Sterowanie przesunięciem robota do pozycji wyjściowej w trybie pracy automatycznej

Jacek Dunaj
Sieć Badawcza Łukasiewicz – Przemysłowy Instytut Automatyki i Pomiarów PIAP, Al. Jerozolimskie 202, 02-486 Warszawa

Streszczenie: W artykule omówiono sposób rozwiązania problemu związanego z obsługą zrobotyzowanego stanowiska, jak prosto i bezpiecznie doprowadzać manipulator z dowolnego położenia do pozycji wyjściowej bez potrzeby przełączania się w tryb pracy ręcznej. Przedstawiona metoda bazuje na odczycie aktualnego i docelowego położenia każdej osi oraz jej obrocie o minimalny kąt w obu kierunkach wykonywany pod nadzorem operatora. Artykuł opisuje jak wykonano program realizujący to zadanie z wykorzystaniem wyświetlanych na panelu programowania komunikatów i okien dialogowych. Przykładowe algorytmy zapisano w języku KRL programowania robotów Kuka, umożliwiającym generowanie komunikatów użytkownika i okien dialogowych.

Słowa kluczowe: pozycja HOME robota, zmienne systemowe, okna dialogowe, komunikaty użytkownika, programowanie w języku KRL

1. Wprowadzenie

Każdy, kto miał do czynienia z obsługą robota przemysłowego, spotkał się z terminem „pozycja `HOME`". Termin ten określa pewne specyficzne położenie manipulatora dobrane w ten sposób, że:

- jeśli program robota nie jest wykonywany (robot jest wyłączony), to manipulator znajdujący się w pozycji `HOME` nie powinien utrudniać dostępu do samego manipulatora i pozostałych elementów stanowiska robotowego,
- w trakcie wykonywania programu robota pozycja `HOME` jest pozycją wyjściową, od której układ sterowania robota powinien rozpoczynać program i do której manipulator ma zostać przesunięty przed zakończeniem programu. Korzystne jest takie dobranie pozycji `HOME`, aby manipulator w optymalny sposób mógł znaleźć się w punktach roboczych, np. w punkcie podjęcia elementu, przed jego przemieszczeniem.

Dobór pozycji `HOME` jest pewnym kompromisem. Pozycja ta obowiązuje niezależnie od wybranego programu [5, 6], a więc dotyczy każdej aplikacji wykonywanej przez robota. Nie jest jednak pozycją określoną na stałe, ponieważ w zależności od potrzeb może być modyfikowana podobnie, jak położenie dowolnego innego punktu roboczego. Zmiana pozycji `HOME` wykonana podczas edycji jednego programu dotyczy wszystkich innych programów.

Pozycja `HOME` manipulatora nie jest związana z żadnym układem współrzędnych prostokątnych ani z żadnym narzędziem. Jest pozycją zapamiętaną jako położenia (kąty skręcenia) wszystkich osi manipulatora i potencjalnych osi zewnętrznych. Jeśli manipulator robota ma 6 osi, to jego pozycję `HOME` określa 6 liczb, z których każda określa położenie pojedynczej osi.

Artykuł ten koncentruje się na problemie przesunięcia robota do pozycji `HOME` podczas eksploatacji zrobotyzowanego stanowiska oraz przedstawia stosunkowo prosty i bezpieczny sposób związany z programowym uproszczeniem tej czynności. Opis ten odnosi się do aplikacji wykonanych z robotami firmy `Kuka`, ale z pewnymi modyfikacjami przedstawioną metodę można zastosować do robotów innych firm.

Autor korespondujący:
Jacek Dunaj, jdunaj@piap.pl

Artykuł recenzowany
nadesłany 03.04.2019 r., przyjęty do druku 23.09.2019 r.

2. Typ zmiennych dla instrukcji ruchu do pozycji HOME i sposób realizacji ruchu

W języku KRL (ang. ***K**uka **R**obot **L**anguage*) do zdefiniowania położenia osi manipulatora można wykorzystać dwie zmienne typu **`struktura`** [2]:

```
STRUC AXIS REAL A1,A2,A3,A4,A5,A6
STRUC E6AXIS REAL A1,A2,A3,A4,A5,A6,
     E1,E2,E3,E4,E5,E6
```

Oba wymienione typy zmiennych zawierają informacje o położeniu poszczególnych osi manipulatora. Nie ma tutaj odwołania do żadnego układu współrzędnych prostokątnych i do żadnego narzędzia. Typ **`E6AXIS`** jest rozszerzeniem typu **`AXIS`** i odnosi się do przypadku, kiedy robot obsługuje od 1 do 6 dodatkowych osi zewnętrznych związanych np. z torem jezdnym.

Jeśli położenie manipulatora robota zostało zadeklarowane jako:

```
DECL E6AXIS PunktPkt
```

przesunięcie robota do tej pozycji gwarantuje instrukcja:

```
PTP PunktPkt
```

Nie można tutaj wykorzystać dwóch innych dostępnych w języku **`KRL`** instrukcji ruchu – **`LIN`** (ruch po prostej) i **`CIRC`** (ruch po okręgu), ponieważ ich argumentami nie mogą być zmienne typu **`AXIS`** i **`E6AXIS`**.

Podczas wykonywania instrukcji **`PTP`** ruchy wszystkich osi manipulatora są zsynchronizowane – obrót wszystkich osi jest uruchamiany i zatrzymywany w tym samym momencie. Oznacza

to, że tylko oś, która ma wykonać największy obrót (oś wiodąca), jest faktycznie przemieszczana z zaprogramowaną prędkością i przyspieszeniem. Wszystkie pozostałe osie dostosowują swoją prędkość i przyspieszenie do tego, aby osiągnąć końcowy punkt ruchu w tym samym momencie, co oś wiodąca.

Wprowadzając do programu aplikacyjnego instrukcję ruchu **PTP** przy pomocy tzw. formularza jednocześnie określa się jednakowe wartości prędkości i przyspieszeń dla wszystkich osi. W programowaniu w trybie **expert** wartości te można ustalać indywidualnie dla każdej osi, ale podczas realizacji instrukcji **PTP** tylko oś wiodąca będzie poruszała się z tymi parametrami. Prędkość i przyspieszenie są podawane jako wartości procentowe ich maksymalnych wartości zdefiniowanych w danych maszynowych. Dodatkowo w trakcie wykonywania programu prędkość i przyspieszenie realizowane instrukcjami ruchu można zmieniać od 0% do 100% za pomocą panelu programowania.

Pozycja **HOME** w układzie sterowania robota Kuka jest definiowana jako zmienna typu **E6AXIS**, więc przesunięcie manipulatora do tej pozycji możliwe jest tylko przy pomocy instrukcji **PTP HOME**. W podręcznikach programowania robotów Kuka [5, 6] zaleca się, aby każdy program aplikacyjny rozpoczynał i kończył się taką instrukcją. Jest to jednak rozwiązanie na tyle bezpieczne, na ile operatorzy stanowiska robotowego są świadomi skutków uruchomienia programu, gdy manipulator pozostaje w przypadkowym położeniu, ponieważ takie działanie może się zakończyć kolizją. Alternatywnym praktycznym zaleceniem jest to, aby aplikacja robota nie dawała się uruchomić w trybie pracy automatycznej, jeśli manipulator nie znajduje się w położeniu **HOME**. W obu przypadkach rozwiązaniem pozostaje ręczne przesunięcie manipulatora do pozycji **HOME** (lub w sąsiedztwo tej pozycji), co wiąże się z kłopotliwymi przełączeniami trybu pracy i ze znajomością ręcznej obsługi robota, z czym w praktyce różnie bywa.

