7: 64

PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

Samodzielna Pracownia Oprogramowania Systemów

4/40 A

Główny wykonawca mgr inż. Zbigniew Pilat

Wykonawcy mgr inż. Małgorzata Jacórzyńska-Smigiera

Konsultant

Nr zlecenia S1311A

Rozwój oprogramowania podstawowego robotów URP.

Etap 7: Opracowanie opisu programu sterującego robotów URP w zakresie języka programowania, interpretera, wykonywania instrukcji oraz modelu kinematyki i generatora trajektorii.

Zleceniodawca praca statutowa PIAP

Pracę rozpoczęto dnia 01.01.94 **zakończono dnia** 28.02.94

Kierownik Pracowni — Z-ca Dyrektora d/s Badawczo-Rozwojowych

mgr inż. Zbigniew Pilat — dr inż. Jan Jabłkowski

Praca zawiera:

stron	23
rysunków	1
fotografii	
tabel	1
tablic	
załączników	

Nr rejestr. 7064

Rozdzielnik - ilość egz:

Egz. 1	BOINTE
Egz. 2	POS
Egz. 3	POS
Egz. 4	
Egz. 5	
Egz. 6	

Analiza deskryptorowa

ROBOTY PRZEMYSŁOWE: UKŁAD STEROWANIA+OPROGRAMOWANIE

Analiza dokumentacyjna

Sprawozdanie zawiera opis programu sterującego robotów URP w zakresie języka programowania, interpretera, wykonywania instrukcji oraz modelu kinematyki i generatora trajektorii robota.

Niniejsze opracowanie jest kontynuacją dokumentacji programu sterującego w wersji URP opracowanej w etapie 5 /lipiec 93 - sprawozdanie PIAP nr rej. 6978/.

Tytuły poprzednich sprawozdań

Nr rej. 6978 - Etap 5. Opracowanie opisu programu sterującego robotów URP w zakresie organizacji programu, wykorzystania zasobów sprzętowych sterownika, inicjalizacji systemu sterowania i pętli głównej programu. PIAP, lipiec 1993r.

UKD

PIAP 41/88 10000

2

1. WSTĘP.

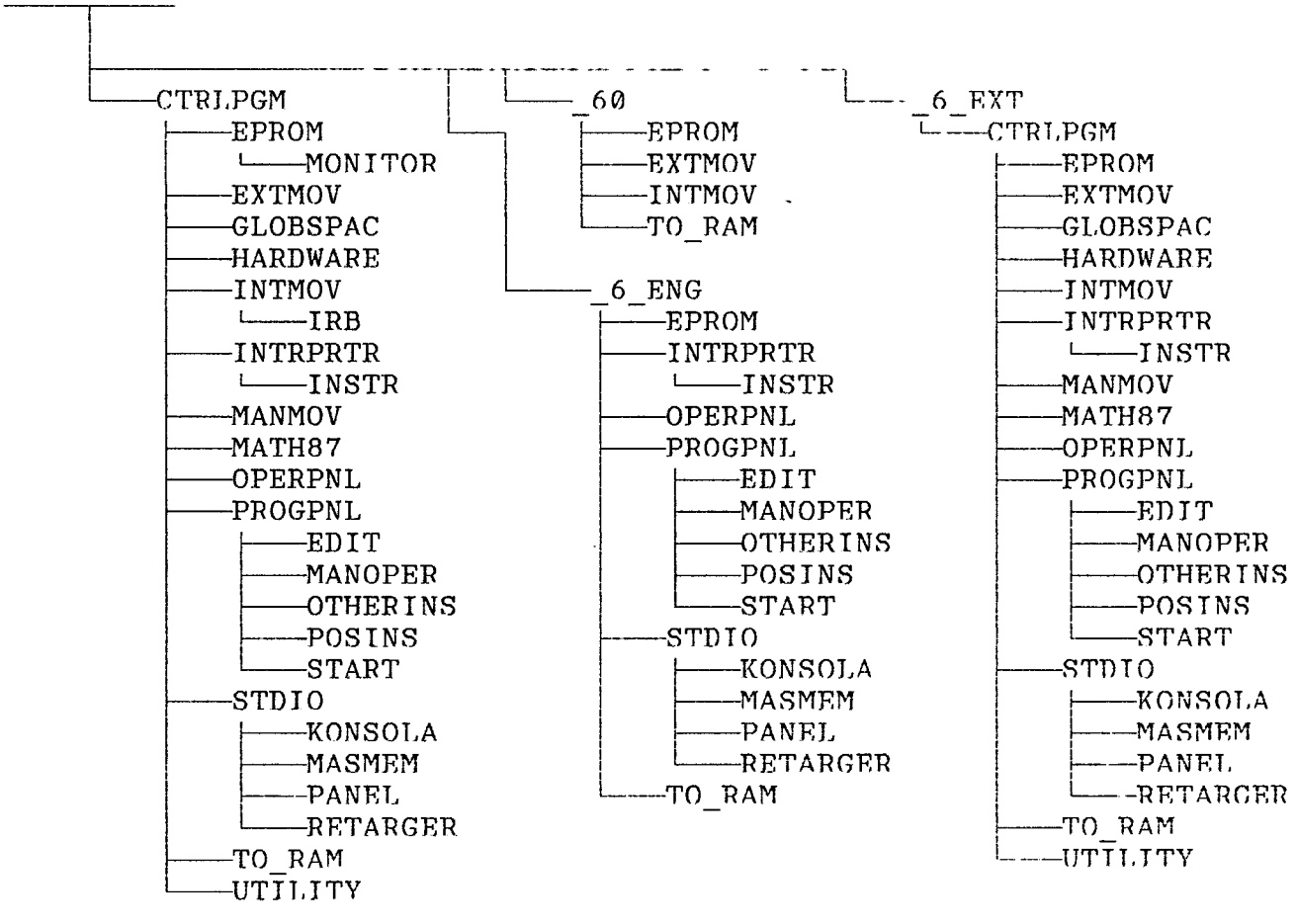
W roku 1991 rozpoczęto w PIAP prace nad nową generacją układów sterowania dla robotów przemysłowych, oznaczonych symbolem URP. Równolegle z konstrukcją hardware'ową powstawało oprogramowanie nowego sterownika. Podstawowe prace software'owe zakończono na początku 1993 roku. Wtedy też rozpoczęto dokumentowanie programu sterującego w wersji URP. W pierwszym etapie (lipiec 1993 - spraw. PIAP nr rej. 6978) opisano następujące grupy zagadnień:

- konfiguracja sprzętowa sterownika robotów URP,
- tworzenie programu sterującego,
- organizacja postaci źródłowej oprogramowania,
- ogólna struktura programu sterującego,
- inicjalizacja układu sterowania robota,
- pętla główna programu sterującego.

Niniejsze opracowanie jest kontynuacją dokumentacji. Obejmuje ono kolejne tematy istotne dla zrozumienia i opanowania programu sterującego robota.

