Języki programowania a programowanie robotów

Istnieje wiele definicji języka programowania. Przykładem może być definicja zaproponowana przez profesora Mordechaja Ben-Ariego.

Język programowania *to zbiór zasad określających, kiedy ciąg symboli tworzy program komputerowy oraz jakie obliczenia opisuje* [I.2].

Rozwijając tę definicję, można powiedzieć, że język programowania to usystematyzowany sposób przekazywania komputerowi poleceń, który pozwala programiście na precyzyjne wskazanie maszynie, jakie dane mają ulec obróbce i jakie czynności należy podjąć w określonych warunkach. Opisanie relacji pomiędzy rzeczywistością a językiem programowania możliwe jest dzięki:

- składni języka (syntaktyce), która określa reguły tworzenia wyrażeń języka z elementarnych symboli;
- semantyce, która wskazuje znaczenie poszczególnych symboli w programie komputerowym;
- pragmatyce, która wyznacza funkcję użytkową języka w interakcji między człowiekiem a maszyną.

Opracowanie języków programowania z jasno sprecyzowanymi regułami znacząco ułatwia komunikację człowieka z maszyną przez możliwość unormowanego manipulowania informacją. Języki programowania można sklasyfikować z uwagi na generacje. Generacja opisuje stopień rozbudowy struktury języka, co jest równoznaczne m.in. z łatwością posługiwania się nim. Poszczególne generacje związane są z rozwojem inżynierii oprogramowania i sprzętu komputerowego. Programy napisane z użyciem starszych generacji języków są szybciej wykonywane przez procesor, ale liczba instrukcji programowych umożliwia wykonanie mniejszej liczby instrukcji przez procesor. Wyższe generacje języków to głównie zwiększenie komfortu programowania i obsługa nawet setek tysięcy instrukcji procesora przy użyciu tylko jednej instrukcji programowej [I.3, I.19]. Wyróżnia się następujące generacje języków programowania [I.3]:

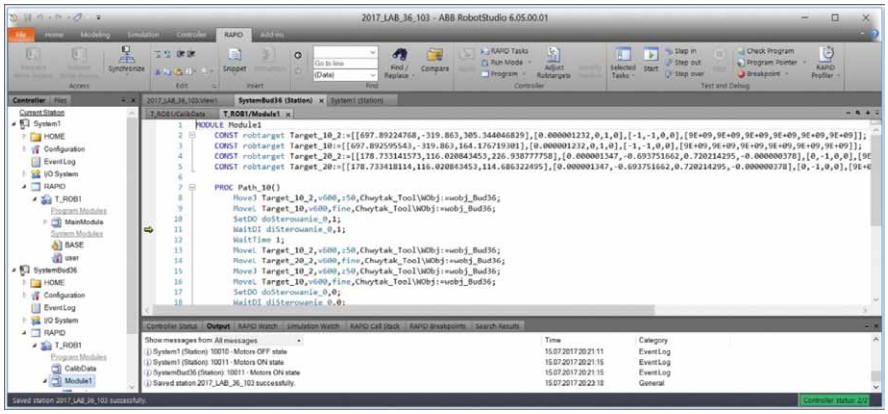
- **Pierwsza generacja języków** (ang. 1GL – *1st Generation Language*) – języki poziomu maszynowego (ang. MML – *Machine Level Languages*). Powstały wraz z narodzinami komputerów i wymagały od programistów pracy na poziomie pojedynczych bitów. Programowanie na poziomie maszynowym sprawiało duże problemy z tworzeniem algorytmów sterujących, co w efekcie generowało dużą liczbę błędów i konieczność ich poprawiania (ang. *debugging*).
- **Druga generacja języków** (ang. 2GL – *2nd Generation Language*) – języki symboliczne określane jako języki niskiego poziomu (ang. *Intermediate Level Languages*). Języki te są zorientowane maszynowo, co oznacza, że są one zależne od architektury danego komputera. Ich używanie jest znacznie prostsze w porównaniu z językami pierwszej generacji i znane są jako języki asemblerowe (systemy nazw mnemonicznych).
- **Trzecia generacja języków** (ang. 3GL – *3rd Generation Language*) – języki wysokiego poziomu (ang. *High Level Languages*) są językami ogólnego przeznaczenia, charakteryzującymi się dużym stopniem uniwersalności. Opracowanie tych języków umożliwiło (niepełne) uniezależnienie się od konkretnego komputera (jeden program, różne kompilatory, różne platformy). Języki tej generacji nazywane są również językami proceduralnymi (z wyodrębnionym zestawem wysokopoziomowych instrukcji) zaprojektowanymi tak, by były łatwiejsze do zrozumienia dla użytkownika (np. języki: C++, Pascal, Java, Delphi).
- **Czwarta generacja języków** (ang. 4GL – *4th Generation Language*) – języki programowania pozwalające – przy użyciu krótkich instrukcji – na utworzenie programu, którego napisanie w językach niższej (np. trzeciej) generacji wymaga użycia setek lub tysięcy razy większej liczby wierszy programu źródłowego. Podstawowym wyróżnikiem języka czwartej generacji jest jego specjalizacja, tworząca z języka efektywne narzędzie w ramach ściśle określonego obszaru zastosowań dzięki wykorzystaniu baz danych, bibliotek, interfejsu użytkownika czy zintegrowanego środowiska programistycznego. Przykładami mogą być tutaj języki arkuszy kalkulacyjnych, systemy zarządzania bazami danych, systemy graficzne itp.
- **Piąta generacja języków** (ang. 5GL – *5th Generation Language*) – inteligentne systemy wiedzy mające graficzny interfejs umożliwiający tworzenie kodu źródłowego (np.: Visual C++). Wygenerowany kod może być przetłumaczony na kod wynikowy za pomocą kompilatorów trzeciej lub czwartej generacji.