3. Zmienne systemowe

Układ sterowania robota Kuka i język KRL zapewniają dostęp do tzw. zmiennych systemowych. Zmienne te są dostępne w trybie programowania **expert** i umożliwiają zintegrowanie programu aplikacyjnego robota z informacjami dostępnymi w jego układzie sterowania. Przy ich pomocy możliwy jest m.in. odczyt aktualnej pozycji robota, odczyt aktualnych pozycji każdej z osi, odczyt aktualnych wartości prędkości i przyspieszeń oraz informacji o tym czy robot znajduje się w pozycji **HOME**. W języku KRL nazwy zmiennych systemowych rozpoczynają się od znaku **$**. Pełną listę zmiennych systemowych wraz z ich opisem można znaleźć w dokumentacji [3]. W artykule przedstawiono podstawowe informacje na temat zmiennych zastosowanych w programowej realizacji przesunięcia robota Kuka do pozycji **HOME** w trybie automatyki. Zmiennymi tymi są:

— zmienna **$AXIS_ACT** typu **E6AXIS** zawierająca informację o aktualnym położeniu wszystkich osi manipulatora. Pozwala przyporządkować informację o położeniu osi do dowolnej zmiennej zadeklarowanej w programie jako zmienna typu **E6AXIS**:

```
E6AXIS AxisPosition          ; deklaracja zmiennej
                               AxisPosition
AxisPosition = $AXIS_ACT     ; instrukcja
                               przyporządkowania
```

— zmienna **$H_POS** typu **E6AXIS** zawierająca informację o położeniu wszystkich osi manipulatora znajdującego się w pozycji **HOME**. Pozwala przyporządkować informację o położeniu osi robota znajdującego się w pozycji **HOME** do dowolnej zmiennej zadeklarowanej w programie jako zmienna typu **E6AXIS**:

```
E6AXIS HomePosition          ; deklaracja zmiennej
                               HomePosition
HomePosition = $H_POS        ; instrukcja
                               przyporządkowania
```

— zmienna **$IN_HOME** typu **SIGNAL** (lub **BOOL**) określająca czy manipulator robota znajduje się w pozycji **HOME**:

```
IF ($IN_HOME == TRUE) THEN
   ;  → manipulator robota jest w pozycji HOME
ELSE
   ;  → manipulator robota nie jest w pozycji HOME
ENDIF
```

— zmienna **$VEL_AXIS[6]** – 6-wymiarowa tablica typu **INT**, której kolejne elementy odpowiadają osiom od **1** do **6**. Zawiera informacje o ustawionych prędkościach ruchu poszczególnych osi podczas wykonywania instrukcji **PTP**. Zmienna **$VEL_AXIS[]** służy nie tylko do odczytu prędkości ruchu poszczególnych osi, ale także przy jej pomocy można te prędkości zadawać.

— zmienna **$ACC_AXIS[6]** – 6-wymiarowa tablica typu **INT**, której kolejne elementy odpowiadają osiom od **1** do **6** (indeksowanie tablicy rozpoczyna się od 1). Zawiera ona informacje o ustawionych przyspieszeniach ruchu poszczególnych osi podczas wykonywania instrukcji **PTP**. Zmienna **$ACC_AXIS[]** służy do odczytu przyspieszeń ruchu poszczególnych osi, umożliwia też zadawanie przyspieszeń.

Przykład:

Prosty przykład programowej realizacji obrotu osi **A1** manipulatora – po zakończonym ruchu różnica między aktualnym położeniem tej osi (**AxisPosition.A1)** a jej położeniem w pozycji **HOME** (**HomePosition.A1)** była nie większa niż 1°. Wykorzystano tu dwie zmienne typu **REAL**:

— zmienna **D1** określa wartość kąta, o jaki będzie zmieniana pozycja osi **A1** manipulatora w kolejnych krokach,

— zmienna pomocnicza **Roznica** do wyznaczania różnicy aktualnego położenia osi **A1** i jej położenia w pozycji **HOME**

```
DECL E6AXIS AxisPosition
DECL E6AXIS HomePosition
DECL REAL D1
DECL REAL Roznica
D1 = 1.0
HomePosition = $H_POS  ; odczyt pozycji
                         HOME
LOOP
 AxisPosition = $AXIS_ACT ; odczyt
           aktualnej pozycji manipulatora
 Roznica = HomePosition.A1
         - AxixPosition.A1
 IF (Roznica >= 0.0) THEN
    IF (Roznica > D1) THEN
  AxisPosition.A1 = AxisPosition.A1 + D1
    ELSE
          GOTO KONIEC
    ENDIF
 ELSE
     Roznica = -1.0 * Roznica
     IF (Roznica > D1) THEN
  AxisPosition.A1 = AxisPosition.A1 - D1
     ELSE
        GOTO KONIEC
     ENDIF
 ENDIF
 PTP AxisPosition
ENDLOOP
 KONIEC:
```

Podobny ciąg instrukcji można zastosować dla pozostałych osi **A2**–**A6**. Wówczas przesunięcie manipulatora do pozycji **HOME** jest „łagodniejsze” niż w przypadku zastosowania pojedynczej instrukcji **PTP HOME**, ale nadal grozi kolizją, gdyż obroty poszczególnych osi są realizowane poza bezpośrednią kontrolą operatora. Język KRL do programowania robotów

Kuka zawiera funkcje biblioteczne, które umożliwiają realizację takiej kontroli przez generowanie odpowiednich komunikatów użytkownika (pod pojęciem komunikatów użytkownika należy rozumieć komunikaty generowane przez aplikację, a nie przez układ sterowania) oraz tworzenia okien dialogowych wyświetlanych na panelu programowania robota. Ze względu na pozornie skomplikowany format parametryzacji tych funkcji i ich funkcjonalne ograniczenia są one mało popularne w aplikacjach przemysłowych.

4. Koncepcja programu realizującego przesunięcie robota do pozycji HOME w trybie automatyki

Biorąc pod uwagę ograniczenia związane z generowaniem komunikatów użytkownika i tworzenia okien dialogowych przyjęto kilka założeń dla programu **MoveToHomePosition()** realizującego przesunięcie robota Kuka do pozycji **HOME** w trybie automatyki

- aplikacja uruchamiana w dowolnym trybie pracy (**T1**, **T2**, **Automatyka** lub **Zewnętrzna Automatyka**) nie będzie rozpoczynała się od instrukcji **PTP HOME** ruchu do pozycji **HOME**, jak zalecają to podręczniki programowania robota. Jej uruchomienie nie będzie także blokowane, jeśli robot nie jest w pozycji **HOME**. Zamiast tego będzie realizowana następująca sekwencja instrukcji:

```
WHILE ($IN_HOME == FALSE)
  MoveToHomePosition()
  WAIT SEC 0.1
ENDWHILE
```

  Wywoływany program **MoveToHomePosition()** ma zapewnić przesunięcie manipulatora do pozycji **HOME** sterując czynnościami operatora. Jeśli manipulator nie znajdzie się w pozycji **HOME**, to program ten będzie wywoływany aż do skutku, tj. aż do momentu, gdy wartość zmiennej systemowej **$IN_HOME** będzie równa **TRUE**.
- program **MoveToHomePosition()** zapewnia możliwość sterowanego przez operatora przesunięcia każdej z sześciu osi manipulatora do pozycji **HOME**. Każde okno dialogowe będzie umożliwiało obrót wybranej osi manipulatora w obu kierunkach wyświetlając jednocześnie informację o aktualnym położeniu osi, o jej położeniu w pozycji **HOME** oraz o wartości kąta, o który następuje zmiana pozycji osi. Schemat działania programu przedstawiono w tabeli 1.

Program **MoveToHomePosition()** wykorzystuje siedem okien dialogowych (używanym terminem jest „Menu”, ponieważ język KRL przy tworzeniu okien dialogowych narzuca ograniczenia w długości wprowadzanych tekstów, termin „menu” jest znacznie krótszy niż termin „okno dialogowe”). Każde okno dialogowe zawiera po sześć przycisków.