2. ZMIANY W ORGANIZACJI POSTACI ŹRÓDŁOWEJ PROGRAMU.

W czasie od opracowania pierwszej części dokumentacji w organizacji postaci źródłowej oprogramowania zaszły pewne zmiany. Wynikały one z wprowadzania nowych typów robotów, a więc i nowych wersji programów sterujących. Ukończone zostało oprogramowanie robota URP-60. W związku z planami eksportowymi, opracowano wersję angielskojęzyczną dla robota URP-6/10 (kinematyka części mechanicznych obu tych robotów jest identyczna, realizowane funkcje również, więc programy sterujące różnią się tylko napisem na panelu programowania w czasie restartu - podawany jest wtedy typ robota). Samodzielną wersją jest program sterujący dla robota URP-6 z dwoma osiami zewnętrznymi, wykonywany specjalnie dla stanowiska spawalniczego z dwoma stolikami obrotowymi. Ponieważ niektóre części, katalogi oprogramowania są wspólne dla różnych wersji, zdecydowano się umieścić całe oprogramowanie źródłowe na wspólnym katalogu URP_PROG. Mieszczący się na nim podkatalog CTRLPGM zawiera podstawową wersję dla robota URP-6. W podlkatalogu _6_EXT jest program dla robota z osiami zewnętrznymi, w podkatalogu _60 - dla URP-60, a w _6_ENG - wersja angielskojęzyczna. Przy tworzeniu postaci wynikowej dla robota URP-60 i wersji angielskojęzycznej korzysta się z bibliotek wersji URP-6. Aktualna struktura katalogu URP-PROG przedstawiona jest na rysunku poniżej.



Rys. 2.1 Struktura katalogu URP_PROG zawierającego postać źródłową programu sterującego.

3. JĘZYK PROGRAMOWANIA ROBOTÓW URP.

Ogólna postać instrukcji w języku programowania robotów URP wygląda następująco:

NUMER MNEMONIK PARAMETRY

NUMER - jest liczbą całkowitą od 1 do 9999,
MNEMONIK - jest słowem określającym typ instrukcji, czasem mogą to być dwa słowa,
PARAMETRY - są instrukcje bezparametrowe, inne mają parametry numeryczne, jeszcze inne parametry nienumeryczne.

W chwili obecnej w programie sterującym robotów URP zaimplementowano 19 instrukcji. Jedna z nich - instrukcja pusta - nie ma mnemoniku. Jest ona niewidzialna dla operatora robota. W każdym programie użytkowym występuje tylko raz, na końcu i jest wstawiana automatycznie przez układ sterowania. Instrukcja ta jest więc znacznikiem końca obszaru programów użytkowych. W obecnej wersji programu sterującego przewidziano wprowadzenie czterech kolejnych instrukcji. Są to włączanie i wyłączanie oscylacji, ruch z interpolacją kołową, ruch ze stałym TCP oraz poszukiwanie swobodne wg sygnałów z czujników. W tablicy kodów, która jest zapisana w zbiorze nagłówkowym inscodes.h w katalogu INTRPRTR, instrukcjom tym zarezerwowano już miejsca i przypisano kody. Poniżej w tabeli zebrano instrukcje opisane w tablicy kodów, wymieniono ich mnemoniki, symbole kodów i kody (używane w programie sterującym).

Mnem.	Symbol kodu	Kod	Instrukcja
***	EMPTY_CODE	0	Instrukcja pusta - koniec programu
POS LIN	POSL_CODE	1	Pozycjonowanie liniowe
**	POSO_CODE	2	Pozycjonowanie bez zmiany TCP
POS QLIN	POSQ_CODE	3	Pozycjonowanie quasiliniowe
**	POSC_CODE	4	Pozycjonowanie kołowe zwykłe
**	POSS_CODE	5	Pozycjonowanie z szukaniem swobodnym
NARZĘDZIE	TOOL_CODE	6	Wybór narzędzia
PRĘDKOŚĆ	DEFV_CODE	7	Określenie predkosci
CHWYTAK	GRIP_CODE	8	Operacja na chwytakach
WY/FL	SET_CODE	9	Ustawienie flagi lub wyjścia
CZEKAJ	WAIT_CODE	10	Czekanie
CZEKAJ	WAITEQ_CODE	11	Czekanie na spełnienie równości
CZEKAJ	WAITNE_CODE	12	Czekanie na spełnienie nierówności
SKOK	JUMP_CODE	13	Skok bezwarunkowy
SKOK GDY	JUMPEQ_CODE	14	Skok warunkowy jeśli równość
SKOK GDY	JUMPNE_CODE	15	Skok warunkowy jeśli nierówność
POWTÓRZ	LOOP_CODE	16	Początek pętli
KONIEC POWT	LPEND_CODE	17	Koniec pętli
WYKONAJ	CALL_CODE	18	Wywołanie podprogramu
PODPROGRAM	SUBR_CODE	19	Początek podprogramu
POWRÓT	RET_CODE	20	Powrót z podprogramu
**	ACTOSC_CODE	21	Aktywacja ruchu oscylacyjnego
**	DEAOSC_CODE	22	Dezaktywacja ruchu oscylacyjnego
POCZĄTEK	STRPGM_CODE	23	Określenie podprogramu rozpoczynaj.
KONIEC	ENDPGM_CODE	24	Określenie programu kończącego

- ** - kody zarezerwowane dla instrukcji przewidzianych do implementacji w dalszych pracach,

- *** - instrukcja pusta - nie ma mnemoniku.

W katalogu INTRPRTR znajduje się też plik intrpcns.c86, zawierający deklarację długości poszczególnych instrukcji w pamięci systemu sterowania. Długości te są zapisane w tablicy instr_lenght[]. Jej długość jest równa liczbie instrukcji zadeklarowanych w programie (patrz tab. wyżej). Adres tablicy instr_lenght[] jest zadeklarowany jako zmienna globalna, dzięki czemu długości poszczególnych instrukcji są dostępne w innych segmentach oprogramowania sterującego. Oczywiście długości te muszą się zgadzać z rozmiarami struktur opisujących pola poszczególnych instrukcji. Struktury te są zdefiniowane w pliku nagłówkowym instruct.h, w katalogu INTRPRTR. Zgodnie z tymi definicjami poszczególne instrukcje mają następującą postać:

Instrukcja pozycjonowania liniowego POZ LIN (długość 30 bajtów):

- bajt 0 - kod instrukcji: POSL_CODE

- bajty 1-2 - numer instrukcji

- bajt 3: - bity 0-5: - niewykorzystane
  bit 6: 0 - pozycjonowanie bezwzględne
  1 - pozycjonowanie względne
  bit 7: 0 - pozycjonowanie zgrubne
  1 - pozycjonowanie dokładne (następna instrukcja ruchu zostaje rozpoczęta po pojawieniu się sygnału INPOS poprzedniej instrukcji)

- bajt 4-5 - prędkość ruchu - jest wyrażona w dziesiątych częściach % wartości prędkości podstawowej,

- bajty 6-9 - współrzędna X punktu TCP,

- bajty 10-13 - współrzędna Y punktu TCP,

- bajty 14-17 - współrzędna Z punktu TCP,

- bajty 18-21 - nutacja osi narzędzia,

- bajty 22-25 - precesja osi narzędzia,

- bajty 26-29 - obrót osi narzędzia,

Instrukcja pozycjonowania quasiliniowego POZ QLIN (26 bajtów):

