dr inż. Wojciech Szynkiewicz dr hab. Cezary Zieliński* mgr inż. Krzysztof Kierzenkowski**
*dr hab. Teresa Zielińska** mgr inż. Andrzej Grodecki* prof. dr hab. Anatol Gosiewski**
**Instytut Automatyki i Informatyki Stosowanej Politechniki Warszawskiej*
***Instytut Techniki Lotniczej i Mechaniki Stosowanej Politechniki Warszawskiej*

ŚRODOWISKO PROGRAMOWE DO TWORZENIA STEROWNIKÓW WIELOROBOTOWYCH DLA ZŁOŻONYCH ZASTOSOWAŃ

Streszczenie: W pracy przedstawiono środowisko programowe służące do tworzenia sterowników dla systemów jedno- i wielorobotowych. Sterowniki te mają strukturę otwartą i są konstruowane z modułów wchodzących w skład biblioteki funkcji i programów napisanych w języku C. *Otwarta i modularna struktura układu sterującego oraz zastosowanie maszyn wieloprocesorowych umożliwa realizację złożonych zadań wymagających współpracy wielu robotów oraz dodatkowych urządzeń. Proponowana metoda konstruowania sterowników rozwiązuje problemy: agergacji danych z różnorodnych czujników rzeczywistych, wykorzystania danych w sterowaniu ruchem, koordynacji ruchów robotów oraz komunikacji z operatorem systemu.*

Abstract: The paper presents a programming environment for constructing flexible control systems for a single- and multi-robot systems. The control systems have an open structure and they are constructed out of ready software modules coded in the C *language. An open and modular control structure implemented in a distributed computing architecture enables execution of complex tasks requiring multi-robot cooperation. The proposed method of constructing controllers solves the following problems: aggregation of data obtained from diverse real sensors, utilization of those data in the control algorithm, coordination of multiple robot arms and communication with an operator.*

1. WPROWADZENIE

Nowe obszary zastosowań robotów wraz z rosnącą złożonością wykonywanych przez roboty zadań stawiają coraz wyższe wymagania ich układom sterowania. Tradycyjne sterowniki robotów przemysłowych mają struktury zamknięte bez możliwości modyfikacji, zaś o ich elastyczności i funkcjonalności decyduje uniwersalność ich języków programowania. Dlatego też prowadzone są prace nad definiowaniem i implementacją rozbudowanych języków specjalnie dostosowanych do programowania robotów [1, 7]. Dążenie do uniwersalności sprawia, że specjalizowane języki programowania robotów muszą mieć wszystkie cechy uniwersalnych języków programowania oraz instrukcje specyficzne dla robotów. W efekcie rozbudowane języki specjalizowane są kłopotliwe w implementacji i trudne do nauczenia, a poza tym zestaw instrukcji takiego języka nie może być rozszerzony lub zmodyfikowany przez użytkownika, jeśli pojawi się taka potrzeba. A zatem podstawową wadą języków specjalizowanych jest ich hermetyczność. Konieczność dołączenia do systemu robotowego dodatkowych urządzeń (np. nowych czujników), zazwyczaj wymusza zmiany w definicji języka, a co za tym idzie i translator języka musi ulec zmianie. Nowe czujniki nie tylko wymagają nowych procedur obsługi (*driver'ów*), ale również informacja uzyskana dzięki

nim musi zostać wykorzystana w odpowiedni sposób w procesie sterowania ruchem, więc zmiany mogą być bardzo duże.