Języki programowania robotów zostały opracowane na podstawie istniejących języków programowania popularnych w informatyce. Ich strukturę, w zależności od generacji kontrolerów robotów, można łatwo porównać ze strukturą języków poszczególnych generacji. Obecnie stosowane są w większości języki wysokiego poziomu pochodzące od języków BASIC i Pascal. Aby możliwe było tworzenie wydajnych programów sterujących, języki robotów zostały wyposażone w szereg dodatkowych instrukcji, umożliwiających m.in.: wykonywanie i kontrolowanie ruchów manipulatora w wybranych układach współrzędnych czy komunikowanie się robota z otoczeniem (instrukcje obsługi wejść/wyjść, instrukcje komunikacji itp.).

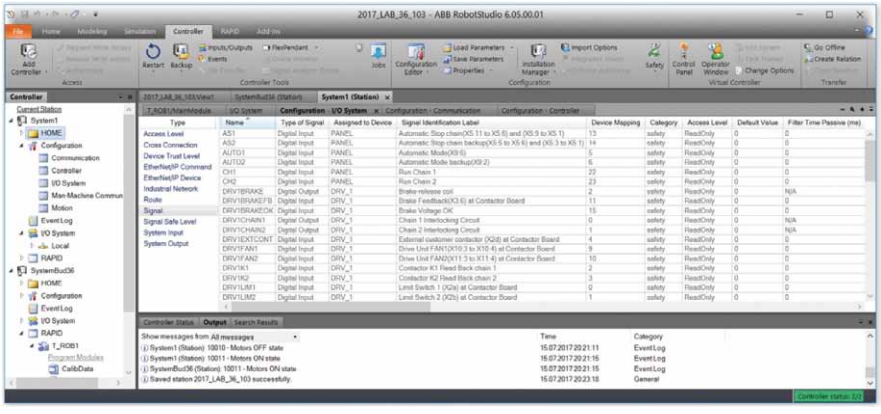
Rys. 1. Ogólny widok panelu nauczania firmy KUKA

W zależności od metody programowania robotów (np. metodą nauczania z wykorzystaniem *teach pendanta* (rys. 1) czy metodą tekstową z wykorzystaniem komputera z edytorem tekstowym), język programowania nawet dla konkretnego robota może się nieco różnić i mieć różnoraką funkcjonalność. Obecnie roboty przemysłowe programowane są często z wykorzystaniem *teach pendanta* bezpośrednio na stanowisku produkcyjnym, co generuje model zachowań i przyzwyczajeń inżynierów. Rozwiązanie takie upraszcza sam proces sterowania i programowania, zastępując skomplikowaną składnię języka zbiorem komend, podlegających parametryzacji, jednak wnosi również pewne ograniczenia co do składni wprowadzanych komend.

Wprowadzenie możliwości programowania robotów przemysłowych za pomocą komputerów pozwoliło na wykorzystanie do programowania zaawansowanych edytorów tekstowych (rys. 2). Poprawiły one w znacznym stopniu komfort pracy programistów i zwiększyły możliwości programistyczne udostępnione przez panele nauczania (rys. 1). Rozszerzeniem możliwości programistycznych jest również zapewnienie dostępu do opcji konfiguracyjnych systemu, które bezpośrednio łączą się z tworzeniem kodów źródłowych, np. umożliwiają definiowanie wejść/wyjść robota (rys. 3).