- bajt 0 - kod instrukcji: POSQ_CODE

- bajty 1-2 - numer instrukcji

- bajt 3: - bity 0-4: - niewykorzystane
  bit 5: 0 - w bajtach 4-5 zapamiętana jest prędkość ruchu
  1 - w bajtach 4-5 zapamiętany jest czas ruchu
  bit 6: 0 - pozycjonowanie bezwzględne
  1 - pozycjonowanie względne
  bit 7: 0 - pozycjonowanie zgrubne
  1 - pozycjonowanie dokładne (analog. jak dla POZ_LIN)

- bajt 4-5 - prędkość lub czas pozycjonowania (decyduje o tym bit 5 bajtu 3):
  - prędkość jest wyrażona w % maksymalnej wartości prędkości konstrukcyjnej robota,
  - czas jest wyrażony w dziesiątych częściach sekundy,

- bajty 6-9 - współrzędna fi - pozycja osi pierwszej,

- bajty 10-13 - współrzędna teta - pozycja osi drugiej,

- bajty 14-17 - współrzędna alfa - pozycja osi trzeciej,

- bajty 18-21 - współrzędna t - pozycja osi czwartej,

- bajty 22-25 - współrzędna v - pozycja osi piątej,

Instrukcja wyboru narzędzia - NARZĘDZIE (4 bajty):
- bajt 0 - kod instrukcji TOOL_CODE,

- bajty 1-2 - numer instrukcji,

- bajt 3 - numer narzędzia.

Instrukcja deklaracji prędkości ruchu - PRĘDKOŚĆ: (7 bajtów)
- bajt 0 - kod instrukcji DEFV_CODE,

- bajty 1-2 - numer instrukcji,

- bajty 3-4 - prędkość podstawowa w (mm/s).

- bajty 5-6 - prędkość maksymalna w (mm/s)

Instrukcja operacji na chwytakach - CHWYTAK: (6 bajtów)
- bajt 0 - kod instrukcji GRIP_CODE

- bajty 1-2 - numer instrukcji

- bajt 3 - zadany stan chwytaków:
  bity 0-1 - numer chwytaka (1 lub 2)
  bit 7 - zadany stan: 0 - otwarty, 1 - zamknięty,
  bit 7 - zadany stan: 0 - otwarty, 1 - zamkniety.

- bajty 4-5 - wielkość opóźnienia (w dziesiątych sekundy).

Instrukcja ustawiania flagi lub wyjścia - WY/FL: (5 bajtów)
- bajt 0 - kod instrukcji SET_CODE,

- bajty 1-2 - numer instrukcji,

- bajt 3 - ustawiany obiekt 1-bitowy,

- bajt 4 - obiekt 1-bitowy, którego wartość podstawiamy pod obiekt w bajcie 3. Obiekt 1-bitowy jest określony nastepująco:
  bity 0-5 - numer obiektu (numer flagi, wejścia lub wyjścia) albo wartość natychmiastowa (0 lub 1),

bity 6-7 - typ obiektu, a mianowicie:
00B - wartość natychmiastowa,
01B - flaga,
10B - wejście,
11B - wyjście.

Instrukcje czekania - CZEKAJ: (5 bajtów)
Dla czekania bezwarunkowego:

- bajt 0 - kod instrukcji WAIT_CODE,
- bajty 1-2 - numer instrukcji,
- bajty 3-4 - czas czekania (w dziesiątych sekundy).

Dla czekania warunkowego:

- bajt 0 - kod instrukcji WAITEQ (czekanie na równość) lub WAITNE (czekanie na nierówność),
- bajty 1-2 - numer instrukcji,
- bajt 3 - lewy argument relacji (obiekt 1-bitowy),
- bajt 4 - prawy argument relacji (obiekt 1-bitowy).

Instrukcje skoku - SKOK (bezwar. - 5, war. - 7 bajtów):
- bajt 0 - kod instrukcji JUMP (skok bezwarunkowy), JUMPEQ (skok przy równości) lub JUMPNE (skok przy nierówności),
- bajty 1-2 - numer instrukcji
- bajty 3-4 - numer instrukcji, do której ma nastąpić skok.

Dodatkowo dla skoku warunkowego:

- bajt 5 - lewy argument relacji (obiekt 1-bitowy),
- bajt 6 - prawy argument relacji (obiekt 1-bitowy).

Instrukcja programowania początku pętli - POWTÓRZ (5 bajtów):
- bajt 0 - kod instrukci LOOP_CODE,
- bajty 1-2 - numer instrukcji,
- bajty 3-4 - liczba powtórzeń.

Instrukcja programowania końca pętli - KONIEC POWTARZANIA: (3 bajty)
- bajt 0 - kod instrukcji LPEND_CODE,
- bajty 1-2 - numer instrukcji.

Instrukcja wywołania podprogramu - WYKONAJ (5 bajtów):
- bajt 0 - kod instrukcji CALL_CODE,
- bajty 1-2 - numer instrukcji,
- bajty 3-4 - numer instrukcji początku podprogramu,

Instrukcja powrotu z podprogramu - POWRÓT: (3 bajty)
- bajt 0 - kod instrukcji RET_CODE,
- bajty 1-2 - numer instrukcji,

Instrukcja początku podprogramu - PODPROGRAM: (3 bajty)
- bajt 0 - kod instrukcji SUBR_CODE,
- bajty 1-2 - numer instrukcji,

Instrukcja deklaracji podprogramu końcowego - KONIEC: (5 bajtów)
- bajt 0 - kod instrukcji ENDPGM_CODE,
- bajty 1-2 - numer instrukcji,
- bajty 3-4 - numer instrukcji początku podprogramu końcowego,

Instrukcja deklaracji podprogramu początkowego - POCZĄTEK: (5 bajtów)
- bajt 0 - kod instrukcji STRPGM_CODE,
- bajty 1-2 - numer instrukcji,
- bajty 3-4 - numer instrukcji początku podprogramu początkowego,