Przycisk **Key[1]**: opisany w każdym menu jako **PREV MENU**. Powoduje zmniejszenie o 1 numeru okna dialogowego i w konsekwencji wyświetlenie menu odpowiadającego temu numerowi. Jeśli zliczono do 0, to za kolejny numer okna dialogowego przyjmowana jest wartość 7.

Przycisk **Key[2]**: w oknach dialogowych od 1 do 6 jest opisany jako **Di MINUS** (**i** = 1, 2, 3, 4, 5, 6). Powoduje zmniejszenie o 1° wartości kąta, o który będzie przemieszczana oś **i** manipulatora. Wartość kąta **Di** może zmieniać się od 1° do 10°, wartością domyślną (początkową) jest 1°. W przypadku okna dialogowego nr 7 przycisk **Key[2]** opisano jako **OTWORZ** i przypisano mu funkcję potencjalnego wywołania podprogramu otwierającego chwytak.

Przycisk **Key[3]**: w oknach dialogowych od 1 do 6 jest opisany jako **Ai MINUS** (**i** = 1, 2, 3, 4, 5, 6). Powoduje przemieszczenie osi **i** manipulatora o kąt **Di** w kierunku ujemnym, a więc wykonanie ciągu trzech instrukcji:

```
AxisPosition    = $AXIS_ACT
AxisPosition.Ai = AxisPosition.Ai - Di
PTP AxisPosition
```

W przypadku okna dialogowego nr 7 przycisk **Key[3]** opisano jako **ZAMKNIJ** i przypisano mu funkcję potencjalnego wywołania podprogramu zamykającego chwytak.

Przycisk **Key[4]**: w oknach dialogowych od 1 do 6 jest opisany jako **Ai PLUS** (**i** = 1, 2, 3, 4, 5, 6). Powoduje przemieszczenie osi **i** manipulatora o kąt **Di** w kierunku dodatnim, a więc wykonanie ciągu trzech instrukcji:

```
AxisPosition    = $AXIS_ACT
AxisPosition.Ai = AxisPosition.Ai + Di
PTP AxisPosition
```

W przypadku okna dialogowego nr 7 przycisk **Key[4]** opisano jako **HOME**, powoduje wykonanie instrukcji **PTP HOME**, a więc synchroniczny ruch wszystkimi osiami do ich pozycji **HOME** (ruch ten generuje układ sterowania robota, jest to ruch poza kontrolą operatora i może spowodować kolizję).

Przycisk **Key[5]**: w oknach dialogowych od 1 do 6 jest opisany jako **Di PLUS** (**i** = 1, 2, 3, 4, 5, 6). Powoduje zwiększenie o 1° wartości kąta, o który będzie przemieszczana oś **i** manipulatora. Wartość kąta **Di** może zmieniać się od 1° do 10°, wartością domyślną jest 1°. W przypadku okna dialogowego nr 7 przycisk **Key[5]** opisano jako **ZAKONCZ**, jego użycie powoduje zakończenie wykonywania programu **MoveToHomePosition()** i przekazanie sterowania do programu nadrzędnego.

Przycisk **Key[6]**: opisany w każdym menu jako **NEXT MENU**. Powoduje zwiększenie o 1 numeru okna dialogowego i w konsekwencji wyświetlenie menu odpowiadającego temu numerowi. Jeśli zliczono do 8, to za kolejny numer okna dialogowego przyjmowana jest wartość 1.

Tabela 1. Schemat działania programu MoveToHomePosition()
Tab. 1. The scheme of the MoveToHomePosition() program

	Key[1]:	Key[2]:	Key[3]:	Key[4]:	Key[5]:	Key[6]:
Menu 1:	PREV MENU	D1 MINUS	A1 MINUS	A1 PLUS	D1 PLUS	NEXT MENU
Menu 2:	PREV MENU	D2 MINUS	A2 MINUS	A2 PLUS	D2 PLUS	NEXT MENU
Menu 3:	PREV MENU	D3 MINUS	A3 MINUS	A3 PLUS	D3 PLUS	NEXT MENU
Menu 4:	PREV MENU	D4 MINUS	A4 MINUS	A4 PLUS	D4 PLUS	NEXT MENU
Menu 5:	PREV MENU	D5 MINUS	A5 MINUS	A5 PLUS	D5 PLUS	NEXT MENU
Menu 6:	PREV MENU	D6 MINUS	A6 MINUS	A6 PLUS	D6 PLUS	NEXT MENU
Menu 7:	PREV MENU	OTWORZ	ZAMKNIJ	HOME	ZAKONCZ	NEXT MENU

5. Prezentacja komunikatów użytkownika i okien dialogowych na panelach programowania robotów

Roboty KUKA, w zależności od układu sterowania KRC2 i KRC4, współpracują z dwoma różnymi panelami programowania (rys. 1–3). W obu przypadkach kod źródłowy programu aplikacyjnego do generowania komunikatów użytkownika i tworzenia okien dialogowych jest taki sam, różnica wynika ze sposobu ich prezentacji na panelu.

W przypadku panelu współpracującego z układem sterowania KRC2 komunikaty użytkownika są wyświetlane w dolnej części wyświetlacza bezpośrednio nad siedmioma przyciskami funkcyjnymi klawiatury (rys. 1). Ten fragment wyświetlacza służy także do prezentacji okna dialogowego, a opisy przycisków tego okna wyświetlane są w okienkach bezpośrednio nad przyciskami funkcyjnymi (rys. 2). Nie ma więc możliwości, aby na panelu jednocześnie były wyświetlane komunikaty użytkownika i okna dialogowe. Wybór funkcji z okna dialogowego następuje po wciśnięciu odpowiedniego przycisku funkcyjnego.

Rys. 1. Panel programowania robota Kuka dla układu sterowania KRC2 z wyświetlonym komunikatem użytkownika
Fig. 1. Kuka robot programming panel for the KRC2 control system with a user message displayed

Rys. 2. Panel programowania robota Kuka dla układu sterowania KRC2 z wyświetlonym oknem dialogowym do przemieszczania osi A1
Fig. 2. Kuka robot programming panel for the KRC2 control system displayed with a dialog box for moving the A1 axis

W przypadku panelu współpracującego z układem sterowania KRC4 komunikaty użytkownika są wyświetlane w górnej części wyświetlacza bezpośrednio pod paskiem stanu (rys. 3). Okna dialogowe mają podobną postać do okien dialogowych znanych z systemów operacyjnych Windows i są prezentowane w centralnej części wyświetlacza. Na panelu jednocześnie mogą być wyświetlane komunikaty użytkownika i okna dialogowe. Wybór funkcji z okna dialogowego następuje po wskazaniu odpowiedniego obszaru.

