly in a thousand-robot swarm*. "Science", Vol. 345, No. 6198, 2014, 795–799, DOI: 10.1126/science.1254295.
122. Russell S., Norvig P., *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, N.J., 2003.
123. Rutkowski L., *Metody i techniki sztucznej inteligencji*. Wydawnictwo Naukowe PWN, Warszawa, 2005.
124. Rzeszutko E., Mazurczyk W., *Insights from nature for cybersecurity*. "Health security", Vol. 13, No. 2, 2014, 82–87, DOI: 10.1089/hs.2014.0087.
125. Sacha K., *Inżynieria oprogramowania*. Wydawnictwo Naukowe PWN, Warszawa, 2010.
126. Schneier B., *Attack trees*. "Dr. Dobb's Journal", December 1999.
127. Shevchenko N., *Threat modeling: 12 available methods*. Raport instytutowy, Carnegie Mellon University, Software Engineering Institute, 2018
128. Shevchenko N., Chick T., O'Riordan P., Scanlon T., Woody C., *Threat modeling: A summary of available methods*. Raport instytutowy, Carnegie Mellon University, Software Engineering Institute, July 2018.
129. Shoham Y., *Agent-oriented programming*. "Artificial Intelligence", Vol. 60, No. 1, 1993, 51–92, DOI: 10.1016/0004-3702(93)90034-9.
130. Shostack A., *Experiences Threat Modeling at Microsoft*. MODSEC@MoDELS, 2008.
131. Shull F., *Evaluation of threat modeling methodologies*. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=474197, 2016.