4. INTERPRETER.

Interpreter jest odpowiedzialny za realizację programu użytkowego robota w trybie automatycznym lub w pracy krokowej. Procedury wykonawcze interpretera są zebrane w katalogu INTRPRTR. Podprogramy te korzystają z pewnej wydzielonej grupy zmiennych globalnych, które są zadeklarowane w pliku intrprvar.c86:

```
char *instr_param - pointer na pole parametrów aktualnie
                    wykonywanej instrukcji,
char flag[NFLAG] - tablica flag programu użytkowego,
char input_prepared - wskaźnik ustawienia wejścia przez użytkownika,
char output_send - wskaźnik wysyłania/niewysyłania wartości
                   na wyjście,
char step_work - określa czy interpreter pracuje krokowo,
char stop_interpreter - określa czy należy zatrzymać wykonywanie
                        programu użytkowego,
char velocity_defined - określa czy została zdefiniowana prędkość
                        podstawowa w ruchu interpolowanym (liniowy),
unsigned char simulation_state - określa stan aktywności symulacji:
       bit 0: symulacja wejść:
              0 - symulacja wyłączona,
              1 - symulacja włączona i wartość wejść wprowadzana
                  jest z panelu programowania,
       bit 1 i 2: symulacja wyjść:
              0 - symulacja wyłączona,
              1 - wartości wyjściowe nie są nigdzie wysyłane,
              2 - wartości wyjść wysyłane są do panelu do akceptacji.
int end_program - numer programu wykonywanego przy zakończeniu
                  wykonywania programu użytkowego,
int start_program - numer programu wykonywanego przy starcie
                    wykonywania programu użytkowego,
int vel_coeff_delta - przyrost prędkości w ruchu automatycznym
                      [promile]
double base_velocity - prędkość podstawowa w pozycjonowaniu
                       liniowym [mm/s]
double max_velocity - prędkość maksymalna w pozycjonowaniu
                      liniowym [mm/s]

double actual_base_velocity - aktualna prędkość podstawowa w ruchu
                              interpolowanym (liniowy) - jest
                              pamiętana w takich jednostkach,
                              aby wynik pomnożenia jej przez liczbę
                              oznaczającą ilość dziesiątych części
                              procenta był od razu właściwą
                              prędkością wyrażoną w mm/s,

LOOP_DESCRIPTION loopd[MAXLOOP] - tablica opisów pętli,
PROGRAM *program_addr - adres pola opisu aktualnego programu,

unsigned char call_exist - wskaźnik informujący, że miało już
                           miejsce wywołanie podprogramu (powrót
                           nie jest błędem),
unsigned char call_number - licznik zagnieżdżeń wywołań podprogramów,
unsigned char max_call_number - maksymalna liczba zagnieżdżeń
                                podprogramów,
unsigned int call_stack[10] - stos adresów powrotu z podprogramów,
```

Zmienne te są inicjowane podczas inicjacji całego interpretera

(wywołanie procedury inijintrp() w podprogramie inicjacji systemu sterowania initmain()). Po zainicjowaniu interpreter jest gotowy do pracy. Jego procedura główna - intrprtr() - jest zapisana w pliku intrprtr.c86. Procedura ta jest wywoływana w podprogramie contpgm() (plik contpgm.c86 w katalogu STARTPROGPNL) po zainicjowaniu przez operatora rozpoczęcia wykonywania programu użytkowego w cyklu automatycznym lub krokowo. Danymi wejściowymi dla interpretera są przede wszystkim dane określające aktualną instrukcję. Nie musi to być pierwsza instrukcja w programie. Może to być nawet dowolna instrukcja podprogramu. Dzięki temu możliwe jest wejście do interpretera w celu kontynuacji przerwanego programu użytkowego. Wynik działania zwracany jest poprzez wartość funkcji w sposób następujący:

OK - normalne wyjście z interpretera, po prawidłowym przebiegu wykonania instrukcji (zakończenie pojedynczej instrukcji w pracy krokowej lu podanie komendy STOP_PROGRAMU przez operatora),
kod błędu - wyjście z powodu błędu wykonania programu użytkowego

W przypadku wyjścia z interpretera z powodu błędu wykonania jest możliwa kontynuacja wykonywania programu użytkowego albo od poziomu (pod)programu, w którym wystąpił błąd (gdyż pamiętana jest "droga powrotu"), lub od początku. W tym ostatnim przypadku musi zostać wykonana procedura zdjęcia stanu "bycia wykonywanym" dla podprogramów, które były w fazie wykonywania. Trzeba też zamknąć otwarte pętle repetycji. Jeśli instrukcja zostanie przerwana w trakcie jej wykonywania (jak to jest możliwe w przypadku instrukcji pozycjonowania oraz czekania) to indeks następnej instrukcji do wykonania (pamiętany w tablicy opisu programu) zostaje ustawiony na indeks przerwanej instrukcji. Dzięki temu ewentualna kontynuacja przerwanego programu użytkowego rozpocznie się od dokonczenia przerwanej instrukcji.

Na początku interpreter ustawia zmianną globalną system_state na wartość AUTO_STATE. Jest to informacja dla innych segmentów programu sterującego, że odbywa się aktualnie wykonywanie programu użytkowego.[1] Zmieniane są ograniczenia na prędkości ruchu wszystkich osi (wywołanie procedury set_intvelconstr()). W pracy ręcznej robot może uzyskać co najwyżej 25% swojej maksymalnej prędkości konstrukcyjnej, natomiast w pracy automatycznej do 100%. Następnie uaktualniana jest wartość współczynnika prędkości (w zależności od aktualnie dobranych przez operatora warunków - funkcja +/- na panelu programowania), po czym na panelu programowania wyświetlony zostaje napis: PRACA AUTOMATYCZNA. Jeśli jest włączona symulacja, w dolnej linii wyświetlacza podawana jest dodatkowa informacja o tym fakcie. W dalszej kolejności zaświecana jest lampka START_PROGRAMU na panelu operacyjnym oraz ustawiany jest zewnętrzny sygnał START_PROGRAMU (informacja do urządzeń współpracujących).[2] Jeśli w programie został zadeklarowany podprogram początkowy, to jest on wykonywany w pierwszej kolejności. Następnie sterowanie jest przekazywane do procedury doinstr(), która jest odpowiedzialna za wykonanie pojedynczej instrukcji. Jeśli interpreter został wywołany w następstwie zainicjowania pracy krokowej, to doinstr() jest wykonywana raz i program przechodzi do sekwencji wyjściowej z interpretera. Jeślki jest to praca ciągła to doinstr() jest wywoływana cyklicznie aż do stwierdzenia konieczności przerwania pracy

---

1. w wersji URP-10 wprowadzono dodatkową zmienną work_mode, która określa tryb pracy robota. Może ona przyjmować dwie wartości AUTO_MODE lub MANUAL_MODE. W trybie MANUAL_MODE uruchomienie interpretera nie jest możliwe. Zmiana trybów odbywa się przyciskami na panelu operacyjnym. Wykorzystano do tego dotychczasowe lampki-przyciski TEMPERATURA i CZYTANIE Z PAMIĘCI
2. w opisywanej wersji URP-6 realizuje się to poprzez procedurę outbyte() - wysłanie bajtu na zewnątrz; w wersji URP-10 wprowadzono specjalne funkcje light_2() i extingsh_2() do operowania tym sygnałem

automatycznej (błąd lub interwencja operatora - sprawdza to procedura intrrptd(). W procedurze doinstr() jest automatycznie uaktualniany indeks bieżącej instrukcji (kolejne instrukcje są pobierane z obszaru programów użytkowych). W sekwencji wyjściowej program sprawdza czy ostatnia instrukcja została dokończona. Jeśli nie, to ona jest traktowana jako instrukcja bieżąca, w przeciwnym razie instrukcją bieżącą jest kolejna instrukcja programu. Jeżeli w programie był zadeklarowany podprogram kończący, to jest on wykonywany. Następnie program gasi lampkę i zewnętrzny sygnał START_PROGRAMU, ustawia stan pracy ręcznej, ograniczenia prędkości dla tego stanu, przywraca aktualny stan wyświetlaczy i przekazuje sterowanie do procedury wywołującej.

