Pomiary Automatyka Robotyka, ISSN 1427-9126, R. 25, Nr 4/2021, 37–44, DOI: 10.14313/PAR_242/37

System sterowania dwurękiego robota usługowego

Dawid Seredyński
Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa

Streszczenie: Praca prezentuje przykład systemu sterowania robota usługowego. Opisano zastosowane narzędzia i otwarte oprogramowanie. Przedstawiono system sterowania poczynając od struktury sprzętu przez specyfikację, aż do implementacji. Opis na poziomie ogólnym pozwala spojrzeć całościowo na problem tworzenia takich systemów, a jednocześnie podkreślono szczegółowe kwestie, które są istotne. Poruszono także kwestie związane z symulacją. Opisany system sterowania robota WUT Velma znalazł zastosowanie w licznych badaniach naukowych.

Słowa kluczowe: system sterowania, robot dwuręki, robotyka usługowa, specyfikacja, agent

1. Wprowadzenie

Stworzenie systemu sterowania dla robota usługowego jest zadaniem trudnym. Chociaż na rynku dostępne są liczne gotowe platformy robotyczne, to nie zawsze można znaleźć właściwą do prowadzenia innowacyjnych badań. Skompletowanie i złożenie podzespołów sprzętowych nie jest wystarczające do rozpoczęcia wartościowych prac badawczych. Na tym etapie, przed twórcami systemu sterowania jest jeszcze długa droga. Należy m.in. stworzyć specyfikację systemu oraz określić, jaką będzie mieć strukturę i jakie zachowania będzie realizował. Jeszcze przed rozpoczęciem implementacji należy wybrać właściwe narzędzia, biblioteki i struktury ramowe. Wybór ten jest szczególnie trudny, gdy brakuje przykładów, na których można się wzorować.

Wychodząc naprzeciw powyższym problemom, w artykule prezentowany jest przykład systemu sterowania robota dwuręcznego, opisany na poziomie dość ogólnym, ale z wyszczególnieniem kwestii szczególnie istotnych. Przedstawiono wykorzystane narzędzia, a także strukturę sprzętową oraz strukturę systemu na poziomie specyfikacji. Artykuł przedstawia obraz złożoności takiego systemu i podsumowuje, jakie elementy taki system powinien zawierać.

2. Zastosowane struktury ramowe i oprogramowanie

2.1. Metoda specyfikacji

Metoda specyfikacji, opisana m.in. w [5], jest dedykowana dla systemów, w których występują roboty. Wyróżnia się różne rodzaje agentów, które komunikują się ze sobą za pomocą międzyagentowych buforów komunikacyjnych. Niektóre agenty mogą być wyposażone w efektory i receptory, stanowiąc tzw. agenty upostaciowione. To ten rodzaj agentów jest z reguły używany do opisywania robotów. W ramach agenta można wyróżnić jego podsystemy: podsystem sterowania, który występuje we wszystkich rodzajach agentów, a także podsystemy związane z efektorami i receptorami. Struktura wewnętrzna agenta i jego podsystemów podlega ograniczeniom, dzięki czemu możliwe jest opisanie w systematyczny sposób różnorodnych systemów robotycznych [18, 19, 23, 2, 3]. Metoda specyfikacji agentowej pozwala na precyzyjne opisanie zachowań poszczególnych agentów i ich podsystemów, a także na opisanie struktury całego systemu. Metoda specyfikacji doczekała się także wariantu opartego na SysML [22].

Podsystemy agenta realizują pewne zachowania, z których każde jest opisane za pomocą tzw. funkcji przejścia. Funkcja przejścia określa, w każdej iteracji, zawartość buforów wyjściowych i pamięci wewnętrznej na podstawie danych w buforach wejściowych i danych w pamięci wewnętrznej z poprzedniej iteracji. Dla każdego zachowania określony jest warunek końcowy i błędu, zaś sposób przełączania zachowań jest określony za pomocą automatu skończonego. Z każdym stanem automatu związane jest jedno zachowanie, a także określone są warunki przejść do kolejnych stanów.

Powyższe założenia, przyjęte dla struktury i zachowania agenta i jego podsystemów, są zarazem proste i pozwalają na specyfikowanie złożonych systemów.

2.2. ROS

ROS (ang. *Robot Operating System*) [9] jest strukturą ramową przeznaczoną do tworzenia oprogramowania związanego z robotyką. ROS jest zbiorem narzędzi, bibliotek i koncepcji, których celem jest ułatwienie tworzenia złożonych systemów dla różnorodnych platform robotycznych. Jednym z założeń ROS jest zachęcenie do prowadzenia wspólnych badań przez ekspertów w różnych dziedzinach robotyki przez umożliwienie wymiany opracowanych rozwiązań. Powyższy cel został osiągnięty – ROS znalazł wiele zastosowań i jest używany w wielu projektach. Obecnie dostępny jest już ROS 2 [7], który stanowi kolejną

Autor korespondujący:
Dawid Seredyński, dawid.seredynski@pw.edu.pl

Artykuł recenzowany
nadesłany 20.09.2021 r., przyjęty do druku 25.11.2021 r.

generację ROS. Niemniej, w niniejszej pracy, skoncentrowano się wyłącznie na pierwszej wersji ROS.

Głównym elementem budulcowym systemów opartych na ROS są tzw. węzły (ang. *nodes*). Węzły są procesami w systemie operacyjnym, które mogą komunikować się ze sobą przez przesyłanie wiadomości i zdalne wywołania procedur (tzw. serwisy). W ROS występują także tzw. akcje, które są podobne do zdalnego wywołania procedury, ale pozwalają na nadzorowanie, przez węzeł wywołujący, stanu wykonywanej procedury, a także na przerwanie jej w dowolnym momencie. Węzły mogą być rozproszone na wielu maszynach, a niskopoziomowa komunikacja między nimi jest realizowana na różne sposoby, w zależności od sytuacji, np. TCP, UDP, komunikacja międzyprocesowa (IPC). Z poziomu węzłów widoczne są tylko sposoby komunikacji dostępne w ROS. Dlatego ROS jest określany także jako middleware. W ramach ROS znajdziemy także biblioteki i narzędzia związane z robotyką, oraz definicje struktur danych dla komunikatów powszechnie używanych w systemach sterowania robotami. Wśród bibliotek i narzędzi wyróżnić można: obliczanie prostej kinematyki (biblioteka *tf*), biblioteki do nawigacji, lokalizacji, budowania mapy środowiska, narzędzia diagnostyczne oraz narzędzia do wizualizacji (biblioteka *rqt*, program *rviz*). W ROS zdefiniowane są także języki i formaty danych, m.in.:

- *Unified Robot Description Format*, URDF – dialekt xml do opisu kinematyki i geometrii robota,
- *Semantic Robot Description Format*, SRDF – dialekt xml pozwalający na wyróżnienie w kinematyce robota takich elementów, jak ramiona, chwytaki, łańcuchy kinematyczne oraz relacje miedzy nimi,
- *ROS message* – język pozwalający na definiowanie złożonych typów danych, które są przesyłane między węzłami w postaci wiadomości i zdalnych wywołań procedur.