Należy zauważyć, że podczas wykonywania przez robota/roboty jednego zadania program sterujący nie ulega zmianie. Ponadto trudno sobie wyobrazić, aby w jednym zadaniu wykorzystywane były wszystkie możliwości rozbudowanego języka programowania robotów. Zazwyczaj jest używany pewien niewielki podzbiór dostępnych instrukcji programowania. W takim przypadku racjonalnym podejściem jest stworzenie sterownika dokładnie „na miarę" aktualnie realizowanego zadania. Naturalną konsekwencją takiego podejścia jest konieczność tworzenia sterownika dla każdego nowego zadania. Jest to ekonomicznie racjonalne tylko wtedy, gdy modyfikowana jest wyłącznie warstwa programowa sterownika, zaś architektura sprzętowa nie ulega zmianom. Przy takiej koncepcji tworzenia sterowników użycie uniwersalnych języków programowania komputerów ma wiele istotnych zalet. Można korzystać ze wszystkich ogólnie dostępnych narzędzi ułatwiających pisanie i uruchamianie programów. Ponadto, w celu uproszczenia pisania takich sterowników, tworzy się biblioteki procedur sterujących robotami [1, 2, 3, 8]. Biblioteki te, zwane również skrótowo językami programowania robotów, umożliwiają konstruowanie systemów robotowych o otwartej strukturze. W przypadku rozbudowy sprzętowej takiego systemu, nowe procedury obsługi oraz sposoby wykorzystania informacji zapisywane są w języku uniwersalnym i dołączane do biblioteki (języka), a translator nie musi być modyfikowany. Jest to metoda o wiele szybsza i tańsza w implementacji niż modyfikacja translatora języka specjalizowanego.

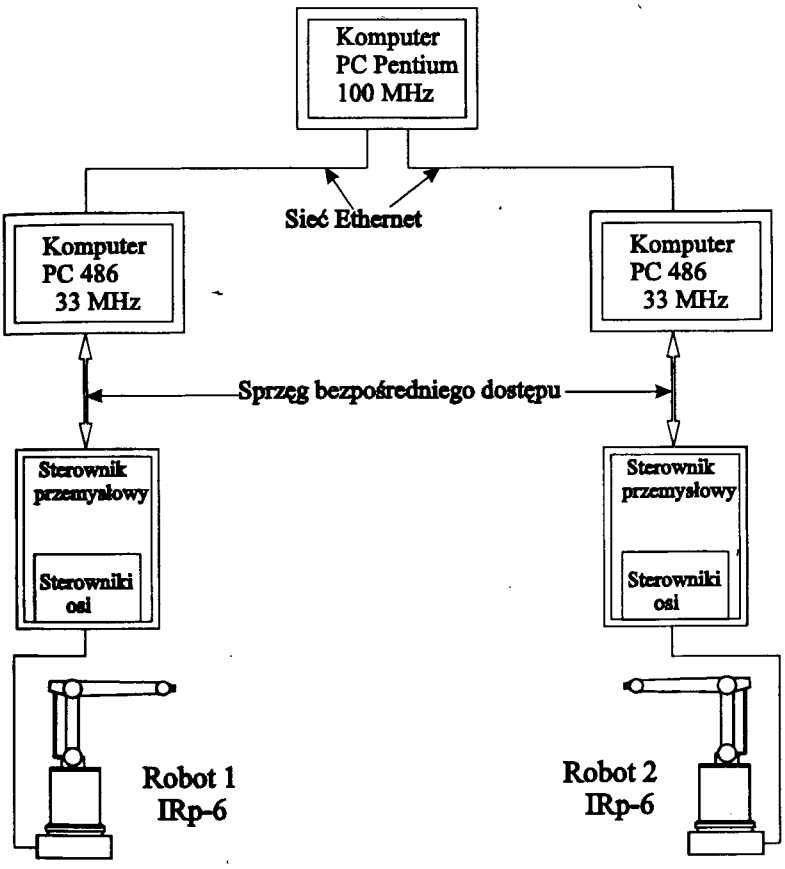
W niniejszej pracy przedstawiono środowisko programowe MRROC (ang. *Multi-Robot Research Oriented Controller*) [5, 9] umożliwiające konstruowanie sterowników zarówno dla systemów jedno- jak też wielorobotowych. Zalety związane z proponowanym sposobem tworzenia zaawansowanych układów sterowania są szczególnie widoczne w złożonych zadaniach wymagających stosowania kilku robotów i współpracujących urządzeń oraz obsługi wielu czujników.