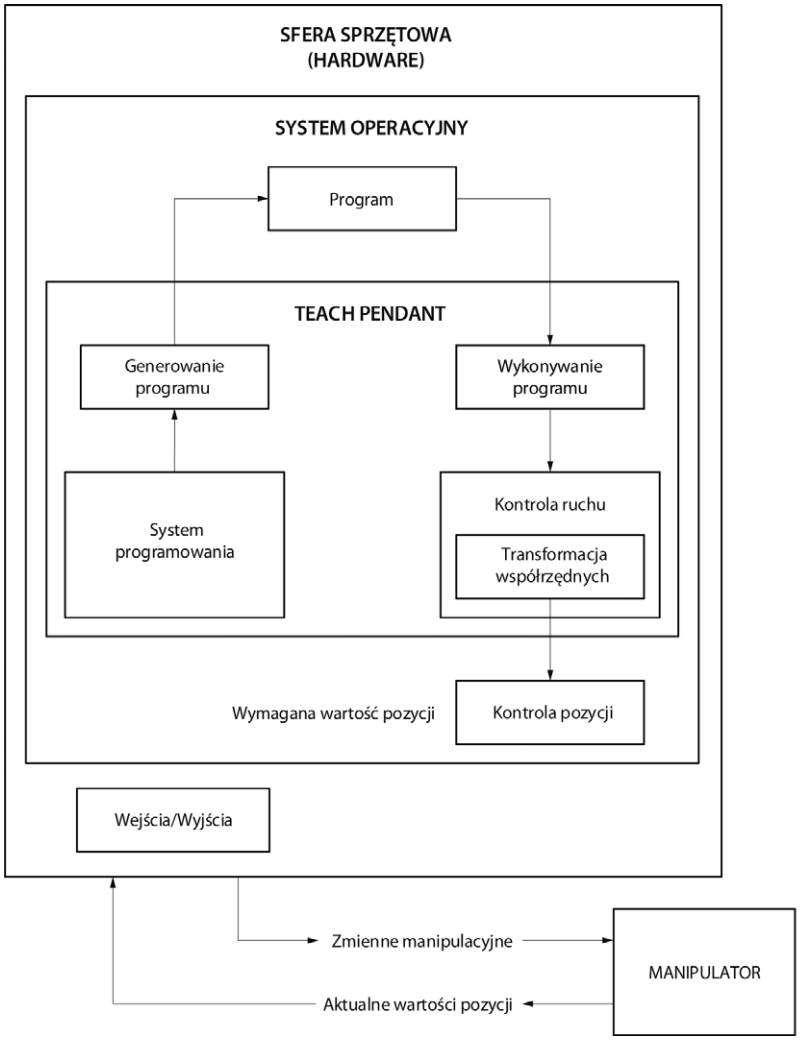


Rys. 2. Ogólny widok okna edytora tekstu w środowisku RobotStudio firmy ABB



Rys. 3. Ogólny widok okna konfiguracji systemu robota w środowisku RobotStudio firmy ABB

Na rysunku 4 przedstawiono podstawową strukturę oprogramowania robota. Umiejscowiony w sferze sprzętowej system operacyjny (np. RobotWare w robotach firmy ABB) umożliwia tworzenie programów sterujących w danym języku programowania (w zależności od firmy). *Teach pendant* jest systemem programowania i pozwala na tworzenie kodów źródłowych. Podczas realizacji programu (np. podczas testowania lub pracy automatycznej) oprócz operacji arytmetycznych i operacji komunikacyjnych związanych z czujnikami i urządzeniami peryferyjnymi, następuje wyznaczenie poszczególnych pozycji układu współrzędnych narzędzia z uwzględnieniem konfiguracji manipulatora. Odpracowanie zadanych wartości położenia i orientacji jest kontrolowane dzięki czujnikom wewnętrznym robota (np. bieżąca informacja o położeniu, prędkości i przyspieszeniu poszczególnych osi manipulatora jest uzyskiwana bezpośrednio lub pośrednio z enkoderów lub resolwerów sprzężonych z aktywnymi osiami urządzenia).