ROS jest zintegrowany z popularnymi bibliotekami i narzędziami, m.in. z symulatorem *Gazebo*, biblioteką do przetwarzania obrazów *OpenCV*, biblioteką do przetwarzania chmur punktów *Point Cloud Library* (PCL), oraz ze strukturą ramową do planowania ruchu robotów *MoveIt*, która korzysta m.in. z bibliotek do planowania ruchu *Open Motion Planning Library* (OMPL) i *Search-Based Planning Library* (SBPL). Oprócz dużej liczby bibliotek i narzędzi, ROS to także ogromna liczba dostępnych węzłów tworzonych przez ogólnoświatową społeczność specjalistów w różnych dziedzinach robotyki.

Międzyprocesowe mechanizmy komunikacji w ROS nie są odpowiednie do tworzenia systemów czasu rzeczywistego, gdyż nie zapewniają spełnienia ograniczeń dotyczących opóźnień.

2.3. Orocos

Orocos [15] jest zestawem narzędzi i bibliotek, które stanowią strukturę ramową dedykowaną systemom sterowania robotami. Podstawowymi elementami systemów opartych na Orocos są komponenty, z których każdy realizuje proste obliczenia. Komponent Orocos jest klasą C++, ze ściśle określonym interfejsem. Załadowany komponent (instancja klasy) może być konfigurowany, uruchamiany i zatrzymywany według potrzeb, a także może komunikować się z innymi komponentami m.in. za pomocą tzw. portów. Orocos został stworzony z myślą o systemach czasu rzeczywistego i zawiera mechanizmy pozwalające na tworzenie systemów, w których możliwe jest zachowanie ograniczeń typowych dla takich systemów.

Komponenty Orocos stanowią podstawowe elementy, z których można zbudować złożony system sterowania. Struktura takiego systemu może być dowolna, co może wiązać się z nadmiarowym, niepotrzebnym wzrostem złożoności podczas jego projektowania i implementacji. Istnieje zatem potrzeba dostosowania dowolności łączenia komponentów do metody formalnej specyfikacji, która oferuje prosty i skuteczny zestaw ograniczeń, dzięki którym nawet bardzo złożone systemy mają przejrzystą strukturę.

2.4. FABRIC

FABRIC [13] to struktura ramowa, która integruje formalną metodę specyfikacji opisaną w rozdziale 2.1 z implementacją w ROS i Orocos. FABRIC umożliwia m.in. wygenerowanie kodów źródłowych dla podsystemów wyspecyfikowanych za pomocą dialektu języka xml opartego na metodzie formalnej specyfikacji. W ten sposób tworzone są m.in. funkcje przejścia zbudowane z komponentów Orocos połączonych w sposób określony w specyfikacji. W tymże języku zapisane są też zachowania poszczególnych podsystemów, a także automaty skończone, które nadzorują wykonywanie określonych zachowań oraz przełączają je, uwzględniając ich warunki początkowe, końcowe i błędu. FABRIC zapewnia mechanizmy komunikacji międzyprocesowej, gdyż każdy podsystem jest realizowany jako oddzielny proces. Struktury danych przesyłanych między podsystemami są specyfikowane zgodnie z formatem *ROS message*. Dodatkowo w ramach FABRIC istnieją narzędzia i mechanizmy introspekcji pracy podsystemów, dzięki czemu można np. śledzić bieżący stan podsystemu, bez konieczności ręcznego tworzenia dedykowanych do tego kanałów komunikacyjnych [20]. Podsystemy wygenerowane półautomatycznie za pomocą FABRIC są w pełni kompatybilne z ROS. Korzystają one z serwera parametrów ROS, m.in. w celu konfiguracji komponentów Orocos, a także możliwe jest przesyłanie danych w obu kierunkach, między ROS i podsystemami opracowanymi w FABRIC, za pomocą wiadomości, serwisów i akcji ROS.

Szczególny nacisk w FABRIC jest położony na możliwość implementacji systemu czasu rzeczywistego RT (ang. *real-time*), przy zastosowaniu odpowiedniego systemu operacyjnego i przy właściwej implementacji komponentów Orocos (m.in. wymagany jest brak dynamicznej alokacji pamięci w kodzie RT). Wszystkie wewnętrzne mechanizmy komunikacji zapewniają spełnienie ograniczeń RT, zaś integracja z ROS jest wykonana w taki sposób, aby nie zaburzyć części RT systemu sterowania.

W fazie tworzenia i rozwijania sterownika robota, a także podczas prowadzenia badań, niezbędne są narzędzia pozwalające na wgląd w stan wewnętrzny sterownika, logowanie błędów oraz zapisywanie dużych ilości danych z buforów komunikacyjnych i pamięci wewnętrznej. FABRIC udostępnia powyższe narzędzia do introspekcji podsystemów, bez naruszania ograniczeń RT, dzięki czemu możliwe jest prowadzenie badań na działającym robocie, bez zaburzania jego pracy.

2.5. Gazebo

Gazebo [4] jest powszechnie stosowanym symulatorem oddziaływań fizycznych. Jest dobrze zintegrowany z ROS, dzięki czemu znalazł zastosowanie w robotyce. Gazebo integruje wiele elementów związanych z symulacją systemów robotycznych, m.in.:

- obliczanie oddziaływań fizycznym w symulowanym świecie za pomocą jednej z dostępnych bibliotek (do wyboru: ODE, DART, Bullet, SimBody),
- wizualizacja symulowanego świata w graficznym interfejsie użytkownika, który dodatkowo pozwala na wprowadzanie zmian oraz wywieranie sił na przedmioty,
- reprezentacja świata oraz jego stanu w formacie SDF (ang. *Simulation Description Format*),
- integracja z ROS, w tym możliwość konwersji modeli w formacie URDF do formatu SDF,
- możliwość tworzenia wtyczek rozszerzających funkcjonalność oraz pozwalających na dodawanie nowych modeli czujników i robotów,

– otwarty kod źródłowy z licencją Apache 2.0, która dopuszcza użycie kodu źródłowego zarówno na potrzeby wolnego, jak i własnościowego oprogramowania,
– duża popularność w środowisku robotyków, a zatem mnogość dostępnych przykładów zastosowania, wraz z kodami źródłowymi i modelami robotów.