Rys. 3. Panel programowania robota Kuka dla układu sterowania KRC4 z wyświetlonym komunikatem użytkownika oraz z oknem dialogowym do przemieszczania osi A1
Fig. 3. Kuka robot programming panel for the KRC4 control system with a user message displayed and a dialog box for moving the A1 axis

6. Programowania komunikatów użytkownika i okien dialogowych – funkcje biblioteczne

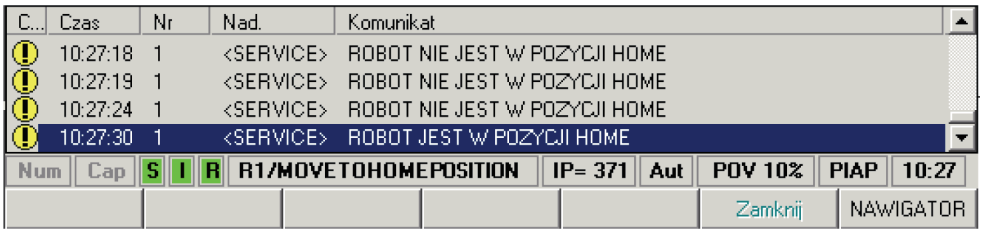
Zasady programowania komunikatów użytkownika i okien dialogowych w języku KRL omawia dokumentacja [3, 4]. W artykule ograniczono się do realizacji programu **`MoveToHomePosition()`**. Wykorzystano tu cztery funkcje biblioteczne:

```
BoolVar = Clear_KrlMsg (clear)
Handle  = Set_KrlMsg (type, name, params[],
                      options)
Handle  = Set_KrlDlg (name, params[],
                      key[], options)
BoolVar = Exists_KrlDlg (Handle, KeyNumber)
```

Pierwsza z nich, **`Clear_KrlMsg()`** uruchomiona z parametrem **`clear = -99`** kasuje wszystkie komunikaty użytkownika utworzone funkcją **`Set_KrlMsg()`**. Potwierdzeniem poprawności jej wykonania jest zwracana zmienna **`BoolVar`** typu **`BOOL`** o wartości **`TRUE`**.

Druga i trzecia funkcja służą odpowiednio do wygenerowania komunikatu użytkownika oraz do utworzenia okna dialogowego. Obie funkcje zwracają wartość zmiennej **`Handle`** typu **`INT`**.

Jeśli zwracana wartość **Handle** jest równa **-1**, komunikat użytkownika nie może zostać wygenerowany (funkcja **Set_KrlMsg()**) lub nie można było utworzyć okna dialogowego (funkcja **Set_KrlDlg()**). Jeśli zwracana wartość **Handle** jest większa od 0, to działanie funkcji zakończyło się powodzeniem (poprawnie wygenerowano komunikat lub utworzono okno dialogowe), a wartości **Handle** można użyć jako parametru dla innych funkcji bibliotecznych.



Rys. 4. Okno komunikatów użytkownika wyświetlane na panelu programowania robota Kuka z układem sterowania KRC2
Fig. 4. User messages window displayed on the Kuka robot programming panel for the **KRC2** control system

Rys. 5. Okno komunikatów użytkownika wyświetlane na panelu programowania robota Kuka z układem sterowania KRC4
Fig. 5. User messages window displayed on the Kuka robot programming panel for the KRC4 control system

Wybranie dowolnego przycisku w oknie dialogowym powoduje jego zamknięcie. Informacji o tym, który przycisk wybrano dostarcza funkcja **Exists_KrlDlg Handle, KeyNumber)**. Jej parametrem wejściowym jest wartość **Handle**, którą funkcja **Set_KrlDlg()** zwróciła tworząc okno dialogowe. Numer przycisku zostaje zwrócony jako parametr wyjściowy **INT KeyNumber**, a potwierdzeniem poprawności wykonania jest wartość **BoolVar** typu **BOOL** równa **TRUE** zwracana przez funkcję.

7. Programowanie komunikatów użytkownika

Program **MoveToHomePosition()** wyświetla komunikaty użytkownika w postaci przedstawionej na rysunkach 4 i 5.

Do programowania komunikatów użytkownika wykorzystywana jest funkcja biblioteczna:

```
Set_KrlMsg(type, name, params[], options)
```

wymagająca wyspecyfikowania czterech parametów:

1. Parametr **type** typu wyliczyniowego **EkrlMsgType**:

```
ENUM EKrlMsgType notify, state, quit,
                 dialog, waiting
```

Określa typ generowanego komunikatu, m.in., czy jest to komunikat użytkownika, czy komunikat o stanie elementów układu sterowania robota, czy komunikat wymaga potwierdzenia etc. W programie **MoveToHomePosition()** wszystkie generowane komunikaty są komunikatami użytkownika, za parametr **type** będzie przyjmowana predefiniowana wartość **#Notify** [3, 4]

2. Parametr **name** typu **KrlMsg_T:**
Jest to parametr typu **struktura** o elementach:

```
STRUC KrlMsg_T CHAR modul[24],
               INT  nr,
               CHAR msg_txt[80]
```

Określa, co ma być wyświetlone w oknie komunikatów na panelu programowania robota. Na panelu programowania współpracującego z układem sterowania KRC2 (rys. 4) okno komunikatów użytkownika podzielono na pięć kolumn:

– kolumna **C...** zawiera ikonę przedstawiającą typ generowanego komunikatu. Wybór ikony następuje na podstawie parametru **type**,
– kolumna **Czas** określa moment wygenerowania komunikatu ustalany na podstawie zegara układu sterowania robota,
– kolumna **Nr** określa numer komunikatu. Wartość ta nie jest nigdzie modyfikowana, ten sam numer może być stosowany wielokrotnie. Zawartość wyświetlana w tej kolumnie zależy od elementu **INT nr** struktury **KrlMsg_T name**,
– w kolumnie **Nad.** w nawiasach <> wyświetlany jest tekst – element **CHAR modul[24]** struktury **KrlMsg_T name**. Maksymalna długość tekstu to 24 znaki,
– w kolumnie **Komunikat** wyświetlany jest tekst – element **CHAR msg_txt[80]** struktury **KrlMsg_T name**. Maksymalna długość tekstu to 80 znaków Do tekstu można wprowadzać wartości do trzech zmiennych.

Na panelu programowania współpracującego z układem sterowania **KRC4** (rys. 5) w oknie komunikatów użytkownika nie ma podziału na kolumny. Jednak wyświetlany komunikat zawiera tę samą informację jak panel programowania układu sterowania **KRC2.**

3. Parametr **params[]** typu **KrlMsg_Par_T:**
Jest to parametr typu **struktura** umożliwiający wprowadzanie do elementu **CHAR msg_txt[80]** struktury **KrlMsg_T name** wartości do trzech różnych zmiennych. Program **MoveToHomePosition()** nie wprowadza do komunikatów użytkownika wartości zmiennych, ale wartości te są wprowadzane do tekstów wyświetlanych w oknach dialogowych. Parametr będzie omówiony w części poświęconej funkcji **Set_KrlDlg()**.
4. Parametr **options** typu **KrlMsgOpt_t:**
Jest to parametr typu **struktura** o elementach:

```
STRUC KrlMsgOpt_T BOOL vl_stop,
                  BOOL clear_p_reset,
                  BOOL clear_p_SAW,
                  BOOL log_to_DB
```

Jej kolejne elementy określają sposób wyświetlania, kasowania lub rejestrowania komunikatów użytkownika i okien dialogowych w zależności od potrzeb:

– **vl_stop** – wartość **TRUE** (wartość domyślna) oznacza, że komunikat użytkownika lub okno dialogowe może zostać wyświetlone zanim robot zakończy wykonywanie poprzedzającej instrukcji ruchu (tryb **advance run**),
– **clear_p_reset** – wartość **TRUE** (wartość domyślna) oznacza, że komunikaty użytkownika będą skasowane w momencie zresetowania programu lub cofnięcia jego wyboru,
– **clear_p_SAW** – wartość **TRUE** (domyślną wartością jest **FALSE**) oznacza, że komunikaty nie będące komunikatami użytkownika, ale wygenerowane przez funkcję biblioteczną **Set_KrlMsg()** zostaną automatycznie usunięte w trakcie zaznaczania bloku instrukcji. Komunikaty użytkownika **#notify** można usuwać tylko za pomocą przycisków **OK** lub **Confirm all** panelu programowania. W przypadku okien dialogowych nie ma możliwości zaznaczania bloków instrukcji – wartość **clear_p_SAW** nie ma znaczenia.