5. REALIZACJA INSTRUKCJI PROGRAMU UŻYTKOWEGO.

Podprogramy odpowiedzialne za realizację poszczególnych instrukcji programu użytkowego są zgrupowane w katalogu INTRPRTR\INSTR. Wszystkie procedury zwracają wartość typu 'int' jako informację o sposobie wykonania. Jeśli instrukcja wykonała się prawidłowo, zawsze zwracana jest wartość zero '0'. W przeciwnym wypadku odpowiedni kod błędu. Daną wejściową dla wszystkich procedur jest adres ciała instrukcji, która ma być realizowana (niektóre procedury danej tej nie wykorzystują). Poniżej zostaną kolejno omówione procedury realizujące instrukcje.

5.1. Instrukcja deklaracji narzędzia - NARZĘDZIE.

Za realizację instrukcji NARZĘDZIE odpowiedzialna jest procedura tool() zapisana w pliku tool.c86. W polu opisującym tę instrukcję w pamięci systemu sterowania, zapisany jest jeden parametr - numer narzędzia, którym będzie operował robot. Pierwszą czynnością programu jest zbadanie, czy narzędzie o zadeklarowanym numerze istnieje w systemie (wywołanie funkcji findtool()), tzn. czy zostało przez operatora wcześniej zdefiniowane. Jeśli tak, to wywoływana jest procedura definiowania narzędzia aktywnego (deftool()), która przełącza jako aktywne narzędzia o zadeklarowanym numerze.

```
WYWOŁANIE
    r = call(instruction);
DANE WYJSCIOWE
    int r;              - określa wynik działania:
        OK                  - instrukcja wykonana prawidłowo
        TOLL_MISSING        - nie znaleziono narzędzia o zadanym numerze
    int tool_number - numer aktualnego narzedzia
    TOOL *tool_addr - adres struktury zawierającej opis aktualnego
                      narzedzia
```

5.2. Instrukcja deklaracji prędości podstawowej i maksymalnej dla ruchu liniowego - PRĘDKOŚĆ.

Instrukcję tę realizuje procedura defv(), zapisana w zbiorze defv.c86. Z ciała instrukcji pobierana jest wartość prędkości maksymalnej i wstawiana do zmiennej globalnej max_velocity. Do zmiennej globalnej base_velocity jest zapisywana podana w instrukcji wartośćprędkości podstawowej. Dodatkowo obliczana jest wartość bieżącej prędkości podstawowej - uwzględniająca również współczynnik prędkości dla całego programu. Wartość ta jest zappamiętywana w zmiennej globalnej actual_base_velocity. Na koniec ustawiana jest na "1" zmienna globalna velocity_defined - jest ona sprawdzana przy rozpoczęciu wykonywania instrukcji pozycjonowania liniowego. Jeśli jej wartość jest różna od "1" (tzn. nie ustawiono wcześniej prędkości podstawowej i maksymalnej) to pozycjonowanie liniowe nie wykona się - system zgłosi błąd. Mechanizm ten wymusza, aby operator wstawił w programie użytkowym instrukcję PRĘDKOŚĆ przed jakoąkolwiek instrukcją pozycjonowania liniowego. Jest wtedy gwarancja, że prędkość ruchu robota została ustalona świadomie. Jest to szczególnie ważne właśnie dla pozycjonowania liniowego, które z

natury wykorzystywane jest jako ruch technologiczny i niewłaściwa prędkość tego ruchu mogłaby mieć złe następstwa.

```
WYWOŁANIE
    r = call(instruction);
DANE WYJSCIOWE
    int r;                  - określa wynik działania:
        OK                      - instrukcja wykonana prawidłowo
        CALL_NOT_FOUND          - nie znaleziono podprogramu o zadanym
                                  numerze
    unsigned int actprog - indeks aktualnego (pod)programu (zmienna
                           globalna)
```

5.3. Instrukcja czekania - CZEKAJ.

Za realizację instrukcji czekania odpowiedzialne są trzy procedury. Interpreter wybiera jedną z nich w zależności od kodu zapisanego w ciele instrukcji. Dla czekania na upływ czasu jest to procedura wait() (plik wait.c86). Pobiera ona z ciała instrukcji wartość czasu oczekiwania i przyjmuje ją jako parametr wywołania podprogramu waitstat() (plik waitstat.c86 w katalogu INTRPRTR). W podprogramie tym jest odliczany czas z jednoczesną kontrolą, czy nie nastąpiła interwencja operatora przerywająca czekanie (przycisk STOP PROGRAMU) lub inne zdarzenie, po którym instrukcję należy zakończyć (np. stwierdzenie błędu pracy systemu sterowania). Wartość zwracana przez waitstat(), informująca o przebiegu czekania jest następnie przekazywana do procedury wywołującej wait(). Dla czekania na spełnienie relacji równości interpreter wywołuje procedurę waiteq()(waiteq.c86). Dla czekania na spełnienie relacji nierówności - waitne() (waitne.c86). Obie one korzystają z tego samego podprogramu o nazwie waitcond() (plik waitcond.c86 w katalogu INTRPRTR). Parametrem wywołania tego podprogramu jest adres ciała instrukcji czekania w obszarze pamięci systemu (skąd pobierane są lewa i prawa strona relacji) oraz kod typu relacji (EQ = 0 przy relacji równości i NE = 1 przy czekaniu na relację nierówności. Wartość zwracana przez waitcond(), informująca o przebiegu czekania jest następnie przekazywana do procedury wywołującej waitne() lub waiteq().

5.4. Instrukcja skoku - SKOK.

Podobnie jak w przypadku czekania instrukcja SKOKU realizowana jest na trzy sposoby. Za wykonanie skoku bezwarunkowego odpowiedzialna jest procedura jump() (w pliku jump.c86). Na początku odszukuje się w niej instrukcji, do której ma nastąpić sko. Jeśli instrukcji o takim numerze w programie nie ma to sygnalizowany jest błąd - procedura zwraca wartość JUMP_NOT_FOUND. Po znalezieniu instrukcji docelowej ustawia się jej indeks jako indeks kolejnej instrukcji do wykonania i procedura kończy działanie zwracając wartość OK (0).

W przypadku skoku warunkowego wywoływana jest procedura jumpeq.c86() (plik jumpeq.c86 - relacja równości) lub jumpne() (jumpne.c86 - relacja nierówności). Obie te procedury wywołują podprogram jumpcond() (plik jumpcond.c86 w katalogu INTRPRTR) podając jako dane wejściowe adres ciała instrukcji skoku w pamięci systemu i parametr określający typ relacji. Podprogram jumpcond() sprawdza wartość logiczną relacji. Jeśli

jest ona prawdziwa, wywoływany jest podprogram wykonujący skok bezwarunkowy - jump(). W przeciwnym wypadku następuje powrót z procedury bez podjęcia jakiejkolwiek akcji.