Rys. 4. Podstawowa struktura oprogramowania robota [I.9]

1. Metody programowania robotów

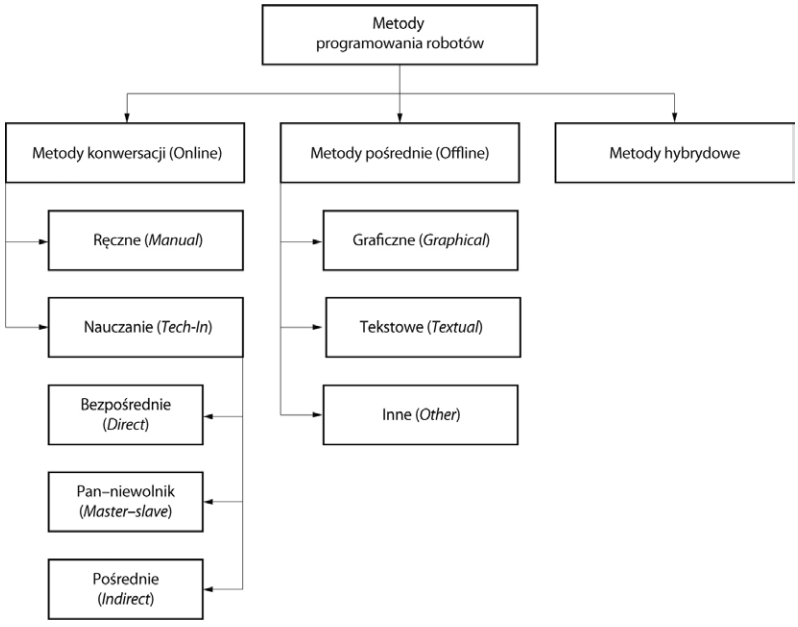
W literaturze można spotkać różne klasyfikacje metod programowania robotów. Najbardziej ogólna wyodrębnia trzy zasadnicze metody, w których kryterium podziału jest konieczność obecności robota podczas procesu programowania (rys. 5).

W tym przypadku można wyróżnić nowy podpunkt czyli [I.9, I.15, II.5]:

- online (nazywane również metodami nietekstowymi lub konwersacji), w których robot jest wymagany do procesu programowania;
- offline – robot nie jest tu wymagany do procesu programowania, który jest realizowany z wykorzystaniem komputerów z edytorami tekstowymi lub graficznymi środowiskami do programowania robotów [I.10];
- hybrydowe – które są połączeniem obu powyższych metod.

1.1. Metody programowania online

Metody konwersacji (ang. *online methods*) umożliwiają liniowe programowanie robotów, przy czym nie



Rys. 5. Metody programowania robotów [I.9]

Rys. 6. Teleoperator (Źródło: https://www.tourmedica.pl/)

pozwalają na bezpośrednią kontrolę stanu czujników. Stanowiły one pierwotny sposób programowania robotów przemysłowych i biorąc pod uwagę realizację trajektorii przez robota – były wystarczające. Metody online obejmują **programowanie ręczne** (ang. *manual*) i tzw. **programowanie przez nauczanie/uczenie** (ang. *teach-in*). Programowanie tymi metodami jest realizowane przez uczenie przy wykorzystaniu dwóch typów ruchu:

- od punktu do punktu (ang. PTP – *Point-to-Point*) – metoda programowania polegająca na ustawieniu manipulatora robota w kolejnych pozycjach trajektorii i zapamiętaniu ich przez kontroler robota po naciśnięciu przez programistę stosownego przycisku na *teach pendancie*; realizacja trajektorii następuje przez przemieszczanie się końcówki manipulatora pomiędzy zaprogramowanymi punktami;
- ciągłego (ang. CP – *Continuous Path*) – metoda programowania polegająca na przemieszczaniu końcówki manipulatora wzdłuż pożądanej trajektorii, przy czym kontroler robota automatycznie (z założoną częstotliwością) zapamiętuje aktualne pozycje; realizacja trajektorii następuje przez przemieszczanie się końcówki manipulatora pomiędzy zaprogramowanymi punktami, jednak w tym przypadku trajektoria jest wiernie odtwarzana z uwagi na dużą liczbę zapisanych punktów.

Programowanie ręczne

Programowanie ręczne (ang. *manual programming*) realizowane jest przez ustawienie statycznych punktów zatrzymania narzędzia, dlatego metoda ta wymaga bezpośredniej pracy ze współrzędnymi robota. Obecnie programowanie tego typu wykorzystywane jest najczęściej do sterowania bezpośredniego (manipulatory, teleoperatory – rys. 6) i jest używane najczęściej do realizacji prostych zadań (np. obsługa maszyn numerycznych).

Do zalet programowania ręcznego można zaliczyć:

- krótki czas programowania;
- brak konieczności używania specjalistycznego komputera;
- możliwość osiągnięcia szybkich ruchów nawet za pomocą prostego kontrolera, ponieważ wykorzystywane są jedynie komendy PTP (*Point-to-Point*).

Wady programowania ręcznego to:

- programowanie wymaga mechanicznej pracy;
- możliwość użycia niewielu punktów;
- brak dodatkowej funkcjonalności.