2. OGÓLNA STRUKTURA SYSTEMU MRROC

Proponowane podejście polega na tworzeniu – z gotowych elementów z biblioteki funkcji i procesów napisanych w języku C – sterownika dedykowanego jednemu zadaniu. Przy projektowaniu środowiska MRROC, aby wstępnie nie ograniczać obszaru potencjalnych jego zastosowań założono, iż w ogólnym przypadku rozpatrywany jest system wielorobotowy z nieznaną *a priori* liczbą efektorów (robotów i innych urządzeń). Z tych samych względów przyjęto, że do systemu może być dołączona dowolna liczba oraz rodzaj receptorów (czujników sprzętowych). Aby spełnić powyższe założenia struktura sterownika musi być otwarta i łatwo modyfikowalna.

2.1. Struktura sprzętowa

Przyjęcie struktury otwartej dającej możliwość realizacji różnych, niekiedy bardzo złożonych algorytmów sterowania wymagających dużych nakładów obliczeniowych, niejako narzuca konieczność stosowania systemów wieloprocesorowych. Maszynę taką może stanowić zbiór jednoprocesorowych uniwersalnych komputerów klasy PC połączonych siecią lokalną. Warstwa sprzętowa sterownika MRROC zrealizowanego w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej [5] składa się z oryginalnych sterowników osi, w które fabrycznie są wyposażone roboty IRp-6, komputerów klasy PC połączonych siecią Ethernet oraz specjalizowanych, szybkich sprzęgów równoległych MIAT



Rys. 1. Struktura sprzętowa systemu dwurobotowego

między komputerami i sterownikami przemysłowymi. Konfiguracja sprzętowa (sprzęg: sterownik przemysłowy – komputer) wymaga przypisania pojedynczemu robotowi przynajmniej jednego komputera. Poza tym nie ma żadnych ograniczeń na liczbę komputerów, jeśli zachodzi potrzeba zwiększenia mocy obliczeniowej można dołączyć następny komputer do sieci lokalnej.

Przykładową architekturę sprzętową systemu dwurobotowego przedstawiono na Rys. 1. Moc obliczeniowa takiego systemu komputerowego jest wystarczająca do realizacji wielu zadań wymagających współpracy dwóch robotów, np. przeniesienia wspólnie przez dwa roboty dużych i/lub elastycznych przedmiotów. Pokazana na Rys. 1 realizacja sprzętowa sterownika jest jedną z możliwych. W warunkach zbliżonych do aplikacji przemysłowych lepszym rozwiązaniem może być zastosowanie jednej maszyny wieloprocesorowej. Z punktu widzenia istniejącego oprogramowania systemu MRROC jedynym ograniczeniem jest typ stosowanych procesorów. Muszą być to procesory serii Intel 80xxx lub procesory z nimi kompatybilne. Wynika to z faktu, że aktualnie jedyną, dostępną platformą sprzętową dla systemu operacyjnego QNX są procesory Intel[1].

[1]Przewidywana jest także wersja systemu QNX dla procesorów serii PowerPC.