5.5. Instrukcja ustawienia stanu wyjścia lub flagi - WY/FL.

Instrukcję tę, polegającą na przyporządkowaniu wyjściu lub fladze odpowiedniej wartości, realizuje procedura set(), zapisana w zbiorze set.c86. Odczytuje ona na początku wartość prawego argumentu przyporządkowania (co ma być podstawione). Następnie sprawdza typ lewego argumentu przyporządkowania (co ma być zmienione). Jeśli lewym argumentem jest stała lub wejście, to sygnalizowany jest błąd. W przypadku flagi ustawiany jest odpowiedni element tablicy flag[]. Działanie procedury set() dla wyjść jest różne w zależności od tego, czy jest włączona symulacja. Jeśli nie - wywoływana jest procedura userout() (plik userout.c86 w katalogu HARDWARE), która podaje odpowiednią wartość na wskazane wyjście użytkowe. W przeciwnym razie procedura bada rodzaj symulacji wyjść. Jeśli jest to symulacja z wyjściami zaślepionymi, nie jest podejmowane żądane działanie. Program użytkowy biegnie dalej tak jakby operacja na wyjściu została wykonana ale stan wyjścia nie uległ zmianie. Jeśli jest włączona symulacja z akceptacją, system nawiązuje dialog z operatorem, aby ten mógł świadomie wprowadzić wartość na jaką ma zostać ustawione wyjście.

5.6. Instrukcja operacji na chwytakach - CHWYTAK.

Za realizację tej instrukcji odpowiedzialna jest procedura grip() zapisana w zbiorze grip.c86. Jej parametrem jest zarówno numer chwytaka, jego stan po operacji (zamknięty/otwarty) jak i opóźnienie przed przejściem do wtykonania kolejnej instrukcji (daje to możliwość uwzględnienia czasu zadziałania urządzenia wykonawczego chwytaka np. siłownika pneumatycznego). Na podstawie tych wartości, zapisanych w ciele instrukcji CHWYTAK, w procedurze grip() wypełniane są pola unii grippers_state. Unia ta jest następnie parametrem wywołania funkcji grippers() (w katalogu HARDWARE), która dokonuje odpowiednich zmian wartości wyjść dwustanowych sterujących zaworami pneumatycznymi chwytaka (w ramieniu górnym robota). Następnie program przeczekuje czas zadeklarowany w instrukcji CHWYTAK jako opóźnienie.

5.7. Instrukcja deklaracji początku pętli - POWTÓRZ.

Parametrem instrukcji POWTÓRZ jest liczba zaprogramowanych powtórzeń pętli programowej. Procedura realizująca tę instrukcję - loop() (plik loop.c86) sprawdza na początku czy nie osiągnięto już maksymalnej liczby zagnieżdżeń pętli (zmienna nesting nie może przekroczyć stałej MAXLOOP - deklarowana w nagłówku intrprtr.h, obecnie ustawiona na 8). W takim przypadku zwracana jest informacja o błędzie - LOOP_OVERFLOW). Jeśli można otworzyć kolejną pętlę, zwiększany jest licznik zagnieżdżeń nesting i ustawiane są dwa kolejne elementy tablicy opisu pętli aktywnych loopd[]. W pole .rep wpisywana jest zadana liczba powtórzeń, a w pole .loopbegin - indeks kolejnej instrukcji po instrukcji POWTÓRZ -

do niej będzie wracał program po napotkaniu instrukcji KONIEC POWTARZANIA. Struktura LOOP_DESCRIPTION opisująca tablicę loopd[] jest zdefiniowana w zbiorze nagłówkowym types.h w katalogu INTRPRTR.

5.8. Instrukcja deklaracji końca pętli - KONIEC POWTARZANIA.

Instrukcja ta jest znacznikiem końca pętli. Realizuje ją procedura lpend() (w pliku lpend.c86). Jeśli w momencie wykonania tej instrukcji nie jest aktywna żdana pętla, procedura sygnalizuje błąd - LPEND_NO_LOOP. W przeciwnym razie zmniejszana jest liczba powtórzeń pozostałych do wykonania - pole .rep w tablicy opisu pętli loopd[]. Jeśli liczba ta osiąga zero, zmniejszana jest liczba zagnieżdżeń (liczba otwartych pętli - zmienna nesting) i procedura kończy działanie - system sterowania przechodzi do wykonania kolejnej instrukcji programu użytkowego. Jeśli nie był to ostatni obieg pętli, na indeks kolejnej instrukcji programu podstawiana jest wartość z pola .loopbegin tablicy loopd[], zapamiętana podczas wykonania instrukcji POWTÓRZ - do wykonania zostaje pobrana instrukcja następna za POWTÓRZ.

5.9. Instrukcja wywołania podprogramu - WYKONAJ.

Instrukcję tę realizuje procedura call(), zapisana w pliku call.c86. Na początku sprawdza się, czy możliwe jest wywołanie podprogramu, tzn. czy nie zostanie przekroczona maksymalna liczba zagnieżdżeń (przechowywana w zmiennej globalnej max_call_number, obecnie ustawionej na 9).

```
WYWOŁANIE
    r = call(instruction);
DANE WYJSCIOWE
    int r;                  - określa wynik działania:
        OK                      - instrukcja wykonana prawidłowo
        CALL_NOT_FOUND          - nie znaleziono podprogramu o zadanym
                                  numerze
    unsigned int actprog - indeks aktualnego (pod)programu (zmienna
                           globalna)
```

5.10. Instrukcja początku podprogramu - PODPROGRAM.

Instrukcję tę realizuje procedura subr(), zapisana w zbiorze subr.c86. Program sprawdza, czy przejście do wykonania instrukcji PODPROGRAM nastąpiło na skutek wywołania podprogramu (instrukcja WYKONAJ). Jeśli tak to procedura kończy działanie, zwracając wartość OK (=0). W przeciwnym wypadku sygnalizowany jest błąd. Procedura korzysta ze zbiorów nagłówkowych errors.h i main.h z katalogu CTRLPGM.

```
WYWOŁANIE
    r = subr(instruction);
DANE WYJSCIOWE
    int r;                  - okresla wynik dzialania:
```

OK	- instrukcja wykonana prawidlowo
SUBR_NOT_CALLER	- proba wykonania podprogramu bez jego wcześniejszego wywolania

5.11. Instrukcja powrotu z podprogramu - POWRÓT.

Instrukcję tę realizuje procedura ret(), zapisana w zbiorze ret.c86. Sprawdza ona, czy wskaźnik liczby zagnieżdżeń podprogramów (call_number) jest różny od zera (oznaczałoby to próbę wykonania instrukcji POWRÓT z programu głównego). Następnie, na podstawie tablicy stosu podprogramów (call_stack) odnajdywany jest indeks instrukcji wywołującej aktualny podprogram, a następnie obliczany jest jej adres w pamięci systemu. W dalszej kolejności, uwzględniając długość instrukcji WYKONAJ, program uaktualnia indeks kolejnej instrukcji programu użytkowego. Na koniec zmniejszany jest o jeden wskaźnik zagnieżdżeń podprogramów.