Programowanie przez nauczanie/uczenie

Programowanie przez nauczanie (ang. *teach-in*) jest programowaniem typu online, a zatem wymaga obecności robota. Obecnie jest to metoda bardzo często wykorzystywana i najczęściej jest realizowana z użyciem panelu nauczania (ang. *teach pendant*). Programowanie przez nauczanie może odbywać się trzema sposobami:

- **Bezpośrednio** (ang. *direct teach-in programming*) – programowanie robota jest realizowane przy wyłączonych hamulcach (ograniczeniach),

a aktualna pozycja końcówki manipulatora jest zapamiętywana poprzez rozkazy (działanie operatora) lub automatycznie z zadaną częstotliwością. Metoda ta jest stosowana zwykle w przypadku lekkich robotów wyposażonych w przekładnie z małym przełożeniem w aplikacjach malowania (rys. 7).

W przypadku robotów o większym udźwigu stosuje się specjalne moduły montowane na kiści robota, wyposażone w czujniki reagujące na oddziaływanie operatora podczas programowania (rys. 8).

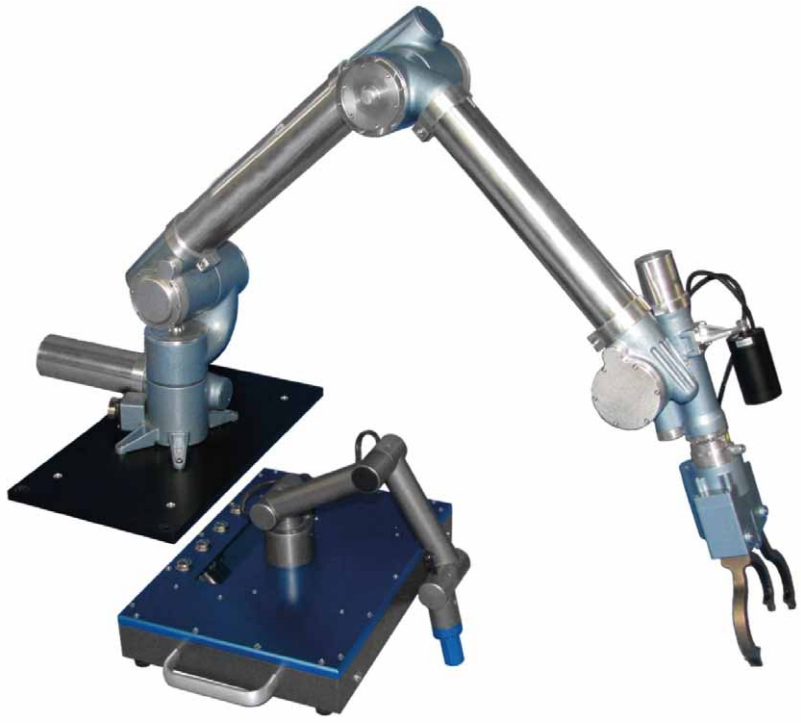
- **„Pan–niewolnik”** (ang. *master–slave programming*) – metoda analogiczna do nauczania bezpośredniego, lecz w tym przypadku operator wykorzystuje model robota rzeczywistego. Ustawiane przez operatora poszczególne konfiguracje modelu są następnie przenoszone za pomocą złożonego oprogramowania do robota docelowego (rys. 9).

Rozwiązanie takie pozwala na sterowanie manipulatorem nawet przez osoby bez specjalnego przeszkolenia. Ograniczenia fizyczne stawiane przez model robota są zgodne z tymi, jakie dotyczą rzeczywistego robota. W związku z tym proces sterowania robotem, jak również programowania (często przez zatwierdzanie położenia modelu manipulatora), jest bardzo intuicyjny.

- **Pośrednio** (ang. *indirect teachin programming*). W tym przypadku generowanie programu i jego zapis do pamięci robota są realizowane do układu sterowania za pomocą przełączników, tablicy programowej lub dedykowanej klawiatury na przenośnym panelu programowania. Ten sposób programowania jest najczęściej używany. Do głównych cech tej metody należy:
  - kontrola trybów pracy robota (tryb pracy automatycznej/ręcznej);
  - ograniczenie prędkości manipulatora do 250 mm/s;
  - pełny dostęp do systemu robota, pozwalający m.in. na wybór układów współrzędnych, dostęp do wszystkich funkcji (np. sterowania stanem wejść/wyjść);
  - możliwość zapisania aktualnej pozycji za pomocą dedykowanego przycisku na panelu nauczania.