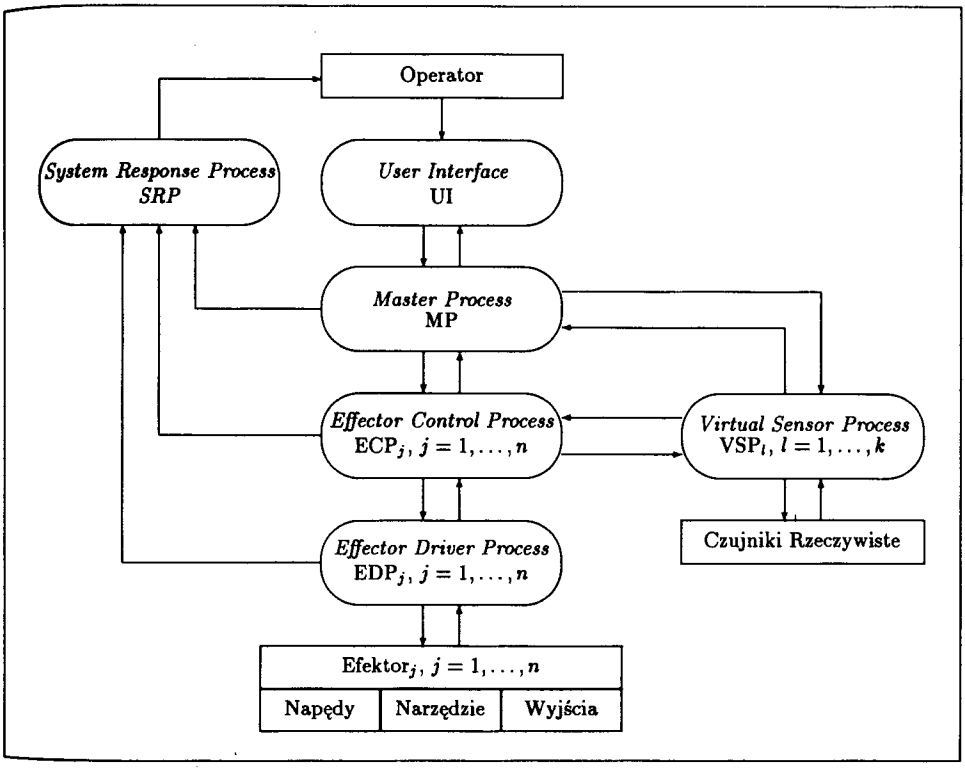
2.2. Oprogramowanie

Wykorzystanie możliwości sprzętu wymaga oprogramowania systemowego, które organizuje pracę komputerów połączonych w sieć i uwalnia programy użytkowe od konieczności bezpośredniej obsługi poszczególnych urządzeń. Dla architektury sprzętowej składającej się z lokalnej sieci komputerów klasy PC, jednym z najlepiej dostosowanych do potrzeb implementacji złożonych struktur sterowania jest – rozproszony system wielozadaniowy czasu rzeczywistego QNX 4.2x [4]. System ten składa się z egzekutora wielozadaniowego i zespołu zadań (procesów) systemowych, współpracujących ze sobą i zadaniami użytkowymi zgodnie z modelem wymiany usług (ang. *client-server model*). System QNX jest zgodny ze standardem POSIX i zawiera skalowalne mikrojądro (poniżej 15 KB), które realizuje wielozadaniowość z wywłaszczaniem, dostarcza mechanizy synchronizacji i komunikacji zadań (sygnały, semafory, kolejki wiadomości). Istnieje także wersja systemu zorientowana na aplikacje wbudowane (ang. *embedded*). Z punktu widzenia implementacji układów sterowania robotami szczególnie istotne jest przewidywalna reakcja (w określonym z góry czasie) na pojawiające się zdarzenia co jest podstawową cechą systemu czasu rzeczywistego.

Modularna struktura i brak prywatnych połączeń między zadaniami użytkowymi umożliwiają elastyczne konfigurowanie systemu stosownie do potrzeb poszczególnych aplikacji. Odwołania zadań użytkowych do zadań systemowych mają postać wiadomości, przekazujących podczas *spotkania* żądanie wykonania określonego zlecenia. Możliwość realizacji spotkania między dowolnymi zadaniami w sieci sprawia, że każde zadanie może korzystać z dowolnych zasobów całej sieci lokalnej.