WYWOŁANIE

```
r = ret(instruction);
```

DANE WYJSCIOWE

int r;	- określa wynik działania:
OK	- instrukcja wykonana prawidłowo
RETURN_FROM_MAIN	- próba powrotu z programu głównego
RET_NOT_FOUND	- nie znaleziono instrukcji wywołującej
program_addr	- adres aktualnego programu (zm. globalna) - struktura zawierająca m.in. indeks kolejnej instrukcji programu użytkowego.

5.12. Instrukcja deklaracji podprogramu końcowego - KONIEC.

W instrukcji KONIEC deklarowany jest numer podprogramu, wywoływanego po każdym zatrzymaniu automatycznego wykonywania programu użytkowego robota. Numer ten jest pobierany z ciała instrukcji, z pola instruction->prog_nr i wstawiany do zmiennej globalnej end_program. Zmienna ta jest wykorzystywana przez interpreter w sytuacji przerwania pracy automatycznej (obojętne z jakiej przyczyny). Po inicjalizacji systemu zmienna end_program jest zerowana, co jest rozumiane jako brak deklaracji ppodprogramu końcowego. Instrukcję KONIEC realizuje procedura endpgm(), zapisana w zbiorze endpgm.c86.

5.13. Instrukcja deklaracji podprogramu początkowego - POCZĄTEK.

W instrukcji POCZĄTEK deklarowany jest numer podprogramu, wywoływanego przy każdym rozpoczęciu automatycznego wykonywania programu użytkowego robota. Numer ten jest pobierany z ciała instrukcji, z pola instruction->prog_nr i wstawiany do zmiennej globalnej start_program. Zmienna ta jest wykorzystywana przez interpreter w sytuacji rozpoczęcia pracy automatycznej. Po inicjalizacji systemu zmienna start_program jest zerowana, co jest rozumiane jako brak deklaracji ppodprogramu początkowego. Instrukcję POCZĄTEK realizuje procedura strpgm(), zapisana

w zbiorze strpgm.c86.

5.14. Instrukcja pozycjonowania quasiliniowego - POZ QLIN.

Instrukcję tę realizuje procedura posq(), zapisana w zbiorze posq.c86. Na początku pobierana jest pozycja docelowa. Następnie obliczany jest zadany czas ruchu lub prędkość (w zależności od postaci instrukcji). Wartości te zapisane w ciele instrukcji muszą być przekształcone do odpowiedniej postaci (czas do sekund, prędkość musi być skorygowana wg współczynnika prędkości dla całego programu - ustawiony przez operatora funkcją +/- na panelu programowania). Następnie wywoływana jest procedura quasilin() (zapisana w pliku qasilin.c86 w katalogu INTMOV) realizująca ruch typu PTP synchroniczne do zadanej pozycji. Po powrocie z quasilin(), jeśli jej wykonanie przebiegło prawidłowo a instrukcja została zaprogramowana jako DOKŁADNIE, program przechodzi do oczekiwania na sygnał INPOS - potwierdzenie, że wszystkie osie weszły w sterfę zerową (procedura wait_for_inpos() w pliku wtfinpos.c86 w katalogu INTRPRTR). Następnie podprogram posq() kończy działanie przekazując do procedury wywołującej wartość zmiennej result - określa ona w tym przypadku jaki był skutek czekania na INPOS. Jeśli system wykrył błąd wykonania funkcji quasilin() lub pozycjonowanie jest typu ZGRUBNIE, to od razu następuje wyjście z procedury posq(), z przekazaniem do podprogramu wywołującego zmiennej określającej sposób wykonania ruchu - result.

5.15. Instrukcja pozycjonowania liniowego - POZ LIN.

Instrukcję tę realizuje procedura posl(), zapisana w zbiorze posl.c86. Na wstępie sprawdza ona czy zostały już w programie określone: prędkość bazowa i narzędzie. Jest to warunek, aby instrukcja pozycjonowania liniowego mogła się wykonać. W praktyce oznacza to, że w programie użytkowym przed instrukcją POS_LIN musi być wykonana instrukcja PRĘDKOŚĆ i instrukcja NARZĘDZIE. Następnie uaktualniana jest pozycja zewnętrzna robota (wywołanie procedury calc_ext() zapisanej w pliku calcext.c86 w katalogu EXTMOV - oblicza aktualną wartość T_POS - por. p. 6). Jest to zabieg niezbędny, umożliwiający efektywną realizację sekwencji instrukcji POZ_QLIN, POZ_LIN. Pierwsza z niech operuje na współrzędnych wewnętrznych i nie uaktualnia pozycji zewnętrznej. Druga uaktualnia obie pozycje. Następnie na podstawie parametrów pozycji docelowej zapisanych w ciele instrukcji obliczana jest macierz pozycji na końcu ruchu liniowego - T_ZAD. Porównując obie macierze startową i końcową (zadaną) oblicza się drogę TCP zadaną do przebycia. Na tej podstawie, uwzględniając zapisaną w ciele instrukcji prędkość ruch obliczany jest czas ruchu. Prędkość ruchu i czas jego trwania, jako zmienne globalne oraz aktualna i docelowa pozycja, jako argumenty formalne są danymi wejściowymi dla podprogramu obliczającego parametry generatora trajektorii prostoliniowej (procedura par_linear() w pliku parlin.c86, w katalogu EXTMOV - por. p.6). Następnie wywoływany jest podprogram realizujący ruch po linii prostej wg obliczonych parametrów (procedura linear(), w pliku linear.c86, w katalogu EXTMOV - por. p.6). Po jej zakończeniu, podobnie jak w realizacji POZ_QLIN sprawdzany jest ewentualnie warunek wejścia osi w strefę zerową i procedura posl() kończy działanie, zwracając do podprogramu wywołującego informację o rezultatach swego działania - zmienna result.

18

5.16. Instrukcja pusta.

Instrukcję tę realizuje procedura empty(), zapisana w zbiorze empty.c86. Instrukcja pusta wystepuje zawsze jako końcowa instrukcja programu głównego (jest to wiec znacznik końca programu). Wykonanie tej instrukcji polega na odszukaniu pierwszej instrukcji programu i rozpoczęciu wykonywania programu od początku.

5.17. Instrukcja neutralna

Instrukcję tę realizuje procedura noop(), zapisana w zbiorze noop.c86. Jest to procedura typu "nic nie rób". Została ona wprowadzona niejako na wszelki wypadek, gdyby w programie znalazła się instrukcja o nieznanym kodzie. Wtedy, jeśli w tablicy długości instrukcji dla tego kodu jest wprowadzona prawidłowa wartość, program będzie się wykonywał prawidłowo, z pominięciem instrukcji nieznanej. Mechanizm ten jest dość wygodny przy wprowadzaniu nowych instrukcji, gdy wiele zmiennych, procedur wspomagających, struktur przewidzianych do realizacji jest już zaimplementowanych, a sam podprogram wykonujący nową instrukcję nie jest jeszcze gotowy.