– **log_to_DB** – wartość **TRUE** (domyślną wartością jest **FALSE**) oznacza, że komunikat użytkownika będzie rejestrowany w pliku **.LOG**

W programie **MoveToHomePosition()** komunikaty użytkownika i okna dialogowe są programowane z domyślnymi ustawieniami parametru **Options**:

```
Options = {vl_stop TRUE, clear_p reset TRUE,
           clear_p_SAW FALSE,
           log_to_DB FALSE}
```

Programowanie komunikatów użytkownika pozornie wydaje się skomplikowane, ale praktycznie sprowadza się do kilku instrukcji. Fragment programu wyświetlający informację (rys. 4 i 5) pokazano w rozdziale 9. Podkreśleniem zaznaczono tam teksty, które będą wyświetlone na panelu programowania robota.

8. Programowanie okien dialogowch

Program **MoveToHomePosition()** wykorzystuje okna dialogowe w postaci przedstawionej na rysunkach 6 i 7.

Do programowania okien dialogowych stosowana jest funkcja biblioteczna:

Set_KrlDlg (name, params[], key[], options)

wymagająca określenia czterech parametów.

1. Parametr **name** typu **KrlMsg_T:**

   Parametr typu **struktura** identyczny z parametrem o tej samej nazwie dla funkcji bibliotecznej **Set_KrlMsg()**. Określa, co i w jaki sposób ma być wyświetlone w oknie dialogowym na panelu programowania robota.

   Na panelu współpracującym z układem sterowania KRC2 (rys. 6) okno dialogowe nie jest podzielone na kolumny, ale zawiera całą informację przyporządkowaną poszczególnym elementom struktury **KrlMsg_T name**:

   – **CHAR modul[24]** wyświetlany jest w nawiasach <> po słowie „Źródło”. Maksymalna długość tekstu to 24 znaki. W tej samej linii po słowie „Nr” wyświetlony jest numer okna dialogowego ustalony na podstawie elementu **INT nr** struktury **KrlMsg_T name**

   – pod wierszem ze słowem „Źródło” wyświetlony jest tekst wprowadzony jako element **CHAR msg_txt[80]** struktury **KrlMsg_T name**. Może zawierać wartości zmiennych.

   Na panelu współpracującym z układem sterowania KRC4 (rys. 7) elementy **CHAR modul[24]** i **INT nr** struktury **KrlMsg_T name** są wyprowadzane w tytule okna dialogowego, a tekst wprowadzony jako element **CHAR msg_txt[80]** zostaje wyświetlony nad przyciskami funkcyjnymi.

2. Parametr **params[]** typu **KrlMsgPar_T:**

   Jest to parametr typu **struktura** o elementach:

```
STRUC KrlMsgPar_T KrlMsgParType_T par_type,
                  CHAR            par_txt[26]
                  INT             par_int
                  REAL            par_real
                  BOOL            par_bool
```

gdzie **KrlMsgParType_T** jest typem wyliczeniowym:

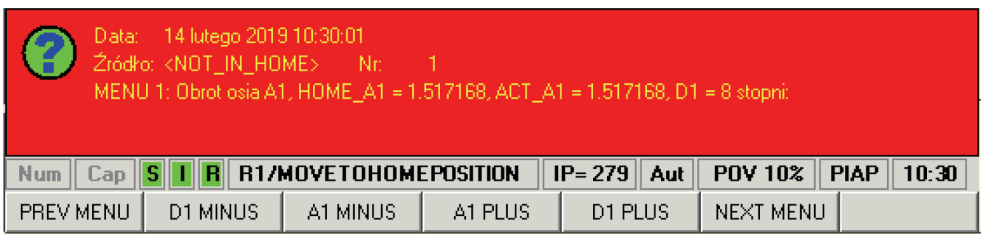
```
ENUM KrlMsgParType_T value, key, empty
```

Do tekstu **CHAR msg_txt[80]** struktury **KrlMsg_T name** można wprowadzać aktualne wartości zmiennych. W oknach dialogowych (rys. 6 i 7) wartościami tych zmiennych są:

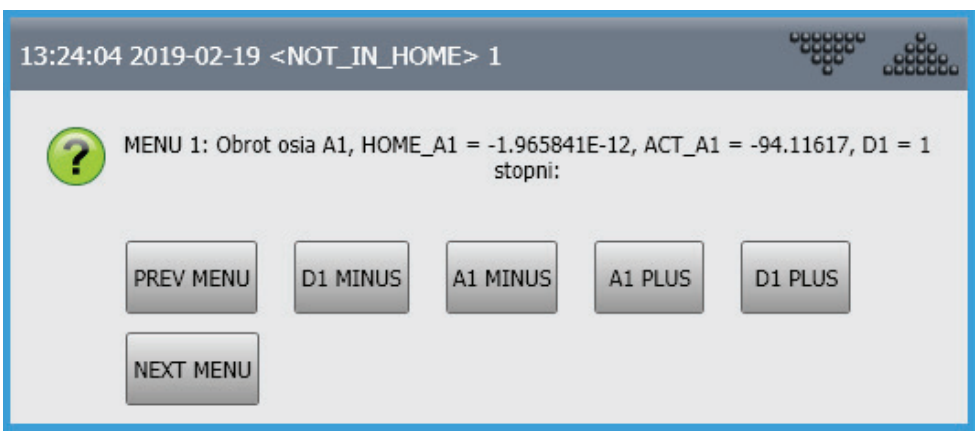
– położenie osi **A1** manipulatora robota w pozycji **HOME** – wartość typu **REAL** wyświetlona za tekstem **HOME_A1 =** ,

– aktualne położenie osi **A1** manipulatora robota – wartość typu **REAL** wyświetlona za tekstem **ACT_A1 =** ,

– aktualna wartość **D1**, o jaką będzie przemieszczana oś **A1** manipulatora robota – wartość typu **INT** wyświetlona za tekstem **D1 =** .



Rys. 6. Okno dialogowe nr 1 (Menu 1) do poruszania osią A1 manipulatora wyświetlane na panelu programowania robota Kuka z układem sterowania KRC2
Fig. 6. Dialog box No. 1 (Menu 1) to move the axis A1 of the manipulator displayed on the programming panel of the robot Kuka with the control system KRC2



Rys. 7. Okno dialogowe nr 1 (Menu 1) do poruszania osią A1 manipulatora wyświetlane na panelu programowania robota Kuka z układem sterowania KRC4
Fig. 7. Dialog box No. 1 (Menu 1) to move the axis A1 of the manipulator displayed on the programming panel of the robot Kuka with the control system KRC4

Wprowadzanie aktualnych wartości zmiennych do elementu **CHAR msg_txt[80]** struktury **KrlMsg_T name** należy poprzedzić wprowadzeniem odnośników do tych zmiennych. Odnośnikami są symbole **%1**, **%2** i **%3** oznaczające odpowiednio kolejne zmienne (maksymalnie można użyć trzech zmiennych). Instrukcja ustalająca wartość parametru **name** dla funkcji **Set_KrlDlg()** będzie miała format jak poniżej (podkreśloną czcionką zaznaczono symbole odwołujące się do zmiennych):

```
name = {modul[]
  „IN_HOME”, Nr 1,
  msg_txt[] „MENU 1:
  Obrot osia A1,
  HOME_A1 = %1,
  ACT_A1 = %2,
  D1 = %3 stopni:”}
```

Do powiązania symbolu odnośnika ze zmienną, a w konsekwencji także do wyboru formatu, w jakim jej wartość będzie wpisania do **CHAR msg_txt[80]** służy parametr **params[]**. Do tekstu maksymalnie można wprowadzić wartości trzech zmiennych – po zadeklarowaniu jako 3-elementowa tablica **Params[3]**:

```
DECL KrlMsgPar_T
Params[3]
```

której kolejny element odpowiada kolejnej zmiennej (numeracja 1–3). Element **Params[i].par_type** struktury określa tzw. typ **i**-tego parametru:

- **#value**: parametr jest wstawiany do tekstu zgodnie z określonym typem zmiennej: text, int, real lub bool,
- **#key**: parametr jest kluczem, którego należy szukać w bazie danych,
- **#empty**: nie przypisano typu parametru.