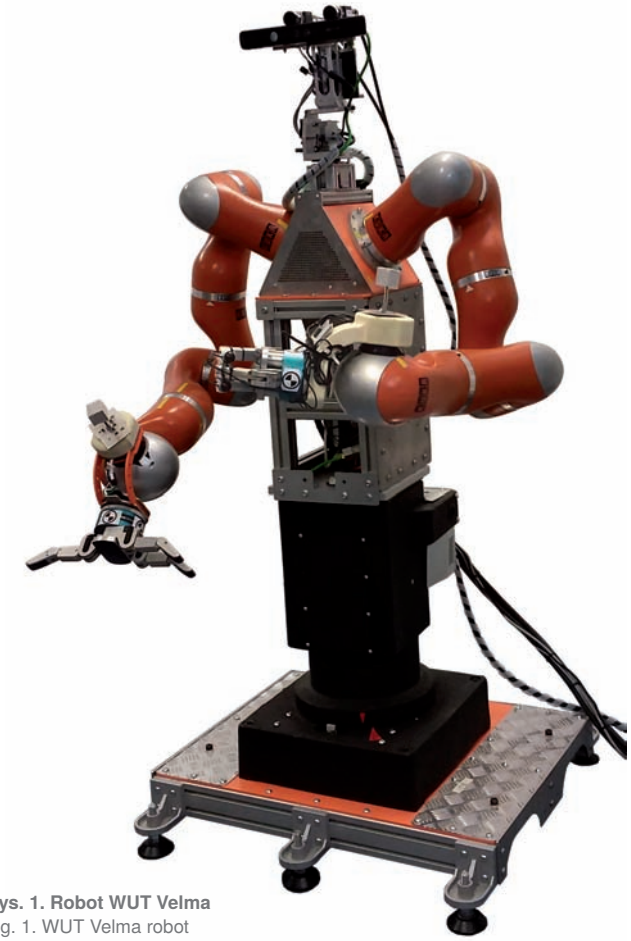
Istotną kwestią jest wybór biblioteki obliczającej oddziaływania fizyczne. W przypadku prostych robotów i prostych scenariuszy ich użycia, pierwszym wyborem jest ODE (ang. *Open Dynamics Engine*) [14], ze względu na dobrą integrację z Gazebo. Jednak w przypadku długich łańcuchów kinematycznych i złożonych zadań, lepszym wyborem jest biblioteka DART (ang. *Dynamic Animation and Robotics Toolkit*) [6]. DART wyróżnia się dokładnością i stabilnością na tle innych bibliotek obliczających oddziaływania fizyczne, dzięki zastosowaniu współrzędnych uogólnionych do reprezentacji układów ciał sztywnych z więzami oraz dzięki zastosowaniu algorytmu Featherstone'a [6]. Z kolei w przypadku ODE, oddziaływania fizyczne są obliczane we współrzędnych kartezjańskich, a spełnienie ograniczeń wynikających z więzów polega na dodaniu wirtualnych impulsów, dzięki którym więzy są zachowane z określoną dokładnością.

W przypadku symulowania robotów manipulacyjnych właściwe jest zastosowanie biblioteki DART, gdyż użycie ODE może powodować duże problemy ze stabilnością symulacji. Kolejną istotną kwestią jest wybranie właściwej biblioteki obliczającej kolizje między obiektami w środowisku. Pierwszym wyborem jest biblioteka FCL (ang. *Flexible Collision Library*) [8], m.in. ze względu na możliwość reprezentacji obiektów za pomocą siatek trójkątów. Pozwala to na wykorzystanie w symulacji obiektów o dowolnym kształcie. O ile wybór ten dobrze współgra z biblioteką ODE, to w przypadku DART pojawiają się problemy ze wzajemnym przenikaniem się obiektów podczas kolizji. Prowadzi to do niepożądanych efektów, m.in. w postaci „przyklejania się" chwyconych przedmiotów do chwytaka, a także do upuszczania chwyconych obiektów, nawet jeśli wykonany chwyt jest stabilny. Z tego powodu warto wybrać metody obliczania kolizji z biblioteki DART, gdyż są one lepiej zintegrowane z funkcjami obliczającymi oddziaływania fizyczne w DART, co przejawia się znacznie bardziej stabilną symulacją: obiekty będące w kolizji nie przenikają się i nie przyklejają się do siebie, a wykonane chwyty są stabilne. Jedyną wadą stosowania DART do obliczania kolizji jest ograniczenie rodzajów brył, z których jest zbudowany robot i obiekty w środowisku, do sfer i prostopadłościanów.

3. Opracowany system

3.1. Sprzęt

W prezentowanym systemie działa jeden robot usługowy WUT Velma[1], który składa się z dwóch ramion wyposażonych w chwytaki, umieszczonych na obrotowej kolumnie, a także z głowy z kamerami umieszczonej na szyi o dwóch stopniach swobody (rys. 1). Ramiona stanowią manipulatory KUKA LWR 4+. Dwa chwytaki BarrettHand różnią się nieco zamontowanym wyposażeniem. Na prawym chwytaku jest sztuczna skóra, czyli matryca czujników ciśnienia, rozmieszczona na wewnętrznej części dłoni oraz na końcach palców, zaś na lewym chwytaku, na koniuszkach palców zamontowano czujniki siły Optoforce, które mierzą kierunek i wartość działającej na nie siły. Na połączeniu końcówek manipulatorów i chwytaków znajdują się 6-osiowe czujniki F/T (skrętnika sił)