6. MODEL KINEMATYKI I GENERATOR TRAJEKTORII ROBOTA.

Procedury odpowiedzialne za przeliczanie modelu kinematycznego oraz generację trajektorii prostoliniowej zostały zgrupowane w katalogu EXTMOV. Model kinematyczny robota określa powiązanie między jego współrzędnymi wewnętrznymi a zewnętrznymi. Jako współrzędne wewnętrzne przyjmuje się kąty obrotu poszczególnych osi. Trzeba jednak pamiętać, że dolna warstwa sterowania robota, warstwa sterowników osiowych nie operuje miarą kątową położenia osi, lecz tzw. elementarnymi przyrostami położenia - inkrementami. Kątowa wartość jednego inkrementu zależy nie tylko od rozdzielczości elementu pomiarowego (w przypadku robota URP jest to rezolwer zamocowany na wale silnika), ale także od rodzaju i parametrów przekładni. Jako współrzędne zewnętrzne w robocie URP stosuje się współrzędne kartezjańskie do opisu położenia punktu roboczego narzędzia oraz kąty Eulera do opisu jego orientacji. Pozycję robota określa się w tzw. bazowym układzie współrzędnych związanym z podstawą robota (patrz Podręcznik programowania robotów URP). W obecnej wersji programu sterującego zaimplementowany jest model kinematyczny opracowany na podst. zasad sformułowanych w książce R. P. Paula "Robot Manipulators" (patrz zał. 1 do spraw. PIAP nr rej. 6637). Przyjęto w nim opis poszczególnych par kinematycznych zgodnie z notacją Denavita-Hartenberga. Na podstawie parametrów D-H tworzone są macierze przejścia między poszczególnymi członami robota. Po ich wymnożeniu otrzymuje się macierz 4x4 określającą pozycję i orientację (w bazowym układzie współrzędnych) układu związanego z kołnierzem piątej osi. Macierz ta, o nazwie T_MAC[4][4] jest w programie zmienną lokalną, gdyż ma ona tylko znaczenie pomocnicze, w pośrednich obliczeniach. Narzędzie, którym operuje robot jest uwzględniane w jego łańcuchu kinematycznym jako kolejny układ współrzędnych związany w sposób sztywny z kołnierzem piątej osi. Parametry narzędzia (pozycja TCP i orientacja w układzie współrzędnych piątej osi), wprowadzone przez użytkownika podczas definiowania narzędzia pozwalają stworzyć macierz przejście pomiędzy układem piątej osi a narzędziem. Macierz ta, o nazwie T_TOOL[4][4] jest w programie zmienną globalną. Iloczyn macierzy T_MAC i T_TOOL daje nam macierz pozycji narzędzia w układzie bazowym - T_POZ[4][4], która podobnie jak T_TOOL jest zmienną globalną. Obliczenie macierzy T_POZ stanowi rozwiązanie prostego zadania kinematycznego. Przejście z tej macierzy do konkretnych wartości współrzędnych pozycji narzędzia i kątów Eulera określających jego orientację jest już trywialne. Zabieg ten jest opisywany dokładnie w większości pozycji literatury traktujących o kinematyce robotów. W programie sterującym wykorzystano konkretne wzory przedstawione we wspomnianej już książce R. P. Paula. Zadanie odwrotne polega na wyznaczeniu, na podstawie pozycji zewnętrznej, pozycji wewnętrznej robota, a więc wartości kątów obrotu w poszczególnych osiach. Konkretnie w przypadku robota URP wynikiem powinna być pozycja w inkrementach. Jako dana wejściowa jest natomiast przyjmowana macierz T_POZ.

Poniżej przedstawiono zadnaia poszczególnych procedur znajdujących się w katalogu EXTMOV (w porządku alfabetycznym plików).

Procedura calc_ext() w pliku calcext.c86:

Obliczenie pozycji zewnętrznej robota (macierz t_pos) oraz cosinusów i sinusów kątów osi na podstawie pozycji wewnętrznej (wyrażonej w inkrementach).

Procedura deftool() w pliku deftool.c86:

Definicja aktualnego narzędzia, według którego jest wykonywany ruch we współrzędnych zewnętrznych. Funkcja oblicza wartość T_TOOL dla aktywnego narzędzia.

Procedura dtoi() w pliku dtoi.c86:

Pomocnicza funkcja przekształcająca liczbę typu double na typ int.

Procedura dtol() w pliku dtol.c86:

Pomocnicza funkcja przekształcająca liczbę typu double na typ long.

Plik extconst.c86:

Stałe globalne wykorzystywane w katalogu EXTMOV.

Procedura initextm() w pliku initextm.c86:

Inicjalizacja zmiennych i procedur wykonujących ruch we współrzędnych zewnętrznych.

Procedura in_to_rad_irp_6() w pliku intorad.c86:

Procedura zamiany inkrementów na radiany.

Procedura it_back_irp6() w pliku inwers.c86:

Obliczenie pozycji wewnętrznej robota (położenie poszczególnych osi w inkrementach - pozycja w układzie wewnętrznym) oraz cosinusów i sinusów osi na podstawie pozycji zewnętrznej (macierz t_pos oraz sinusy i cosinusy pozycji wyjściowej - poprzedniej).

Procedura it_back_irp6() w pliku it_back1.c86:

Obliczanie sinusów i cosinusów kątów w poszczególnych osiach robota URP-6 na podstawie macierzy pozycji układu współrzędnych związanego z kołnierzem piątej osi.

Procedura linear() w pliku linear.c86:

Wykonanie ruchu po linii prostej z jednostajną zmianą orientacji osi narzędzia.

Procedura make_pp() w pliku make_pp.c86:

Generator trajektorii prostoliniowej. Obliczenie pozycji pośredniej na trajektorii prostoliniowej na podstawie parametrów ruchu liniowego i wielkości drogi juz przebytej

Procedura move() w pliku move.c86:

Ruch elementarny do pozycji wyrażonej we współrzędnych zewnętrznych. Ruch jest wykonywany w ten sposób, że inkrementy, przeprowadzające ramię robota do zadanej pozycji wyrażonej we współrzędnych wewnętrznych, są wysyłane równomiernie na wszystkie osie (odpowiednik PTPL w IRb-6/60).

Procedura make_urp6_mac2() w pliku m_urp6_2.c86:

Obliczanie macierzy pozycji kołnierza robota URP-6 na podstawie

sinusów i cosinusów kątów w osiach - wykorzystanie przekształconych wzorów

Procedura par_linear() w pliku parlin.c86:

Generator trajektorii prostoliniowej. Obliczenie parametrów dla generatora linii prostej na podstawie pozycji początkowej i końcowej ruchu prostoliniowego. Parametry te są wykorzystywane w procedurze make_pp().

Procedura rad_to_in_irp_6() w pliku radtoin.c86:

Procedura przeliczania radianów na inkrementy.

Procedura stinterp() w plik stinterp.c86:

Ustawienie zmiennej calcper - okres makrointerpolacji (w ms). W obecnej wersji możliwe są cztery wartości: 8, 16, 32, 64ms. Wynika to z możliwości dolnej warstwy sterowania - cyfrowych sterowników osi MV20.

SPIS TRESCI