W programie **MoveToHomePosition()** dla wszystkich zmiennych przyjęto predefiniowaną wartość **value** [3, 4]. Podczas inicjowania struktury **Params[i]** wszystkie pozostałe elementy, tj. **par_txt**, **par_int**, **par_real** oraz **par_bool** są wstępnie oznaczane jako nieokreślone. Instrukcja inicjująca tę strukturę określa, który jej element będzie dalej wykorzystywany i przypisuje mu wartość początkową. Kolejna instrukcja przyporządkowania wpisuje wartość zmiennej do odpowiedniego elementu struktury **Params[i]**. Aby wyświetlić wartości zmiennych **HomePosition.A1** (typ **REAL**), **AxisPosition.A1** (typ **REAL**), oraz **D1** (typ **INT**) należy wykonać następujące instrukcje:

```
Params[1] = {par_type #value, par_real 0.0}
Params[1].par_real = HomePosition.A1
Params[2] = {par_type #value, par_real 0.0}
Params[2].par_real = AxisPosition.A1
Params[3] = {par_type #value, par_int 0}
Params[3].par_int = D1
```

3. Parametr **key[]** typu **KrlMsgDlgSk_t:**

Jest to parametr typu **struktura** o następujących elementach:

```
STRUC KrlMsgDlgSk_t KrlMsgParType_T sk_Type,
                    CHAR  sk_txt[10]
```

gdzie typ danych **KrlMsgParType_T** jest typem wyliczeniowym:

```
ENUM KrlMsgParType_T value, key, empty
```

Parametr ten definiuje przyciski funkcyjne okna dialogowego. Programując okna dialogowe w języku KRL należy pamiętać, że maksymalnie mogą one zawierać siedem przycisków funkcyjnych. Parametr **key[]** należy zadeklarować jako 7-elementową tablicę **Key[7]**:

```
DECL KrlMsgDlgSk_t Key[7]
```

której elementy odpowiadają kolejnym przyciskom funkcyjnym (1–7). Element **Key[i].sk_Type** struktury określa tzw. typ etykiety **i**-tego przycisku funkcyjnego, któremu można przyporządkować jedną z trzech predefiniowanych wartości:

- **#value**: **sk_txt[]** odpowiada etykiecie przycisku,
- **#key**: **sk_txt[]** to klucz bazy danych zawierający etykietę przycisku,
- **#empty**: nie przypisano typu etykiety.

W programie **MoveToHomePosition()** dla wszystkich przycisków funkcyjnych przyjęto predefiniowaną wartość **value** [3, 4]. Drugi element **Key[i].sk_txt[10]** struktury zawiera opis **i**-tego przycisku funkcyjnego. Jeśli wprowadzono tam „pusty" tekst, tj. tekst zawierający same spacje, to przycisk ten nie pojawi się w oknie dialogowym.

4. Parametr **options** typu **KrlMsgOpt_t:**

Parametr ten omówiono w części poświęconej funkcji bibliotecznej **Set_KrlMsg().**

Jak w języku KRL można zaprogramować przedstawione na rysunkach 6 i 7 okno dialogowe nr 1 (Menu 1) programu **MoveToHomePosition()** pokazano w punkcie 6 rozdziału 9. Identycznie programuje się pozostałe okna dialogowe 2–6. W oknie dialogowym nr 7 nie są wyświetlane wartości żadnych zmiennych, więc inicjując parametry funkcji **Set_KrlDlg()** można pominąć ustawienia parameteru **params** (punkt 8 rozdziału 9).

9. Struktura programu MoveToHomePosition()

Poniżej zawarto fragment programu **MoveToHomePosition()**. Pominięto fragmenty, w których kod źródłowy powtarzał się (np. tworzenie kolejnych okien dialogowych):

1. Początek programu: Deklaracje używanych zmiennych:

```
GLOBAL DEF  MoveToHomePosition( )
   DECL INT  i
   DECL INT  KeyNumber    ; numer wybranego
                            przycisku
   DECL INT  MenuNumber   ; numer okna
                            dialogowego
   DECL INT  D1, D2, D3, D4, D5, D6    ;
                            przyrosty katow
                            osi
; Zmienne do zapamiętywania predkosci
                          i przyspieszen osi:
   DECL INT  STORE_VEL_AXIS [6]
   DECL INT  STORE_ACC_AXIS [6]
    ; Zmienne do pamiętania pozycji
  poszczegolnych osi
   DECL E6AXIS  AxisPosition
   DECL E6AXIS  HomePosition
; Zmienne do tworzenia komunikatow i okien
                           dialogowych
   DECL INT             Handle
   DECL  KrlMsg_t       name
   DECL  KrlMsgPar_t    Params [3]
   DECL  KrlMsgOpt_t    Options
   DECL  KrlMsgDlgSk_t Key [7]
```

2. Start programu: Odczytanie pozycji wszystkich osi i instrukcja ruchu do odczytanej pozycji (praktycznie manipulator nie wykonuje żadnego ruchu). W przypadku robota Kuka po zmianie trybu pracy na **Automatyka** (nie dotyczy to trybu **Zewnętrzna Automatyka**) trzeba utrzymywać wciśnięty przycisk **Start programu** na panelu programowania robota do momentu wykonania pierwszej instrukcji **PTP** (w domyśle ma to być instrukcja **PTP HOME**). Wykonanie poniższych instrukcji spełnia ten warunek pozwalając jednocześnie na zwolnienie przycisku:

```
AxisPosition = $AXIS_ACT
PTP AxisPosition
```

3. Odczyt i zapamiętanie aktualnie ustawionych wartości prędkości i przyspieszeń realizowanych podczas ruchu **PTP** przez każdą oś (wartości te będą odtworzone na końcu programu), a następnie ustawienie ich mniejszych wartości:

```
FOR i=1 TO 6
     STORE_VEL_AXIS[i] = $VEL_AXIS[i]
     STORE_ACC_AXIS[i] = $ACC_AXIS[i]
     $VEL_AXIS[i] = 75
     $ACC_AXIS[i] = 75
ENDFOR
```

4. Zainicjowanie zmiennych określających przyrost kąta poszczególnych osi, ustawienie numeru pierwszego menu, odczyt i zapamiętanie położenia osi manipulatora w pozycji **HOME**:

```
D1 = 1 ;--> poczatkowy przyrost kata osi A1
           to 1 stopien
D2 = 1 ;--> poczatkowy przyrost kata osi A2
           to 1 stopien
D3 = 1 ;--> poczatkowy przyrost kata osi A3
           to 1 stopien
D4 = 1 ;--> poczatkowy przyrost kata osi A4
           to 1 stopien
D5 = 1 ;--> poczatkowy przyrost kata osi A5
           to 1 stopien
```

```
D6 = 1 ;--> poczatkowy przyrost kata osi A6
           to 1 stopien
MenuNumber    = 1
HomePosition  = $H_POS
```

5. Początek głównej pętli programu: Skasowanie poprzedniego komunikatu użytkownika, odczyt informacji czy manipulator znajduje się w pozycji **HOME** i wyświetlenie komunikatu generowanego na podstawie tej informacji (w instrukcjach podkreśloną czcionką zaznaczono to, co zostanie wyświetlone na panelu programowania), odczyt aktualnej pozycji manipulatora:

```
LOOP
   Clear_KrlMsg (-99)
   IF ($IN_HOME == TRUE) THEN
     name = {modul[] „SERVICE”, Nr 1,
            msg_txt[]
            „ROBOT JEST W POZYCJI HOME”}
   ELSE
      name = {modul[] „SERVICE”, Nr 1,
            msg_txt[]
            „ROBOT NIE JEST W POZYCJI HOME”}
   ENDIF
   Handle = Set_KrlMsg (#Notify, name,
Params[], Options)
   AxisPosition = $AXIS_ACT
```