Rys. 1. Robot WUT Velma
Fig. 1. WUT Velma robot

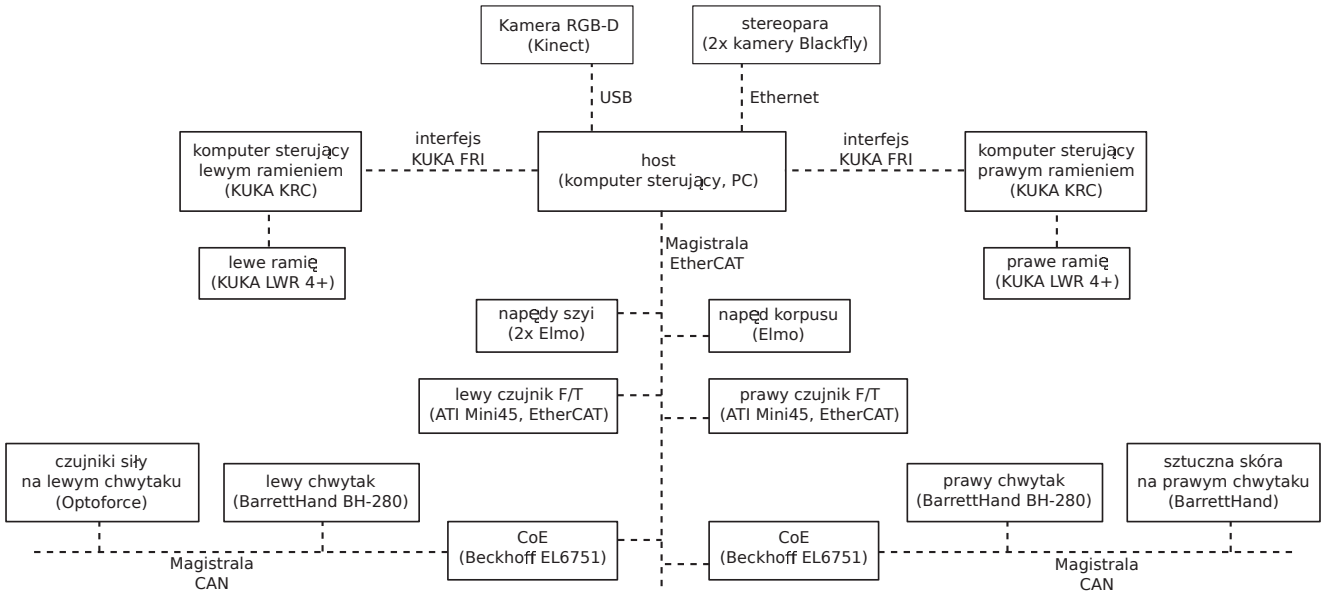
– ATI Mini45 z interfejsem EtherCAT. Obrotowa kolumna, na której są zamontowane ramiona i szyja została zaadoptowana z pierwszego stopnia swobody robota Irp-6. Szyja pozwala na obracanie głowy na boki oraz w kierunkach góra-dół, dzięki zastosowaniu dwóch silników z przekładniami harmonicznymi. Głowa jest wyposażona w kamerę RGB-D (Kinect) oraz w stereoparę kamer RGB.

Liczne urządzenia stanowiące rzeczywiste efektory i receptory robota są podłączone do komputera sterującego (*host*) za pomocą kilku rodzajów interfejsów (rys. 2). Ramiona podłączono za pośrednictwem oryginalnych komputerów sterujących KRC z *host* za pomocą FRI (ang. *Fast Research Interface, KUKA*). Większość pozostałych urządzeń, m.in. sterowniki silników kolumny i szyi, czujniki F/T i chwytaki podłączono do *host* za pomocą magistrali EtherCAT. Dodatkowo w przypadku chwytaków BarrettHand, które obsługują tylko komunikację za pomocą magistrali CAN, konieczne było zastosowanie przejściówki CoE (ang. *CAN over EtherCAT*).

3.2. System sterowania – specyfikacja

System sterowania ma strukturę zależną od zastosowania. W zależności od scenariusza, w systemie można wyróżnić agenty planujące, agenty bazy wiedzy, agenty realizujące oraz nadzorujące wykonanie zadań. Niemniej wyróżnić można zbiór agentów, które są niezależne od realizowanego scenariusza i zadania. Z punktu widzenia niniejszej pracy, najbardziej interesujący jest agent upostaciowiony a_{ex}, który nadzoruje pracę sprzętu (robota WUT Velma) oraz udostępnia pozostałym agentom informacje o bieżącym stanie robota, obserwacje pozyskane z czujników oraz wygodny interfejs do zlecania i nadzorowania poleceń (rys. 3). Drugim ważnym agentem jest agent a_{task} realizujący pewne zadanie według zleconego mu

[1] https://www.robotyka.ia.pw.edu.pl/robots/velma



Rys. 2. Schemat sprzętu robota WUT Velma
Fig. 2. Hardware configuration of WUT Velma robot

planu. Określenie pozostałych agentów w systemie jest zależne od scenariusza, który ma być realizowany przez system.

Agent a_{ex} składa się z podsystemu sterowania c_{ex}, wirtualnego efektora $e_{ex,body}$, wirtualnego receptora $r_{ex,cam}$, a także z rzeczywistych efektorów i receptorów. Należy tutaj podkreślić, że robot może być uruchomiony w świecie symulowanym lub w świecie rzeczywistym. W pierwszym przypadku sprzęt opisany w rozdziale 3.1 nie musi być uruchomiony, ani nawet nie musi być dostępny. System w symulacji można uruchomić na dowolnym komputerze z systemem Linux (nie musi być RT). Natomiast w drugim przypadku, uruchomienie robota w świecie rzeczywistym wymaga, aby sprzęt oraz komputer host, z systemem czasu rzeczywistego, były dostępne.

Rzeczywiste efektory agenta a_{ex} to:
- $E_{ex,armL}$ – lewe ramię,
- $E_{ex,armR}$ – prawe ramię,
- $E_{ex,ec}$ – pozostałe urządzenia na magistrali EtherCAT.

Rzeczywiste receptory to Kinect $R_{ex,kinect}$ i stereopara $R_{ex,stereo}$.

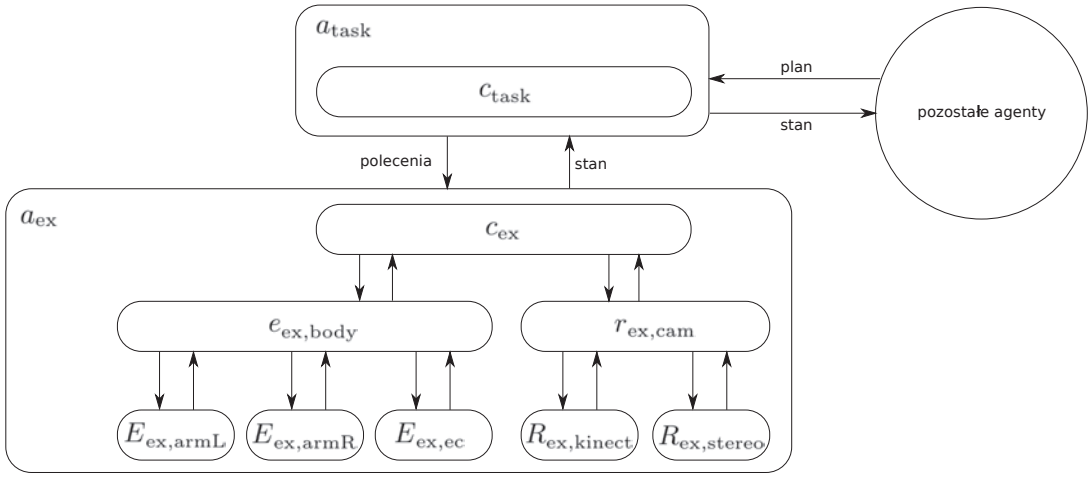
Podsystemy agenta a_{ex} realizują pętlę sterowania o częstotliwości 500 Hz. W każdej iteracji w c_{ex} obliczane jest nowe sterowanie na podstawie bieżącego stanu efektorów oraz wykonywanego w danej chwili polecenia. Złącza ramion oraz złącze obrotowe korpusu sterowane są metodą obliczanego momentu. W przypadku ramion KUKA LWR zadany przez c_{ex} moment jest osiągany dzięki sprzętowemu sterownikowi KUKA, który korzysta z czujników momentów w złączach ramion. W przypadku złącza korpusu, które nie jest wyposażone w czujnik momentu, zadany moment jest przeliczany na prąd, a następnie jest wysyłany do sterownika sprzętowego realizującego regulator prądowy. Przekazanie obliczonych momentów następuje za pośrednictwem podsystemów $e_{ex,body}$, a następnie $E_{ex,ec}$, $E_{ex,armL}$ i $E_{ex,armR}$. Z kolei dwa złącza szyi oraz złącza chwytaka sterowne są pozycyjnie, za pośrednictwem sterowników sprzętowych. W tym przypadku c_{ex} oblicza pozycję zgodnie z zadaną trajektorią i przekazuje je do $e_{ex,body}$, następnie polecenie przekazywane jest dalej do $E_{ex,ec}$.