Warstwę programową sterownika zbudowanego z elementów pakietu MRROC stanowi grupa współbieżnie działających procesów, będących aplikacjami systemu QNX 4.2x. Sterowniki do realizacji konkretnych zadań tworzone są z modułów w postaci funkcji i procesów napisanych w języku C. W celu uproszczenia sposobu tworzenia tego typu sterowników, została napisana obszerna biblioteka gotowych do użycia modułów: funkcji i procesów. Wybierając dowolne elementy z biblioteki, a w przypadku braku odpowiednich elementów modyfikując istniejące lub tworząc całkowicie nowe funkcje i procesy można skonstruować sterownik dedykowany dokładnie danemu zadaniu. A zatem ze strony układu sterowania praktycznie nie ma ograniczeń na rodzaj zadania, które może realizować system robotyczny. Użytkownik piszący sterownik dla danego zadania może skoncentrować się wyłącznie na implementacji własnego algorytmu. W ramach istniejącej ogólnej struktury sterownika tworzy lub modyfikuje gotowe fragmenty odpowiednich programów części modyfikowalnej procesów. Pozostałe funkcje takie jak inicjacja poszczególnych procesów, ich synchronizacja oraz komunikacja międzyprocesowa są realizowane w tych częściach procesów, których użytkownik nie musi modyfikować.

W przypadku systemów złożonych, a takim jest system wielorobotowy, układ sterowania ma często wielowarstwową strukturę hierarchiczną typu *master/slave*. Warstwa nadrzędna – koordynator (*master*) – realizuje cel globalny systemu jako całości, warstwa niższa (*slaves*) realizuje sterowanie poszczególnymi efektorami. Mamy więc tutaj wyraźny podział funkcji, przez co struktura sterownika jest przejrzysta. To zaś z kolei ułatwia diagnostykę potencjalnych błędów i znacznie upraszcza modyfikację oraz rozbudowę sterownika. Cecha ta jest szczególnie cenna przy proponowanej koncepcji tworzenia sterowników, kiedy każda zmiana zadania powoduje rekonstrukcję sterownika. Poszczególne funkcje sterownika MRROC realizowane są przez moduły w postaci odrębnych procesów działających w węzłach sieci lokalnej. Modularność struktury sprawia, że wymiana jednego z elementów systemu nie pociąga za sobą zmiany pozostałych. Ogólna struktura funkcjonalna sterownika została przedstawiona na Rys. 2.



Rys. 2. Hierarchiczna struktura systemu MRROC.

3. MODUŁY STEROWNIKA

Rozproszony sterownik robotowy tworzony w środowisku MRROC ma strukturę hierarchiczną, w której można wyróżnić kilka warstw. Warstwa komunikacji użytkownika z rozproszonym sterownikiem składa się z dwóch modułów: procesu UI (ang. *User Interface*) obsługującego zlecenia operatora oraz procesu SRP (ang. *System Response Process*) wyświetlającego stan systemu. Oba procesy realizują komunikację z użytkownikiem za pośrednictwem okienkowego środowiska graficznego. Użytkownik steruje robotami za pomocą okienkowego menu, poruszając się w ramach skończonej listy dostępnych zleceń. Zakres dostępnych zleceń zależny jest od bieżącego stanu systemu. Proces SRP odbiera komunikaty od pozostałych procesów, które informują o zaistnieniu szczególnej sytuacji (zmiana stanu systemu, informacja o błędzie), formatuje i wyświetla komunikaty na ekranie monitora, przechowuje w buforze nadchodzące komunikaty, tak aby można było prześledzić działanie systemu od chwili jego uruchomienia.

Warstwa nadrzędna, proces MP (ang. *Master Process*), odpowiada za koordynację procesów sterujących efektorami. Proces MP składa się z dwóch podstawowych części: *stałej powłoki* (niemodyfikowalnej części procesu) oraz części *modyfikowanej* przez użytkownika. W części stałej procesu dokonuje się: inicjacja procesu i otwarcie kanałów komunikacyjnych między odpowiednimi procesami.