6. Jeśli wartość zmiennej **MenuNumber** wynosi 1 to wyświetlenie okna dialogowego nr **1** (Menu nr 1) związanego z obsługą obrotu osi **A1** manipulatora (patrz także przykład 3):

```
IF (MenuNumber == 1) THEN
Key[1] = {sk_Type #value, sk_txt[]
           „PREV MENU”}
Key[2] = {sk_Type #value, sk_txt[]
           „D1 MINUS”}
Key[3] = {sk_Type #value, sk_txt[]
           „A1 MINUS”}
Key[4] = {sk_Type #value, sk_txt[]
           „A1 PLUS”}
Key[5] = {sk_Type #value, sk_txt[]
           „D1 PLUS”}
Key[6] = {sk_Type #value, sk_txt[]
           „NEXT MENU”}
Key[7] = {sk_Type #value, sk_txt[] „ „}
IF ($IN_HOME == TRUE) THEN
  name = {modul[] „IN_HOME”, Nr 1,
           msg_txt[]
           „MENU 1: Obrot osia A1,
           HOME_A1 = %1,
           ACT_A1 = %2, D1 = %3 stopni:”}
 ELSE
      name = {modul[] „NOT_IN_HOME”, Nr 1,
           msg_txt[]
           „MENU 1: Obrot osia A1,
           HOME_A1 = %1,
           ACT_A1 = %2, D1 = %3 stopni:”}
  ENDIF
  Params[1] = {par_type #value,
           par_real 0.0}
  Params[1].par_real = HomePosition.A1
  Params[2] = {par_type #value,
           par_real 0.0}
  Params[2].par_real = AxisPosition.A1
  Params[3] = {par_type #value, par_int 0}
  Params[3].par_int = D1
  Options = {vl_stop true,
           clear_p_reset true,
           clear_p_SAW false,
           log_to_DB false}
   Handle  = Set_KrlDlg (name, Params[],
           Key[], Options)
ENDIF
```

7. Jeśli wartość zmiennej **MenuNumber** wynosi 2, 3, 4, 5 lub 6 to wyświetlenie w analogiczny sposób jak powyżej okna dialogowego nr 2, 3, 4, 5 lub 6 (Menu 2, 3, 4, 5 lub 6) związanego z obsługą obrotu osi **A2**, **A3**, **A4**, **A5**, **A6** manipulatora:

```
IF (MenuNumber == 2) THEN
   ; Wyswietlenie okna dialogowego nr 2
     (Menu 2)
ENDIF
-----------------------------------------
IF (MenuNumber == 6) THEN
   ; Wyswietlenie okna dialogowego nr 6
     (Menu 6)
ENDIF
```

8. Jeśli wartość zmiennej **MenuNumber** wynosi 7 to wyświetlenie okna dialogowego nr 7 obsługującego wybór podprogramów obsługi chwytaka, ruchu do pozycji **HOME** wymuszonego przez pojedynczą instrukcją **PTP HOME** bądź zakończenia wykonywania programu **MoveToHomePosition()**:

```
IF (MenuNumber == 7) THEN
   Key[1] = {sk_Type #value, sk_txt[]
            „PREV MENU”}
   Key[2] = {sk_Type #value, sk_txt[]
            „OTWORZ”}
   Key[3] = {sk_Type #value, sk_txt[]
            „ZAMKNIJ”}
   Key[4] = {sk_Type #value, sk_txt[]
            „HOME”}
   Key[5] = {sk_Type #value, sk_txt[]
            „ZAKONCZ”}
   Key[6] = {sk_Type #value, sk_txt[]
            „NEXT MENU”}
   Key[7] = {sk_Type #value, sk_txt[] „ „}
   IF ($IN_HOME == TRUE) THEN
      name = {modul[] „IN_HOME”, Nr 1,
            msg_txt[]
           „MENU 7: Sterowanie chwytakiem +
            HOME + ZAKONCZ:”}
   ELSE
      name = {modul[] „NOT_IN_HOME”, Nr 1,
            msg_txt[]
           „MENU 7: Sterowanie chwytakiem +
            HOME + ZAKONCZ:”}
   ENDIF
   Options = {vl_stop true,
            clear_p_reset true,
            clear_p_SAW false,
            log_to_DB false}
   Handle  = Set_KrlDlg (name, Params[],
            Key[], Options)
ENDIF
```

9. Jeśli wartości **Handle** zwracane przez funkcje biblioteczne **Set_KrlDlg()** do tworzenia okien dialogowych są większe od 0 (poprawnie utworzono okno dialogowe) to należy odczytać numer przycisku, który wybrano w oknie dialogowym. Liczba ta jest zwracana jako zmienna **KeyNumber**:

```
IF (Handle > 0) THEN
   WHILE (Exists_KrlDlg (Handle,
            KeyNumber))
      WAIT SEC 0.1
   ENDWHILE
```

Okno dialogowe poprawnie zwróciło numer wybranego przycisku wyboru. Początek ciągu instrukcji wykonywanych zgodnie z funkcją przypisaną do okna dialogowego i przycisku:

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **1**. We wszystkich oknach oznacza to zwiększenie o 1 numeru okna dialogowego, a w konsekwencji wyświetlenie okna odpowiadającego nowemu numerowi:

```
SWITCH KeyNumber
CASE 1
   MenuNumber = MenuNumber - 1
   IF (MenuNumber <= 0) THEN
       MenuNumber  = 7
   ENDIF
```

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **2**. W oknach dialogowych nr **i** (**i** = 1, 2, 3, 4, 5, 6) oznacza to zmniejszenie o 1 stopień wartości kąta, o który będzie przemieszczana oś **i** manipulatora (wartość kąta **Di** może zmieniać się od 1 do 10°). W przypadku okna dialogowego nr **7** przycisk nr **2** uruchamia podprogram otwierający chwytak:

```
CASE 2
   IF (MenuNumber == 1) THEN
      IF (D1 > 1) THEN
          D1 = D1 - 1
      ENDIF
   ENDIF
-----------------------------------------
   IF (MenuNumber == 6) THEN
      IF (D6 > 1) THEN
          D6 = D6 - 1
          ENDIF
       ENDIF
   IF (MenuNumber == 7) THEN
      OtworzChwytak()
   ENDIF
```

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **3**. W oknach dialogowych nr **i** (**i** = 1, 2, 3, 4, 5, 6) oznacza przemieszczenie osi **i** manipulatora o kąt **Di** w kierunku ujemnym. W przypadku okna dialogowego nr **7** przycisk nr **3** uruchamia podprogram zamykający chwytak:

```
CASE 3
   IF (MenuNumber == 1) THEN
      AxisPosition.A1 = AxisPosition.A1 - D1
      PTP AxisPosition
   ENDIF
-----------------------------------------
   IF (MenuNumber == 6) THEN
      AxisPosition.A6 = AxisPosition.A6 - D6
      PTP AxisPosition
   ENDIF
   IF (MenuNumber == 7) THEN
      ZamknijChwytak()
   ENDIF
```