Obliczanie zadanego momentu w c_{ex} jest procesem złożonym i wieloetapowym. Zostało to szczegółowo opisane w [11]. Zadany moment obliczany jest przez wielopoziomowy, hierarchiczny sterownik, w którym momenty wyliczone na niższym poziomie są rzutowane na przestrzeń zerową wyznaczoną przez sterowniki na wyższym poziomie [10]. Takie podejście pozwala na jednoczesne uwzględnienie wielorakich ograniczeń podczas realizacji zadania. W sterowniku hierarchicznym brane są pod uwagę m.in. ograniczenia ruchu w przestrzeni złącz [16], ograniczenia związane z wewnętrznymi kolizjami między członami robota [1], a także sterowanie impedancyjne w dwóch wariantach [17]. W przypadku sterowania impedancyjnego, moment obliczany jest zgodnie z prawem sterowania impedancyjnego, zatem z dokładnością do komponentu tłumienia, zadana siła jest proporcjonalna do współczynnika sztywności i do odchylenia sterowanego obiektu od punktu równowagi. Zadawanie ruchu polega na zadawaniu położenia punktu równowagi. Szczegółowy opis sterowania impedancyjnego dla robota WUT Velma można znaleźć m.in. w [17].

Dwa warianty sterowania impedancyjnego realizowane są przez dwa zachowania c_{ex}:
- ${}^{c}\mathcal{B}_{ex,CartImp}$ – sterowanie impedancyjne w przestrzeni operacyjnej, które pozwala na zadawanie ruchów obu końcówek, wyrażonych w przestrzeni kartezjańskiej. Parametrami polecenia ruchu w tym przypadku są sztywność i tłumienie, które także wyrażone są w przestrzeni kartezjańskiej,
- ${}^{c}\mathcal{B}_{ex,JntImp}$ – sterowanie impedancyjne w przestrzeni konfiguracyjnej, które pozwala na zadawanie ruchu w przestrzeni konfiguracyjnej, a zatem można zadać położenie i prędkość dla każdego złącza. W tym przypadku, również sztywność i tłumienie wyrażone są w przestrzeni złącz.

Zachowanie ${}^{c}\mathcal{B}_{ex,CartImp}$ jest wykorzystywane do wykonywania niewielkich, precyzyjnych ruchów prowadzących do kontaktu ze środowiskiem, np. chwytanie, otwieranie drzwi, śledzenie konturu, zmywanie. Zachowanie ${}^{c}\mathcal{B}_{ex,JntImp}$ jest wykorzystywane do wykonywania obszernych ruchów, które w znaczny sposób zmieniają konfigurację robota, np. ustawienie ręki przed obiektem, który zostanie chwycony, przygotowanie postury do realizacji zadania otwierania drzwi.

Podstawowe zachowanie $e_{ex,body}$, oznaczone symbolem ${}^{c}\mathcal{B}_{ex,body,transp}$, przekazuje polecenia otrzymane od c_{ex} do rzeczywistych efektorów. W przypadku otrzymania błędnego polecenia, np. zbyt duży zadany moment, $e_{ex,body}$ zaczyna realizować zachowanie ${}^{c}\mathcal{B}_{ex,body,safe}$, w którym przejmuje pełną kontrolę nad rzeczywistymi efektorami i przez pewien określony czas ignoruje polecenia od c_{ex}. Aby powrócić do normalnej pracy, tj. do wykonywania przez $e_{ex,body}$ zachowania ${}^{c}\mathcal{B}_{ex,body,transp}$, podsystem sterowania c_{ex} musi wysyłać poprawne sterowanie oraz wysłać polecenie wyjścia $e_{ex,body}$ z zachowania ${}^{c}\mathcal{B}_{ex,body,safe}$. Dzięki temu można zapobiec sytuacji, w której



Rys. 3. Ogólna struktura systemu sterowania dla robota WUT Velma
Fig. 3. Generic structure of the control system for WUT Velma robot

w wyniku błędu w c_{ex} następuje wygenerowanie niepoprawnego sterowania, mogącego prowadzić do zniszczenia robota lub jego otoczenia. Jest to szczególnie ważne podczas tworzenia implementacji, a także podczas wprowadzania zmian w sterowniku. Podsystem sterowania c_{ex} realizuje bardzo złożone zachowania i nie zawsze jest możliwe wychwycenie wszystkich błędów.

Agent a_{task} realizuje zadanie wg określonego planu, tj. przekształca plan na polecenia zrozumiałe dla a_{ex} oraz nadzoruje rezultaty działań wykonywanych przez a_{ex}. W obszarze na rys. 3 oznaczonym jako „pozostałe agenty" mogą znajdować się, w zależności od potrzeb realizowanego scenariusza, np. agent bazy wiedzy, agenty planujące, agent nadzorujący wykonywanie zadań, agent stanowiący interfejs z człowiekiem, agenty przetwarzające dane wizyjne w zaawansowany sposób.