Rys. 7. Programowanie bezpośrednie – robot Roberta (Źródło: http://spectrum.ieee.org)

Rys. 8. Konstrukcje modułu dedykowanego do uczenia robotów kolaboracyjnych firmy FANUC (Źródło: FANUC)



Rys. 9. Terabot®-S Manipulator firmy Oceaneering (Źródło: http://www.oceaneering.com)

Do głównych zalet programowania przez nauczanie należy zaliczyć:
- demonstracyjny charakter programowania;
- małe wymagania pamięci tworzonego programu;
- krótki czas tworzenia prostych programów sterujących;
- prosta implementacja.

Za wady programowania tą metodą należy uznać:
- czasochłonność podczas tworzenia dużych, kompleksowych zadań;
- konieczność zatrzymania produkcji podczas programowania robota;
- pełne możliwości testowania programu są możliwe po kompletnym wyposażeniu stanowiska produkcyjnego;
- często niekompletna lub zła dokumentacja programu robota;
- trudności z wykorzystaniem informacji z czujników w programie;
- słabe wsparcie komend manipulowania danymi (trudności w tworzeniu algorytmów z dużą liczbą opcji programowych i obliczeń arytmetycznych).

1.2. Metody programowania offline

Główną wadą programowania online, zwłaszcza dla odbiorców zrobotyzowanych stanowisk produkcyjnych, jest konieczność wykorzystywania rzeczywistego urządzenia, a co za tym idzie – konieczność zatrzymania produkcji. Dlatego też inżynierowie poszukiwali metod, które umożliwiłyby skrócenie czasu uruchamiania nowych stanowisk produkcyjnych oraz modyfikacji oprogramowania na istniejących stanowiskach. Rozwiązaniem okazały się metody offline (ang. *offline methods*), które umożliwiają tworzenie oprogramowania bez konieczności podłączenia rzeczywistego urządzenia. Tworzenie i testowanie aplikacji sterujących w przypadku budowy nowych stanowisk produkcyjnych może być realizowane równolegle z budową stanowisk. Natomiast w przypadku modyfikacji oprogramowania na istniejących stanowiskach większość prac programistycznych może być wykonana przed zatrzymaniem produkcji (wynikiem tego są krótsze przestoje). Programowanie robotów bez konieczności ich wykorzystania można realizować metodą:
- strukturalną, w języku wysokiego poziomu (rys. 2);
- graficzną;

lub innymi metodami.

Rys. 10. Robot IRB 8700 firmy ABB (Źródło: ABB)

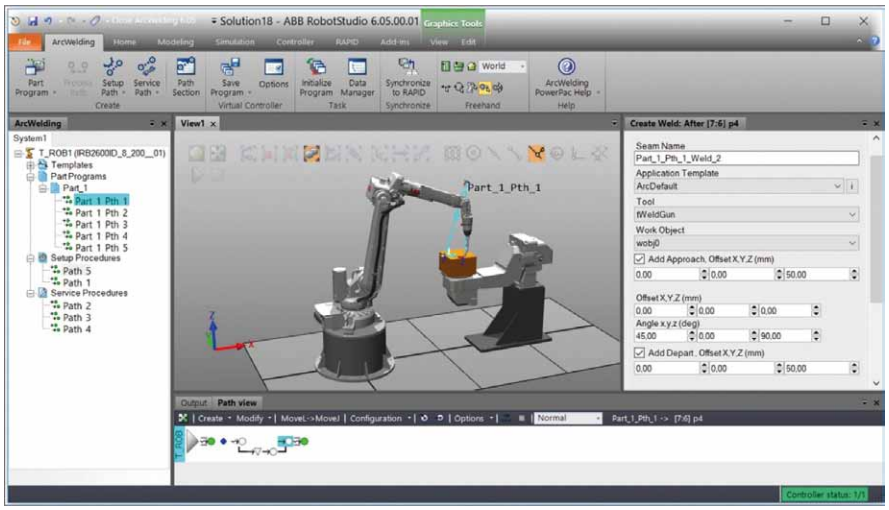
Do głównych zalet programowania offline należy zaliczyć:
- brak konieczności udziału robota;
- możliwość tworzenia złożonych, wielowariantowych programów (również aplikacji wielozadaniowych);
- stosowanie zaawansowanych edytorów tekstowych wspierających programistów podczas tworzenia programów oraz ich modyfikacji;
- łatwość tworzenia dokumentacji programu;
- łatwość wykorzystania informacji pochodzących z czujników.

Główne wady to:
- brak możliwości dokładnego zdefiniowania pozycji (potrzeba dodatkowej kalibracji pozycji przy wykorzystaniu oprogramowania lub przedefiniowania pozycji rzeczywistego robota metodą nauczania z wykorzystaniem *teach pendanta*);
- konieczność weryfikacji pełnego programu sterującego na rzeczywistym stanowisku produkcyjnym.