132. Siciliano B., Sciavicco L., Villani L., Oriolo G., *Robotics: Modelling, Planning and Control. Advanced Textbooks in Control and Signal Processing*. Springer, London, 2009.
133. Silver D., Huang A., Maddison C., Guez A., Sifre L., Driessche G., Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., Dieleman S., Grewe D., Nham J., Kalchbrenner N., Sutskever I., Lillicrap T., Leach M., Kavukcuoglu K., Graepel T., Hassabis D., *Mastering the game of go with deep neural networks and tree search*. "Nature", Vol. 529, 2016, 484–489.
134. Simoens P., Dragone M., Saffiotti A., *The internet of robotic things: A review of concept, added value and applications*. "International Journal of Advanced Robotic Systems", Vol. 15, No. 1, 2018, DOI: 10.1177/1729881418759424.
135. Staniak M., Zieliński C., *Structures of visual servos*. "Robotics and Autonomous Systems", Vol. 58, No. 8, 2010, 940–954, DOI: 10.1016/j.robot.2010.04.004.
136. Stenmark M., Haage M., Topp E.A., *Simplified programming of re-usable skills on a safe industrial robot – prototype and evaluation*. Proceedings of the IEEE/ACM Conference on Human-Robot Interaction (HRI), Vienna, Austria, March, 6–9 2017, DOI: 10.1145/2909824.3020227.
137. Taleby Ahvanooey M., Li Q., Wu M., Wang S., *A survey of genetic programming and its applications*. KSII Transactions on Internet and Information Systems, Vol. 13, No. 4, 2019, 1765–1793, DOI: 10.3837/tiis.2019.04.002.
138. Tang F., Parker L., *A Complete Methodology for Generating Multi-Robot Task Solutions using ASyMTRe-D and Market-Based Task Allocation.*, 2007 IEEE International Conference on Robotics and Automation, 3351–3358, DOI: 10.1109/ROBOT.2007.363990.
139. Tenorth M., Beetz M., *KnowRob: a knowledge processing infrastructure for cognition-enabled robots*. "International Journal of Robotics Research", Vol. 32, No. 52013, 566–590, DOI: 10.1177/0278364913481635.
140. Tenorth M., Perzylo A.C., Lafrenz R., Beetz M., *The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments*. IEEE International Conference on Robotics and Automation. 2012, DOI: 10.1109/ICRA.2012.6224812.
141. Tenorth M., Perzylo A.C., Lafrenz R., Beetz M., *Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework*. "IEEE Transactions on Automation Science and Engineering", Vol. 10, No. 3, 2013, 643–651, DOI: 10.1109/TASE.2013.2244883.
142. Traczyk W., *Metody wnioskowania w systemach ekspertowych*. [In:] W. Traczyk (red.), *Problemy sztucznej inteligencji*, 53–75, Warszawa, 1995. Wiedza i Życie.
143. Turing A., *Computing machinery and intelligence*. "Mind", Vol. 59, No. 236, 1950, 433–460, DOI: 10.1093/mind/LIX.236.433.
144. Vaughan R.T., Gerkey B.P., *Reusable robot software and the Player/Stage project*. [In:] D. Brugali (red.), *Software Engineering for Experimental Robotics*, Vol. 30 serii Springer Tracts in Advanced Robotics, 267–289, Springer, 2007, DOI: 10.1007/978-3-540-68951-5_16.
145. Čapek K., *Four Plays: R.U.R., The Insect Play, The Makropulos Case, The White Plague*. Methuen Publishing Ltd., 1999.
146. von Neumann J., *The general and logical theory of automata*. Cerebral Mechanisms in Behavior. The Hixon Symposium, 1–41. John Wiley & Sons, New York, 1951.
147. Wooldridge M., *Multiagent systems. Intelligent Agents*, 27–77. MIT Press, Cambridge, MA, USA, 1999.
148. Xia X., Pan X., Li N., He X., Ma L., Zhang X., Ding N., *GAN-based anomaly detection: A review*. "Neurocomputing", Vol. 493, 2022, 497–535, DOI: 10.1016/j.neucom.2021.12.093.
149. Yaacoub J.-P., Noura H., Salman O., Chehab A., *Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations*. "International Journal of Information Security", Vol. 21, No. 1, 2022, 115—158, DOI: 10.1007/s10207-021-00545-8.
150. Yim M., Shen W.-M., Salemi B., Rus D., Moll M., Lipson H., Klavins E., Chirikjian G.S., *Modular self-reconfigurable robot systems* [*grand challenges of robotics*]. "IEEE Robotics & Automation Magazine", Vol. 14, No. 1, 2007, 43–52, DOI: 10.1109/MRA.2007.339623.
151. Zhang X., Zhu Y., Ding Y., Zhu Y., Stone P., Zhang S., *Visually grounded task and motion planning for mobile manipulation*. IEEE International Conference on Robotics and Automation (ICRA), May 23–27 2022.
152. Zheng N., Mazumder P., *Fundamentals and learning of artificial neural networks*. [In:] *Learning in EnergyEfficient Neuromorphic Computing: Algorithm and Architecture Co-Design*, 11–60, 2020.
153. Zielińska E., Mazurczyk W., Szczypiorski K., *Trends in steganography*. "Communications of the ACM", Vol. 57, No. 3, 2014, 86–95, DOI: 10.1145/2566590.2566610.
154. Zieliński C., *TORBOL: An object level robot programming language*. "Mechatronics", Vol. 1, No. 4, 1991, 469–485, DOI: 10.1016/0957-4158(91)90032-6.
155. Zieliński C., *Description of semantics of robot programming languages*. "Mechatronics", Vol. 2, No. 2, 1992, 171–198, DOI: 10.1016/0957-4158(92)90030-R.
156. Zieliński C., *The MRROC++ system*. Proceedings of the First Workshop on Robot Motion and Control, RoMoCo '99, 147–152, DOI: 10.1109/ROMOCO.1999.791067.
157. Zieliński C., *A Quasi-Formal Approach to Structuring Multi-Robot System Controllers*. Second International Workshop on Robot Motion and Control, RoMoCo'01, 121–128, DOI: 10.1109/ROMOCO.2001.973442.
158. Zieliński C., *Formal approach to the design of robot programming frameworks: the behavioural control case*. "Bulletin of the Polish Academy of Sciences – Technical Sciences", Vol. 53, No. 1, 2005, 57–67.
159. Zieliński C., *Systematic approach to the design of robot programming frameworks*. Proceedings of the 11th IEEE International Conference on Methods and Models in Automation and Robotics (on CD), 639–646. Technical University of Szczecin, 2005.
160. Zieliński C., *Transition-function based approach to structuring robot control software*. [In:] K. Kozłowski (red.), *Robot Motion and Control*, Vol. 335 serii Lecture Notes in Control and Information Sciences, 2006, 265–286. Springer-Verlag.
161. Zieliński C., *Koboty – w przemyśle i laboratoriach badawczych*. „Automatyka", Nr 12, 2020, 28–40, 2020.
162. Zieliński C., *Robotic System Design Methodology Utilising Embodied Agents*, Vol. 296 serii Advances in Intelligent Systems and Computing, 523–561. Springer, 2021.
163. Zieliński C., Kornuta T., Stefańczyk M., Szynkiewicz W., Trojanek P., Walęcki M., *Języki programowania robotów przemysłowych*. „Pomiary Automatyka Robotyka", Vol. 16, Nr 11, 2012, 10–19.
164. Zieliński C., Stefańczyk M., Kornuta T., Figat M., Dudek W., Szynkiewicz W., Kasprzak W., Figat J., Szlenk M., Winiarski T., Banachowicz K., Zielińska T., Tsardoulias E.G., Symeonidis A.L., Psomopoulos F.E., Kintsakis A.M., Mitkas P.A., Thallas A., Reppou S.E., Karagiannis G.T., Panayiotou K., Prunet V., Serrano M., Merlet J.-P.,