3.3. System sterowania – implementacja

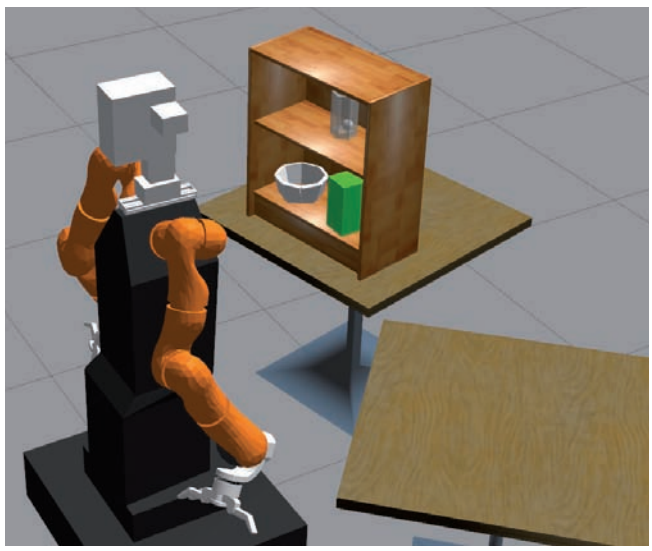
Implementacja systemu sterowania jest wielopoziomowa. W przypadku agenta a_{ex}, jego specyfikacja jest zapisana w formacie właściwym dla FABRIC. Na najniższym poziomie zastosowano komponenty Orocos, które stanowią fragmenty funkcji przejścia dla zachowań. Na wyższym poziomie implementacja jest wykonana w FABRIC przez automatyczną generację kodu na podstawie specyfikacji. Parametry takie jak model kinematyczny robota w formacie URDF, ograniczenia dla ruchu (maksymalne położenia w przestrzeni złącz, maksymalne dopuszczalne momenty) są wczytywane podczas uruchamiania systemu za pośrednictwem serwera parametrów ROS.

W przypadku, kiedy system sterowania steruje rzeczywistym robotem, konieczne jest spełnienie ograniczeń dotyczących częstotliwości pracy i opóźnień. Polecenia obliczane przez c_{ex} muszą być wysłane w określonym czasie do $E_{\text{ex,ec}}$, $E_{\text{ex,armL}}$ i $E_{\text{ex,armR}}$, aby sterowniki sprzętowe otrzymały najnowsze dane na czas. Podsystemy agenta a_{ex} realizują pętlę sterowania 500 Hz i działają one w ramach systemu czasu rzeczywistego. Integracja z ROS, który nie zapewnia spełnienia ograniczeń RT, jest wykonana przez FABIC. Dzięki temu możliwe jest połączenie systemu czasu rzeczywistego, w którym działa pętla sterowania, z pozostałą częścią systemu, nie RT, zaimplementowaną w ROS.

Z kolei, w przypadku systemu sterowania działającego w symulowanym świecie, spełnienie ograniczeń RT nie jest wymagane, gdyż system sterowania jest zsynchronizowany z symulatorem. Na jeden krok symulatora przypada jeden krok pętli sterowania, zatem kolejny krok symulatora jest uruchamiany po otrzymaniu nowych danych z systemu sterowania, a iteracja systemu sterowania zostaje uruchomiona po zakończeniu kroku symulatora. W tym przypadku wszystkie rzeczywiste efektory $E_{\text{ex,ec}}$, $E_{\text{ex,armL}}$ i $E_{\text{ex,armR}}$ i rzeczywiste receptory $R_{\text{ex,kinect}}$ i $R_{\text{ex,stereo}}$ są zaimplementowane jako wtyczki do Gazebo. Należy podkreślić, że wszystkie pozostałe podsystemy agenta a_{ex}, tj. podsystem sterowania i wirtualne efektory i receptor oraz wszystkie inne agenty są identyczne w przypadku systemu działającego w świecie rzeczywistym i systemu działającego w symulacji.

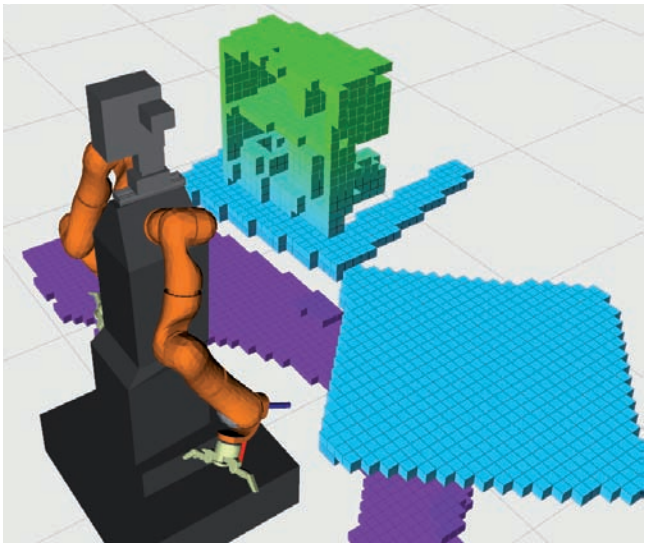
Agent a_{ex} udostępnia swoje bufory komunikacyjne w formie interfejsu akcji ROS. Agent a_{task} może wysyłać polecenia ruchu dla ramion i korpusu w jednym z dwóch dostępnych zachowań ${}^{c}\mathcal{B}_{\text{ex,CartImp}}$ i ${}^{c}\mathcal{B}_{\text{ex,JntImp}}$, a także w obu tych zachowaniach możliwe jest wykonywanie zadanej trajektorii dla szyi i dla chwytaków. Stan robota jest cyklicznie przesyłany od a_{ex} do a_{task} w formie wiadomości (ROS topic).

Symulowany świat można zobrazować za pomocą programu Gazebo Client (rys. 4). Stan robota można wizualizować za pomocą programu ROS rviz (rys. 5), który pozwala też na wyświetlenie różnorodnych danych diagnostycznych (np. model kolizji wewnętrznych w prawie sterowania –

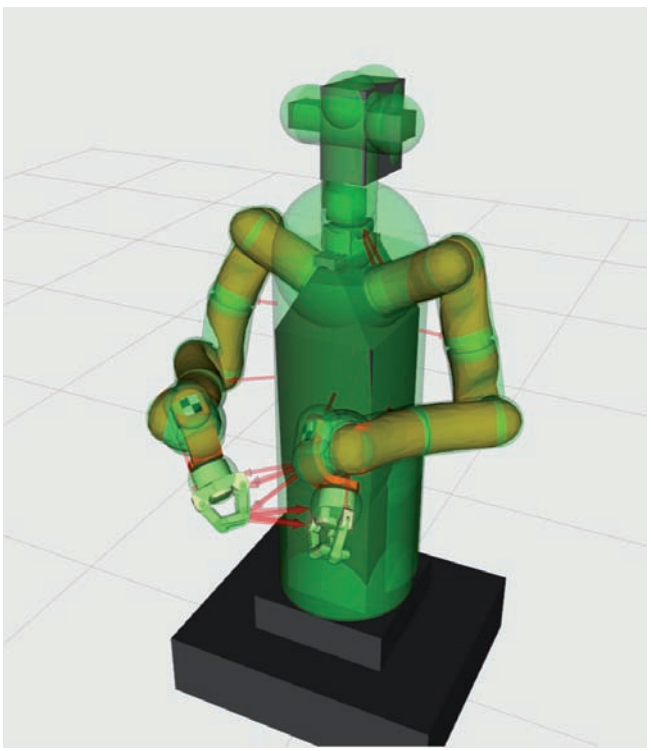


Rys. 4. Widok na symulowany świat w programie Gazebo Client
Fig. 4. A view at the simulated world in Gazebo Client

na rys. 6) oraz obserwacji jakie czyni robot (np. mapa zajętości na rys. 5).