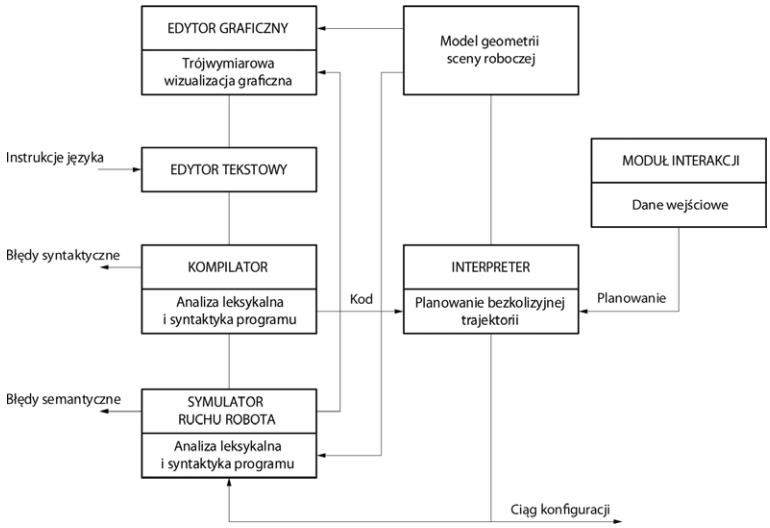
Jedną z metod programowania offline jest programowanie strukturalne nazywane również systemowym programowaniem robotów. Programowanie strukturalne jest wykorzystywane w językach wysokiego poziomu i wymaga rozbudowanego systemu. Do głównych zadań tej metody można zaliczyć:

- implementację programu;
- sprawdzenie programu (składni, poprawności semantycznej);
- testowanie programu;
- przechowywanie i obsługiwanie programu;
- zapewnienie komunikacji pomiędzy użytkownikami;
- ładowanie programów do kontrolera robota i tworzenie wersji zapasowej.

Graficzne metody programowania są używane do tworzenia modeli (programy typu CAx – *Computer-aided technologies*), które są wykorzystywane podczas innych metod programowania/modelowania ruchów robotów. Zabieg tworzenia modeli umożliwia testowanie trajektorii ruchu bez konieczności pracy z rzeczywistymi urządzeniami, np. sprawdzenie kąta natarcia narzędzia. Praca taka ma wiele zalet (np. nie ma konieczności zatrzymywania linii technologicznej w fabryce, aby przetestować nowe oprogramowanie). Można się tutaj odnieść do wirtualnych środowisk do programowania robotów w trybach online/offline (m.in: RobotStudio firmy ABB [I.15, I.16], Roboguide firmy FANUC [I.15, I.25]), które stają się dzisiaj nieodzowne w procesie projektowania zrobotyzowanych stanowisk produkcyjnych w technologii 3D, konfigurowania systemów robotów i tworzenia dla nich oprogramowania [I.5, I.18]. Środowiska tego typu nie tylko pozwalają na tworzenie reprezentacji w technologii 3D na komputerze, ale również umożliwiają prowadzenie symulacji pracy zrobotyzowanych stanowisk produkcyjnych z pełną analizą trajektorii ruchu robotów, cykli pracy i wydajności, tworzenie programów w zaawansowanych edytorach oraz współpracę z robotami w trybie online. Można więc powiedzieć, że środowiska tego typu są wyposażone w mechanizmy umożliwiające stosowanie graficznych metod programowania, zwłaszcza że specjalistyczne dodatki programowe do tych środowisk (np.: ArcWelding PowerPac, Machining PowerPac – RobotStudio – rys. 11) umożliwiają programowanie zadaniowo-symboliczne bez konieczności znajomości przez programistę składania kodu źródłowego w danym języku programowania (np. języku RAPID). Wykorzystanie



Rys. 11. Okno pakietu ArcWelding PowerPac programu RobotStudio firmy ABB



Rys. 12. Struktura programowania robota w języku wysokiego poziomu [I.9]

takich narzędzi, jak CAD to path (generowanie ścieżki na podstawie modelu obiektu importowanego do środowiska graficznego), umożliwia szybkie programowanie trajektorii robota wraz z precyzyjnym orientowaniem narzędzia w każdym punkcie trajektorii.