Arampatzis S., Giokas A., Penteridis L., Trochidis I., Daney D., Iturburu M., *Variable structure robot control systems: The RAPP approach*. "Robotics and Autonomous Systems", Vol. 94, 2017, 226–244, DOI: 10.1016/j.robot.2017.05.002.
165. Zieliński C., Kornuta T., *Diagnostic requirements in multi-robot systems*. [In:] J. Korbicz, M. Kowal (red.), Intelligent Systems in Technical and Medical Diagnostics, Vol. 230 serii Advances in Intelligent Systems and Computing, 345–356. Springer, 2014.
166. Zieliński C., Kornuta T., *Programowe struktury ramowe do tworzenia sterowników robotów*, „Pomiary Automatyka Robotyka", R. 19, Nr 1, 2015, 5-14, DOI: 10.14313/PAR_215/5.
167. Zieliński C., Szynkiewicz W., Mianowski K., Nazarczuk K., *Mechatronic design of open-structure multirobot controllers*. "Mechatronics", Vol. 11, No. 8, 2001, 987–1000, DOI: 10.1016/S0957-4158(00)00038-6.
168. Zieliński C., Trojanek P., *Stigmergic cooperation of autonomous robots*. "Journal of Mechanism and Machine Theory", Vol. 44, No. 4, 2009, 656–670, DOI: 10.1016/j.mechmachtheory.2008.08.012.

Robotics: Techniques, Functions, Social Role
Part 1. Technical Foundations of Intelligence and Security of Robots

Abstract: In order to assess the impact of robots on society, it is necessary to carefully analyze the state-of-the-art, and in particular the fundamental issues that have yet to be resolved, however having significant impact on the potential societal changes resulting from the development of robotics. The aforementioned impact depends on the level of intelligence of robots, so this aspect dominates in the presented analysis. The presentation has been divided into three parts: 1) analysis of technical factors affecting the intelligence and security of robots, 2) analysis of current capabilities of robots, 3) analysis of diverse predictions of how robotics will evolve, and thus the attitudes towards the influence of the result of this development on society. This part of the paper is devoted to the first of the above mentioned three issues.

Keywords: robot, robotic system, artificial intelligence, robot cybersecurity

prof. dr hab. inż. Cezary Zieliński
cezary.zielinski@pw.edu.pl
ORCID: 0000-0001-7604-8834

Jest profesorem na Wydziale Elektroniki i Technik Informacyjnych (WETI) Politechniki Warszawskiej (PW). W PW pracuje od 1985 r., a od 2008 r. również w Przemysłowym Instytucie Automatyki i Pomiarów. W PW sprawował funkcje: prodziekana ds. nauki i współpracy międzynarodowej WEITI (2002-2005), zastępcy dyrektora ds. naukowych Instytutu Automatyki i Informatyki Stosowanej (IAiIS) (2005-2008), dyrektora IAIS (2008-2016, 2020–) oraz prodziekana ds. ogólnych WEiTI (2016–2020). Od 1996 r. jest kierownikiem Zespołu Robotyki w IAiIS. Od 2007 roku jest członkiem Komitetu Automatyki i Robotyki Polskiej Akademii Nauk. Pracował również w Loughborough University of Technology (1992) oraz Nanyang Technological University (1999–2001). Jego zainteresowania badawcze koncentrują się na zagadnieniach związanych z metodami programowania i sterowania robotów. Jest autorem i współautorem ponad 200 publikacji z tego zakresu.