Rys. 5. Wizualizacja stanu robota oraz obserwacji przedstawionych w postaci mapy zajętości (kolorowe woksele) w programie ROS rviz
Fig. 5. Visualization of state of the robot and its observations as an occupancy map (the colourful voxels) in ROS rviz



Rys. 6. Wizualizacja stanu robota w ROS rviz, wraz z wizualizacją modelu kolizji wewnętrznych. Zielone bryły przybliżają geometrię robota, a czerwone strzałki reprezentują miejsca, w których generowana jest wirtualna siła wzajemnie odpychająca pary członów w celu uniknięcia kolizji
Fig. 6. Visualization of state of the robot in ROS rviz with visualization of self-collision model. Green spheres and capsules approximate geometry of the robot and red arrows represent virtual forces that push pairs of links away to avoid self-collisions

4. Podsumowanie

Przedstawiony system sterowania służy do badań dotyczących różnorodnych aspektów sterowania, planowania i wykonywania zadań w robotyce usługowej. Badano zarówno algorytmy sterowania, jak i planowanie geometryczne, w tym planowanie chwytu oraz wykonywanie złożonych zadań wymagających ciągłej interakcji robota ze środowiskiem.

4.1. Zastosowania

Na systemie sterowania o strukturze jak na rys. 3, przeprowadzono liczne badania, udokumentowane w postaci artykułów naukowych. W pracy [17] zbadano wpływ początkowej konfiguracji manipulatora na sukces wykonania zadania chwytania z wykorzystaniem, w końcowej fazie ruchu zachowania ${}^{c}\mathcal{B}_{\mathrm{ex,CartImp}}$. Dla pewnej stałej określonej pozycji chwytaka, ale przy różnych początkowych konfiguracjach manipulatora, zadanie dojścia do obiektu i wykonania chwytu mogło być wykonane lub nie. Badania te uzasadniają konieczność stosowania zachowania ${}^{c}\mathcal{B}_{\mathrm{ex,JntImp}}$, m.in. w celu osiągniecia konfiguracji, która jest właściwa i „wygodna" dla zadań wykonywanych w przestrzeni operacyjnej.

Wykorzystanie zachowania ${}^{c}\mathcal{B}_{\mathrm{ex,CartImp}}$ stanowi duże ułatwienie w zadaniach takich jak otwieranie drzwi [16]. Dzięki możliwości zadawania sztywności w przestrzeni operacyjnej, można precyzyjnie kontrolować ruch robota a także siły, które wywiera na środowisko podczas interakcji, np. otwierania drzwi. Zastosowanie niskiej sztywności pozwala na ograniczenie siły, z jaką chwytak ciągnie za uchwyt, przy jednoczesnym dopasowaniu trajektorii chwytaka do kinematyki drzwi.

W pracy [12] przedstawiono algorytm planowania chwytów w sytuacji, kiedy można przewidzieć siły, które będą wywierane na chwycony obiekt. Chwytanie i manipulacja obiektami zostały zweryfikowane na systemie sterowania rzeczywistym robotem WUT Velma.

W pracy [11] przedstawiono planer lokalny pozwalający na planowanie trajektorii redundantnego robota manipulacyjnego. Metoda planowania oparta jest na modelu, w którym zaimplementowano hierarchiczne prawo sterowania metodą wyliczanego momentu. Działanie planera zostało zweryfikowane m.in. na zadaniu wyciągania przedmiotów z szafki przez robota WUT Velma. Niektóre elementy prawa sterowania wykorzystane w planerze zostały zastosowane w systemie sterowania robota Velma.

Ostatnie prace dotyczyły rozszerzenia robota WUT Velma o bazę mobilną. W symulowanym środowisku prowadzono badania nad mobilną manipulacją [21]. Przedstawiony system sterowania służy także w dydaktyce oraz w prowadzeniu prac dyplomowych jako baza, na której kolejne pokolenia robotyków doskonalą swoje umiejętności.

4.2. Wnioski i dalsze prace

Zaprojektowanie i uruchomienie systemu sterowania robota usługowego, zarówno w symulacji, jak i w rzeczywistości, na którym można prowadzić badania, jest dużym wyzwaniem. Właściwa metoda specyfikacji pozwala na zaprojektowanie systemu realizującego złożone zachowania, zachowując przy tym prostotę i funkcjonalność. Dodatkowym atutem jest możliwość zweryfikowania poprawności specyfikacji. Zastosowanie automatycznej generacji kodu (np. FABRIC) do wykonania części implementacji na podstawie specyfikacji pozwala na uniknięcie licznych błędów wynikających z omylności człowieka. Komponentowe podejście pozwala na zbudowanie złożonego systemu z licznych małych, prostych elementów, z których każdy można przetestować niezależnie od innych. Nieoceniona jest możliwość wykorzystania dostępnych komponentów do budowy własnego systemu (np. ROS, Orocos).

Szczególnie cenna jest możliwość symulowania działania systemu sterowania w sposób możliwie jak najbardziej zbliżony do pracy w rzeczywistym środowisku. Dzięki temu można wychwycić dużo błędów, popełnionych na różnych poziomach, od implementacji prawa sterowania do implementacji algoryt-

mów planujących. Symulacja jest bardzo istotnym elementem zwiększającym bezpieczeństwo pracy z robotami.

W ramach przyszłych prac rozważana jest możliwość migracji na ROS 2. Istotne są także prowadzone obecnie prace nad bezpieczeństwem sterowników robotów.