Struktura oraz funkcje części modyfikowalnej procesu MP, zależą od rodzaju oraz stopnia

złożoności zadania. Rola i funkcje procesu MP uwarunkowane są stopniem współpracy robotów przy realizacji zadania. W przypadku niezależnie działających robotów – działanie procesu MP sprowadza się do uruchomienia procesów sterowania efektorami, a następnie oczekiwania na zlecenia operatora. W przypadku luźnej współpracy robotów proces MP pełni funkcje takie jak uprzednio, a ponadto realizuje synchronizację i komunikację pomiędzy procesami ECP_j. W zadaniach wymagających ścisłej współpracy robotów rola procesu MP jest dominująca, gdyż wykonuje on całość programu sterującego wszystkimi efektorami. Procesy ECP są tylko pośrednikami (między procesem MP i procesami EDP) i umożliwiają jednoczesne przekazywanie zleceń do wielu (*driver'ów*) bez blokowania procesu nadawcy, czyli MP. Ruchy robotów są ściśle koordynowane, co wiąże się z koniecznością częstej synchronizacji procesów sterowania efektorami.

Następna warstwa sterownika składająca się z procesów ECP (ang. *Effector Control Processes*) realizuje algorytmy sterowania poszczególnymi efektorami stosownie do wykonywanego przez system robotowy zadania. Algorytm realizacji zadania implikuje strukturę, funkcje oraz liczbę procesów ECP_j. Ze względu na sposób współpracy robotów wyróżnia się dwie podstawowe struktury procesów ECP: strukturę dla zadań, w których roboty działają całkowicie niezależnie i strukturę dla zadań, w których roboty luźno ze sobą współpracują. W pierwszym przypadku pomiędzy efektorami nie ma żadnych interakcji, więc procesy ECP_j są całkowicie niezależne i realizują programy sterujące poszczególnymi efektorami.

Luźna współpraca wiąże się z synchronizacją robotów w wybranych punktach przestrzeni i chwilach czasowych. Oznacza to synchronizację odpowiednich procesów ECP_j w ściśle określonych miejscach programu sterującego. Procesy ECP_j nie synchronizują się między sobą bezpośrednio lecz przez MP.

Warstwa najniższa odwołuje się bezpośrednio do sprzętu. W jej skład wchodzą procesy obsługi urządzeń (ang. *drivers*) (efektorów, czujników). Procesy EDP (ang. *Effector Driver Processes*) są odpowiedzialne za zarządzanie pracą pojedynczego robota i rozwiązują m.in. proste i odwrotne zadania kinematyki oraz zapewniają bezpośredni dostęp do sprzętu. Są to zatem funkcje charakterystyczne dla danego typu robota, a nie realizowanego zadania.

Procesy VSP (ang. *Virtual Sensor Processes*) realizują odczyt i przetwarzanie danych z czujników rzeczywistych. Środowisko MRROC stwarza możliwość dołączania dowolnej liczby *czujników wirtualnych*. Każdy z tych czujników, zaimplementowany w postaci oddzielnego procesu, może działać w dowolnym węźle sieci. Kilka procesów sterowania efektorami może korzystać z jednego czujnika wirtualnego. Każdy proces ECP lub MP może też korzystać z wielu różnych czujników wirtualnych. Możliwość taką zapewniają specjalizowane procesy, które zarządzają pracą czujników wirtualnych oraz pośredniczą w wymianie danych pomiędzy procesami VSP a procesami ECP i MP. Z czujnikami wirtualnymi możliwe są dwa podstawowe tryby współpracy: interaktywny i nieinteraktywny.

Wyodrębnienie procesów komunikujących ze sprzętem sprawia, że wprowadzenie do systemu nowego efektora (robota) lub czujnika wymaga jedynie stworzenia odpowiedniego programu obsługi (*driver'a*). Pozostałe elementy systemu nie muszą być zmieniane.