Z powyższego wynika, że niegdyś wyraźne granice pomiędzy różnymi metodami programistycznymi zacierają się i trudno dziś jednoznacznie wskazać narzędzia przypisane do konkretnej metody. Naukowcy pracują nad **nowymi metodami programowania**. Głównym celem jest wprowadzenie kolejnych uproszczeń i przyspieszenie procesu programowania. I choć większość z tych metod wciąż jest w fazie badań, część z nich jest stosowana już dzisiaj. Do opracowywanych metod można zaliczyć:

- programowanie zorientowane na warsztat;
- programowanie zorientowane na zadanie;
- programowanie w wirtualnej rzeczywistości;
- programowanie wizualne oparte na symbolach i diagramach;
- programowanie hybrydowe (łączenie wielu metod programowania);
- programowanie gestami i ustnymi komendami;
- autonomiczne uczenie się.

W przyszłości przewiduje się utworzenie systemu programowania robota (rys. 12), gdzie planowanie poszczególnych torów ruchu, oparte na podstawie modelu geometrii sceny i kodów instrukcji języka, dokonywane będzie w interpreterze systemu. Interpreter języka zadaniowo zorientowanego programowania robota powinien być zatem wyposażony w system planowania bezkolizyjnych i najkrótszych geometrycznych torów ruchu manipulatora (odpowiadających wszystkim instrukcjom ruchów międzyoperacyjnych typu PTP), jak również w system transformacji, definiowanych funkcyjnie ścieżek kartezjańskich ruchów operacyjnych, w tory ruchu manipulatora w jego wewnętrznym układzie współrzędnych. Jak na razie, wszystkie języki o tego typu interpreterach są w fazie badań eksperymentalnych ze względu na wciąż aktualny problem syntezy efektywnego systemu planowania bezkolizyjnych geometrycznych torów ruchu [I.9].

W wyniku działania interpretatora powinno się uzyskać sekwencję konfiguracji manipulatora, realizującą żądany ruch. Sekwencja taka daje możliwość w dalszej kolejności trójwymiarowej graficznej symulacji ruchu robota na zamodelowanej scenie, co pozwala testować poprawność programu robota w trybie offline i wykrywać jego ewentualne błędy semantyczne (oznaczeń).

1.3. Hybrydowe metody programowania

Mówiąc o programowaniu robotów, nie sposób pominąć rozwiązania związanego z zaawansowanymi zagadnieniami, wynikającymi z zastosowania dedykowanych pakietów dodatkowych, służących do obsługi czujników zewnętrznych czy też systemu wizyjnego. W takich przypadkach bardzo często programowanie punktów i ruchu manipulatora realizowane jest w trybie online, zaś konfiguracja i parametryzacja programu wykonywane są z wykorzystaniem komputera. Część firm stara się poszerzyć funkcjonalność paneli nauczania tak, aby jak najwięcej funkcji mogło być realizowanych na stanowisku bez konieczności dostępu do komputera (system wizyjny iRVision firmy FANUC [I.24, I.26] w całości można skonfigurować, korzystając z panelu Teach Pendant, który ma ekran dotykowy oraz gniazdo USB do podłączenia myszki). Tego typu rozwiązania są jednak rzadko spotykane i nie są tak wygodne, jak skorzystanie z komputera PC z dedykowanym oprogramowaniem. Do głównych zalet programowania hybrydowego należy zaliczyć:

- większą wygodę w związku z większym ekranem komputera w stosunku do ekranu panelu nauczania;
- możliwość korzystania z zaawansowanych funkcji tworzenia części trajektorii (CAD to path);
- dostęp do oprogramowania firm dostarczających systemy wizyjne i inne rozwiązania sprzętowe, mające niezależne oprogramowanie podlegające integracji;
- łatwość tworzenia dokumentacji programu;
- łatwa konfiguracja wszelkiego rodzaju rozwiązań wymagających komunikacji pomiędzy kontrolerem robota a innymi urządzeniami wchodzącymi w skład stanowiska (sterowniki, maszyny CNC, SCADA itp.);
- możliwość korzystania z kodu wymagającego wcześniejszej kompilacji (w przypadku robotów firmy FANUC wykorzystanie kodu zapisanego w Karelu wymaga dokonania jego kompilacji, co może być zrealizowane albo z wykorzystaniem dedykowanego kompilatora, albo z wykorzystaniem kompilatora zaimplementowanego w środowisku Roboguide).

Główne wady to:

- koszty związane z koniecznością zakupu dedykowanego środowiska programistycznego (Robot Studio, Roboguide, Kuka.Sim Pro itp.);
- wymagany zewnętrzny komputer, zazwyczaj z systemem Windows (oprogramowanie większości firm zgodne jest z systemem operacyjnym firmy Microsoft). ■

Bibliografia dostępna pod linkiem:
nis.com.pl/bibliografia.html

Fragment pochodzi z książki:
Programowanie robotów przemysłowych
W. Kaczmarek. J. Panasiuk,
Wydawnictwo Naukowe PWN, 2017