Bibliografia

1. Dietrich A., Albu-Schäffer A., Hirzinger G., *On continuous null space projections for torque-based, hierarchical, multi--objective manipulation*. 2012 IEEE International Conference on Robotics and Automation (ICRA), 2978–2985, DOI: 10.1109/ICRA.2012.6224571.
2. Kasprzak W., Szynkiewicz W., Stefańczyk M., Dudek W., Figat M., Węgierek M., Seredyński D., Zieliński C., *Agentowa struktura wielomodalnego interfejsu do Narodowej Platformy Cyberbezpieczeństwa*, część 1. „Pomiary Automatyka Robotyka", R. 23, Nr 3, 2019, 41–54, DOI: 10.14313/PAR_233/41.
3. Kasprzak W., Szynkiewicz W., Stefańczyk M., Dudek W., Węgierek M., Seredyński D., Figat M., Zieliński C., Agent--based approach to the design of a multimodal interface for cyber-security event visualisation control. "Bulletin of the Polish Academy of Sciences: Technical Sciences", Vol. 68, No. 5, 2020, 1187–1205, DOI: 10.24425/bpasts.2020.134662.
4. Koenig N., Howard A., *Design and use paradigms for Gazebo, an open-source multi-robot simulator*. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vol. 3, 2004, 2149–2154, DOI: 10.1109/IROS.2004.1389727.
5. Kornuta T., Zieliński C., Winiarski T., *A universal architectural pattern and specification method for robot control system design*. "Bulletin of the Polish Academy of Sciences: Technical Sciences", Vol. 68, No. 1, 2020, 3–29, DOI: 10.24425/bpasts.2020.131827.
6. Lee J., Grey M., Ha S., Kunz T., Jain S., Ye Y., Srinivasa S., Stilman M., Liu K., DART: *Dynamic Animation and Robotics Toolkit*. "The Journal of Open Source Software", Vol. 3, 2018, DOI: 10.21105/joss.00500.
7. Maruyama Y., Kato S., Azumi T., *Exploring the performance of ROS2*. Proceedings of the 13th International Conference on Embedded Software (EMSOFT), 2016, DOI: 10.1145/2968478.2968502.
8. Pan J., Chitta S., Manocha D., FCL: *A general purpose library for collision and proximity queries*. 2012 IEEE International Conference on Robotics and Automation, 3859–3866, 2012, DOI: 10.1109/ICRA.2012.6225337.
9. Quigley M., Conley K., Gerkey B., Faust J., Foote T., Leibs J., Wheeler R., Ng A., *ROS: an open-source Robot Operating System*. ICRA workshop on open source software, Vol. 3, 2009.
10. Sentis L., Khatib O., *Synthesis of whole-body behaviors through hierarchical control of behavioral primitives*. "International Journal of Humanoid Robotics", Vol. 2, No. 4, 2005, 505–518.
11. Seredyński D., Banachowicz K., Winiarski T., *Graph-based potential field for the end-effector control within the torque--based task hierarchy*. 21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR, 2016, 645–650, DOI: 10.1109/MMAR.2016.7575212.
12. Seredyński D., Winiarski T., Banachowicz K., Zieliński C., *Grasp planning taking into account the external wrenches acting on the grasped object*. Kozłowski K. (red), 10th International Workshop on Robot Motion and Control (RoMoCo), 2015, 40–45, DOI: 10.1109/RoMoCo.2015.7219711.
13. Seredyński D., Winiarski T., Zieliński C., *FABRIC: Framework for Agent-Based Robot Control Systems*. Kozłowski K. (red), 12th International Workshop on Robot Motion and Control (RoMoCo), 2019, 215–222, DOI: 10.1109/RoMoCo.2019.8787370.
14. Smith R., *Open Dynamics Engine*. 2005, [www.ode.org].
15. Soetens P., *A Software Framework for Real-Time and Distributed Robot and Machine Control*. Praca doktorska, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium, May 2006, [www.mech.kuleuven.be/dept/resources/docs/soetens.pdf].
16. Winiarski T., Banachowicz K., Seredyński D., *Multi--sensory feedback control in door approaching and opening*. Filev D., Jabłkowski J., Kacprzyk J., Krawczak M., Popchev I., Rutkowski L., Sgurev V., Sotirova E., Szynkarczyk P., Zadrożny S. (eds.), Intelligent Systems'2014, AISC, Vol. 323, 2015, 57–70, DOI: 10.1007/978-3-319-11310-4_6.
17. Winiarski T., Banachowicz K., Seredyński D., *Two mode impedance control of Velma service robot redundant arm*. R. Szewczyk, C. Zielinski, M. Kaliczynska (eds.), Progress in Automation, Robotics and Measuring Techniques. Vol. 2 Robotics., AISC, Vol. 351, 2015, 319–328, DOI: 10.1007/978-3-319-15847-1_31.
18. Winiarski T., Dudek W., Stefańczyk M., Zieliński L., Giełdowski D., Seredyński D., *An intent-based approach for creating assistive robots' control systems*. arXiv preprint arXiv:2005.12106, 2020.
19. Winiarski T., Jarocki S., Seredyński D., *Grasped Object Weight Compensation in Reference to Impedance Controler Robots,* "Energies", Vol. 14, No. 20, 2021, DOI: 10.3390/en14206693.
20. Winiarski T., Seredyński D., *Wizualizacja sterowników robotów bazujących na teorii agenta upostaciowionego*. XV Krajowa Konferencja Robotyki – Postępy robotyki, Vol. 1, 2018, 417–426.
21. Winiarski T., Sikora J., Seredyński D., Dudek W., *DAIMM Simulation Platform for Dual-Arm Impedance Controlled Mobile Manipulation*, 7th International Conference on Automation, Robotics and Applications (ICARA), 2021, 180–184, DOI: 10.1109/ICARA51699.2021.9376462.
22. Winiarski T., Węgierek M., Seredyński D., Dudek W., Banachowicz K., Zielinski C., *EARL – Embodied Agent--Based Robot Control Systems Modelling Language*. "Electronics", Vol. 9, No. 2, 2020, DOI: 10.3390/electronics9020379.
23. 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.

Control System of Two Handed Service Robot

Abstract: This work presents an example control system of a service robot. All used concepts, tools and open source software are described. The control system is presented starting from configuration of hardware, specification, up to its implementation. Generality of the image allows the reader to look at the problem globally, while some important, detailed aspects are highlighted. Simulation–related problems are also described. The presented system of WUT Velma robot has been used in many research works.

Keywords: control system, two handed robot, service robotics, specification, agent

mgr inż. Dawid Seredyński
dawid.seredynski@pw.edu.pl
ORCID: 0000-0003-2528-6335

Jest asystentem w Politechnice Warszawskiej w Instytucie Automatyki i Informatyki Stosowanej. Jego zainteresowania naukowe obejmują planowanie geometryczne i hierarchiczne w robotyce usługowej, a także architektury systemów sterowania robotów..