4. WYBRANE PRZYKŁADY ZASTOSOWAŃ

Korzystając z mechanizmów udostępnianych przez środowisko MRROC skonstruowano sterowniki do realizacji różnordnych zadań wymagających zarówno ściślej współpracy (wspólne przenoszenie sztywnego obiektu przez dwa roboty), jak też luźnej (odkładanie obiektów na taśmociąg przez jednego robota i zdejmowanie przez) drugiego, przekazywa-

nie sobie nawzajem w przedmiotów przez dwa roboty. Na Rys.3 pokazano dwa roboty IRp-6 wspólnie przenoszące sztywną belkę. W środowisku MRROC stworzono sterownik

Rys. 3. Dwa roboty IRp-6 przenoszące wspólnie belkę.

programowy do realizacji zadania przemieszczenia jednego obiektu przez dwa roboty. Proces MP generuje ściśle skoordynowane trajektorie ruchu dla obu robotów. Kolejne odcinki trajektorii (pojedyncze kroki) są realizowane przez procesy EDP każdego z robotów.

Stworzono też szereg sterowników dla jednego robota: robot grający w warcaby, chwytający przedmioty przemieszczające się na taśmociągu itd. Wykorzystano do realizacji tych zadań różnorodne czujniki od prostych czujników dotykowych i zbliżeniowych poprzez czujnik siły do kamery wizyjnej. Środowisko MRROC umożliwia istotne uproszczenie i radykalne przyspieszenie pisania tego rodzaju specjalizowanych sterowników.

5. WNIOSKI

W pracy zwrócono szczególną uwagę na przedstawienie możliwości wykorzystania środowiska MRROC do złożonych aplikacji w warunkach zbliżonych do przemysłowych, do których realizacji tradycyjne sterowniki są całkowicie nieprzystosowane. Prowadzone są obecnie prace nad rozwojem tego systemu (m.in. zastosowanie programowania obiektowego – język C++) oraz wykorzystaniem go do sterowania innymi robotami np. robotem o strukturze szeregowo-równoległej oraz szybkim robotem odsprzężonym dynamicznie. Należy podkreślić, że istnieje możliwość zaadaptowania tego środowiska do warunków przemysłowych. Wynika to z cech zarówno systemu QNX (mała zajętość pamięci, struktura modularna, możliwość zastosowania w systemach wbudowanych) jak też właściwości samego środowiska MRROC (otwartość i elastyczność, istnienie szeregu gotowych funkcji i procesów, duża łatwość modyfikacji istniejących oraz tworzenia nowych procedur sterowania robotami i innymi urządzeniami). Użytkownik pracujący w środowisku MRROC nie

musi być biegłym programistą. Wystarczą podstawowe wiadomości i ogólna znajomość programowania w języku C, aby móc tworzyć własne aplikacje.

LITERATURA

[1] Blume C., Jakob W.: *PASRO: Pascal for Robots*; Springer-Verlag, Berlin 1985.

[2] Corke P., Kirkham R.: *The ARCL Robot Programming System*; Int. Conf. Robots for Competitive Industries, Brisbane, Australia, 14-16.07 1993, pp.484-493.

[3] Hayward V., Paul R. P.: *Robot Manipulator Control Under Unix RCCL: A Robot Control C Library*; Int. J. Robotics Research, Vol.5, No.4, Winter 1986, pp.94-111.

[4] *QNX System Architecture*; QNX Software Ltd., Canada, 1992.

[5] Szynkiewicz W., Zieliński C., Kierzenkowski K., Zielińska T., Grodecki A.: *Rozproszony sterownik wielorobotowy MRROC*; Materiały V Krajowej konferencji robotyki, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1996, tom 1, str.287-295.

[6] *User's Guide to VAL II: Programming Manual*; Ver.2.0, Unimation Incorporated, A Westinghouse Company, August 1986.

[7] Zieliński C.: *TORBOL: An Object Level Robot Programming Language*; Mechatronics, Vol.1, No.4, Pergamon Press, 1991, pp.469-485.

[8] Zieliński C.: *Robot Programming Methods*; Oficyna Wydawnicza Politechniki Warszawskiej, 1995.

[9] Zieliński C.: *Control of a Multi-Robot System*; 2nd Int. Symp. Methods and Models in Automation and Robotics MMAR'95, Międzyzdroje, 30.08–2.09 1995, pp.603-608.