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **4**. W oknach dialogowych nr **i** (**i** = 1, 2, 3, 4, 5, 6) oznacza to przemieszczenie osi **i** manipulatora o kąt **Di** w kierunku dodatnim. W przypadku okna dialogowego nr 7 przycisk nr **4** inicjuje wykonanie instrukcji **PTP HOME** – bezpośredniego ruchu wszystkimi osiami do pozycji **HOME**:

```
CASE 4
   IF (MenuNumber == 1) THEN
      AxisPosition.A1 = AxisPosition.A1 + D1
      PTP AxisPosition
   ENDIF
-----------------------------------------
   IF (MenuNumber == 6) THEN
      AxisPosition.A6 = AxisPosition.A6 + D6
      PTP AxisPosition
   ENDIF
   IF (MenuNumber == 7) THEN
      PTP HOME
   ENDIF
```

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **5**. W oknach dialogowych nr **i** (**i** = 1, 2, 3, 4, 5, 6) oznacza to zwiększenie o 1 stopień wartości kąta, o który będzie przemieszczana oś **i** manipulatora (wartość kąta **Di** może zmieniać się od 1 do 10°). W przypadku okna dialogowego nr **7** przycisk nr **5** powoduje zakończenie wykonywania pętli **LOOP ... ENDLOOP** i zakończenie wykonywania programu **MoveToHomePosition()**:

```
CASE 5
   IF (MenuNumber == 1) THEN
      IF (D1 < 10) THEN
          D1 = D1 + 1
      ENDIF
   ENDIF
-----------------------------------------
   IF (MenuNumber == 6) THEN
      IF (D6 < 10) THEN
          D6 = D6 + 1
          ENDIF
       ENDIF
   IF (MenuNumber == 7) THEN
      GOTO THE_END
   ENDIF
```

– na jednym z siedmiu okien dialogowych wybrano przycisk funkcyjny nr **6**. We wszystkich oknach oznacza to zmniejszenie o 1 numeru okna dialogowego a w konsekwencji wyświetlenie okna odpowiadającego nowemu numerowi:

```
CASE 6
   MenuNumber = MenuNumber + 1
   IF (MenuNumber > 7) THEN
      MenuNumber = 1
   ENDIF
```

– przypadek **DEFAULT** – błąd wyboru, zakończenie wykonywania programu **MoveToHomePosition()**:

```
 DEFAULT
   GOTO THE_END
```

10. **Koniec pętli**: **ENDSWITCH**, koniec pętli **ENDIF**, koniec Pętli **ENDLOOP**

```
          ENDSWITCH
       ENDIF
    ENDLOOP
```

11. **Zakończenie programu**: Przywrocenie wartości prędkości i przyspieszeń osi sprzed wywolania programu, kasowanie komunikatow użytkownika:

```
THE_END:
   FOR i=1 TO 6
      $VEL_AXIS[i] = STORE_VEL_AXIS[i]
      $ACC_AXIS[i] = STORE_ACC_AXIS[i]
   ENDFOR
   Clear_KrlMsg (-99)
   WAIT SEC 0.1
END
```

10. Uwagi końcowe

Problem doprowadzania manipulatora robota Kuka do pozycji **HOME** ze wszystkimi negatywnymi konsekwencjami, w tym z kilkoma kolizjami manipulatora, pojawił się w aplikacji wykonanej przez Przemysłowy Instytut Automatyki i Pomiarów PIAP w Cukrowni Nakło nad Notecią [7]. Eksploatację zrobotyzowanej linii do naważania i paletyzacji worków z cukrem uruchomiono zanim osoby mające nadzorować jej pracę zostały gruntownie przeszkolone w obsłudze robota i nabrały doświadczenia w obsłudze całej linii. Było to spowodowane pilną potrzebą wywiązania się Cukrowni z kontraktu na dostarczenie 4,5 tys. ton cukru w ciągu zaledwie kilku tygodni. W efekcie

praca linii początkowo była często zatrzymywana, co niekiedy wymagało restartowania programu nadrzędnego sterownika PLC i aplikacji samego robota. Ze względów bezpieczeństwa restart aplikacji robota wymagał, aby manipulator znajdował się w pozycji **HOME**, co wymuszało konieczność ręcznego doprowadzenia robota do tej pozycji. Powstał pomysł, by do aplikacji dołączyć podprogram umożliwiający bezpieczne przemieszczanie manipulatora bez potrzeby przełączania się z trybu pracy automatycznej w tryb pracy ręcznej. Ponadto podprogram powinien zapewniać bezpieczny i niezależny obrót każdej osi z minimalną prędkością i minimalnym przyrostem kąta obrotu oraz wyświetlać informację o aktualnym i docelowym położeniu przemieszczanej osi. Wydaje się, że wymagania te zostały spełnione, ponieważ od momentu włączenia podprogramu do aplikacji nie zanotowano żadnej kolizji, co oczywiście było także rezultatem nabrania doświadczenia przez personel nadzorujący pracę linii.

Zaprezentowaną procedurę można dołączyć do każdej aplikacji z robotem Kuka, ponieważ nie odwołuje się ona do żadnego narzędzia ani nie wykorzystuje żadnego układu współrzędnych zdefiniowanego przez użytkownika. Jej działanie polega tylko na obracaniu o minimalny kąt każdej osi z osobna, przez co może być bezpiecznie stosowana nawet przez osobę z niewielkim doświadczeniem w obsłudze robota.

Bibliografia

1. KUKA System Software KR C2 / KR C3: Expert Programming – Release 5.2, 2003 KUKA Roboter GmbH
2. KUKA System Software KR C...: System Variables, 2004 KUKA Roboter GmbH
3. KUKA Expert Documentation Programming Messages For KUKA System Software 8.2 and VW System Software 8.2, 2011 KUKA Roboter GmbH
4. KUKA Expert Documentation Programming Messages For KUKA System Software 5.5 and 5.6, 2009 KUKA Roboter GmbH
5. KUKA System Software 8.2: Operating and Programming Instructions for System Integrators, 2011 KUKA Roboter GmbH
6. KUKA System Software 8.3: Instrukcja obsługi i programowania użytkownika końcowego, 2013 KUKA Roboter GmbH
7. Dunaj J., *Programowa realizacja różnych schematów paletyzacji worków z luźną zawartością wykonywana robotem przemysłowym*, „Pomiary Automatyka Robotyka", R. 22, Nr 2, 2018, 49-60, DOI: 10.14313/PAR_228/49.

Control of Robot Movement to the Starting Position in Automatic Mode

Abstract: The article concerns with a problem related to the exploration of an industrial robotic station, namely how to simply and safely move the manipulator from any position to the output position without having to switch to the manual mode. The presented method is based on reading the current and the final position of each axis and its rotation by the minimum angle in both directions performed under the supervision of the operator. The article describes how the program performing this task was carried out using messages and dialog boxes displayed on the robot programming panel. Also presented in the KRL language for programming of Kuka robots, how programmer can generate user messages and dialog boxes.

Keywords: Robot's HOME position, system variables, generating dialog boxes and messages, programming in the Kuka Robot Language (KRL)

mgr inż. Jacek Dunaj
jdunaj@piap.pl

W 1980 r. ukończył studia na Wydziale Elektrycznym Politechniki Warszawskiej, od 1985 r. jest zatrudniony w Przemysłowym Instytucie Automatyki i Pomiarów. Specjalizuje się w programowaniu różnego rodzaju sprzętu: mikroprocesorów, kontrolerów, sterowników i robotów przemysłowych, systemów wizyjnych a także komputerów PC programowanych w języku asemblera i C/C++ w środowisku różnych systemów operacyjnych. Współautor oprogramowania dla kilku urządzeń opracowanych w PIAP, a także wielu wdrożeń przemysłowych, w szczególności wymagających współpracy ze sobą kilku różnych urządzeń automatyki i wykorzystania oprogramowania biurowego (baz danych, arkuszy kalkulacyjnych). Autor i wykonawca programu robota paletyzującego zrobotyzowanej linii naważania i paletyzacji worków z cukrem w Cukrowni Nakło nad Notecią.