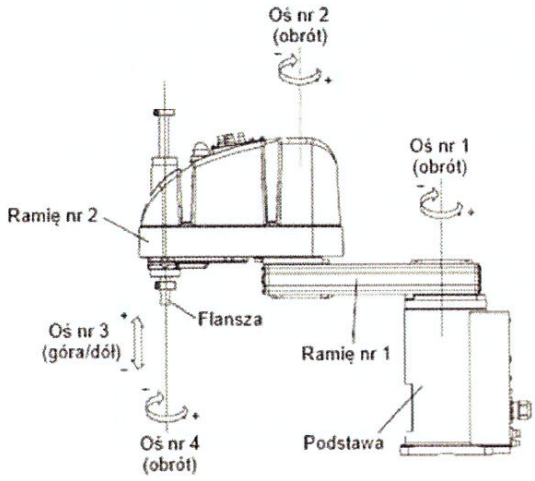
Robot EPSON SCARA T3-401S

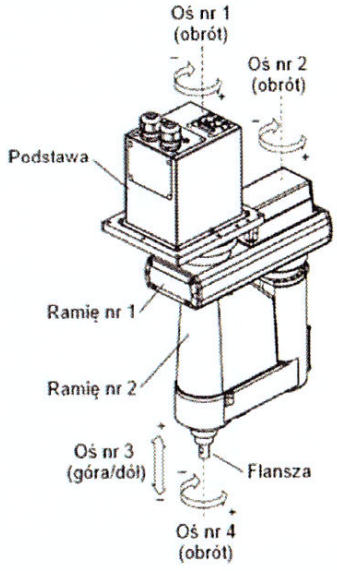
Parametry podstawowe		
Udźwig [kg]		1/3
Ilość osi		4
Zasięg	horyzontalny [J1+J2] [mm]	400 [225+175]
	wertykalny [J3] [mm]	150
	orientacja [J4] [°]	+/–360
Powtarzalność	horyzontalna [J1+J2] [mm]	+/–0,02
	wertykalna [J3] [mm]	+/–0,02
	orientacja [J4] [°]	+/–0,02
Moment bezwładności nom./max. [kgm^2]		0,003/0,01
Siła wzdłuż osi Z [N]		Stała 89
Ciężar [kg]		16
Sposób montażu		Podłogowy
Kontroler		Zintegrowany

Napędy robota wykorzystują silniki AC, a pomiar położenia realizowany jest za pomocą enkoderów przyrostowych.

Osie robota

W zależności od typu roboty Epson Scara posiadają 3 lub 4 osie, które umożliwiają wykonywanie następujących ruchów:





Układ współrzędnych

Położenie elementów może być rozpatrywane w jednym z trzech rodzajów układów współrzędnych:

układ współrzędnych robota – zwany również bazowym (base) związany jest z podstawą robota i ma swój początek w osi podstawy na wysokości górnego położenia flanszy.

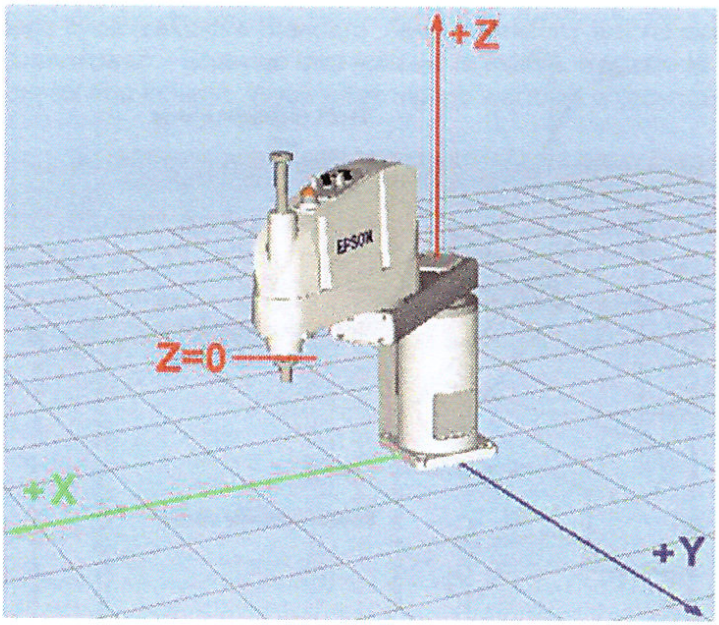
układ współrzędnych narzędzia (tool) – związany jest z narzędziem zamontowanym na flanszy robota, a jego początek znajduje się w ostatniej osi robota.

lokalny układ współrzędnych – dowolny system zdefiniowany przez użytkownika; jego początek może znajdować się w dowolnym punkcie obszaru pracy robota.

Zdefiniować można 15 różnych lokalnych układów współrzędnych; definiuje się je w oknie Robot Manager na karcie Locals lub używając polecenia języka SPEL+ Local, np. Local 1, XY(x, y, z, u) .

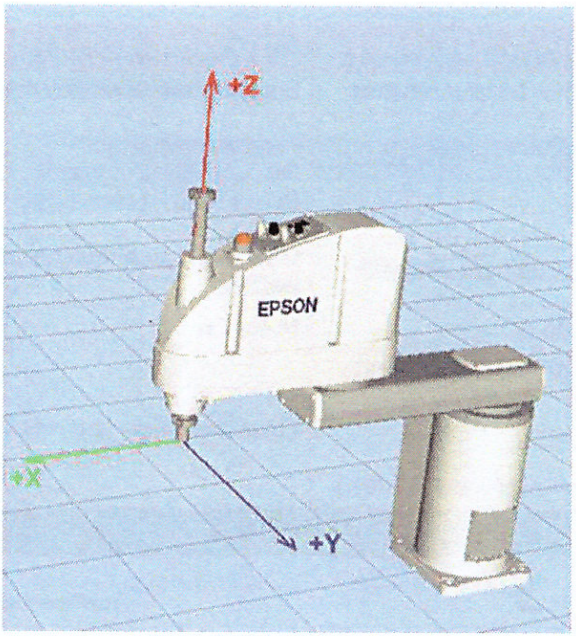
Układ współrzędnych

Local coordinate system:



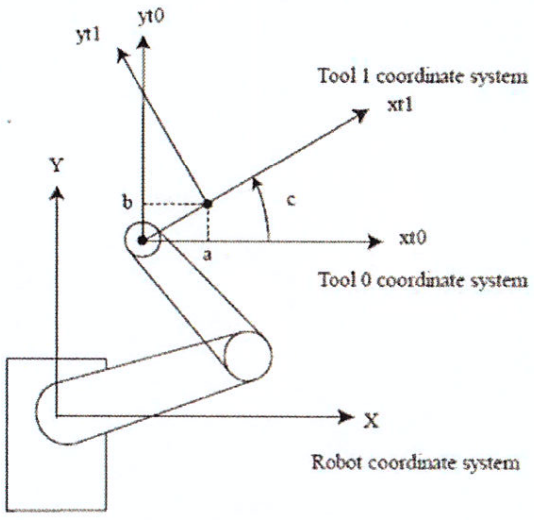
Układ współrzędnych

Tool coordinate system:



Układ współrzędnych

Gdy ostatnia oś ustawiona jest na 0 stopni, to osie X i Y układu narzędzia są równoległe do osi X i Y układu bazowego:



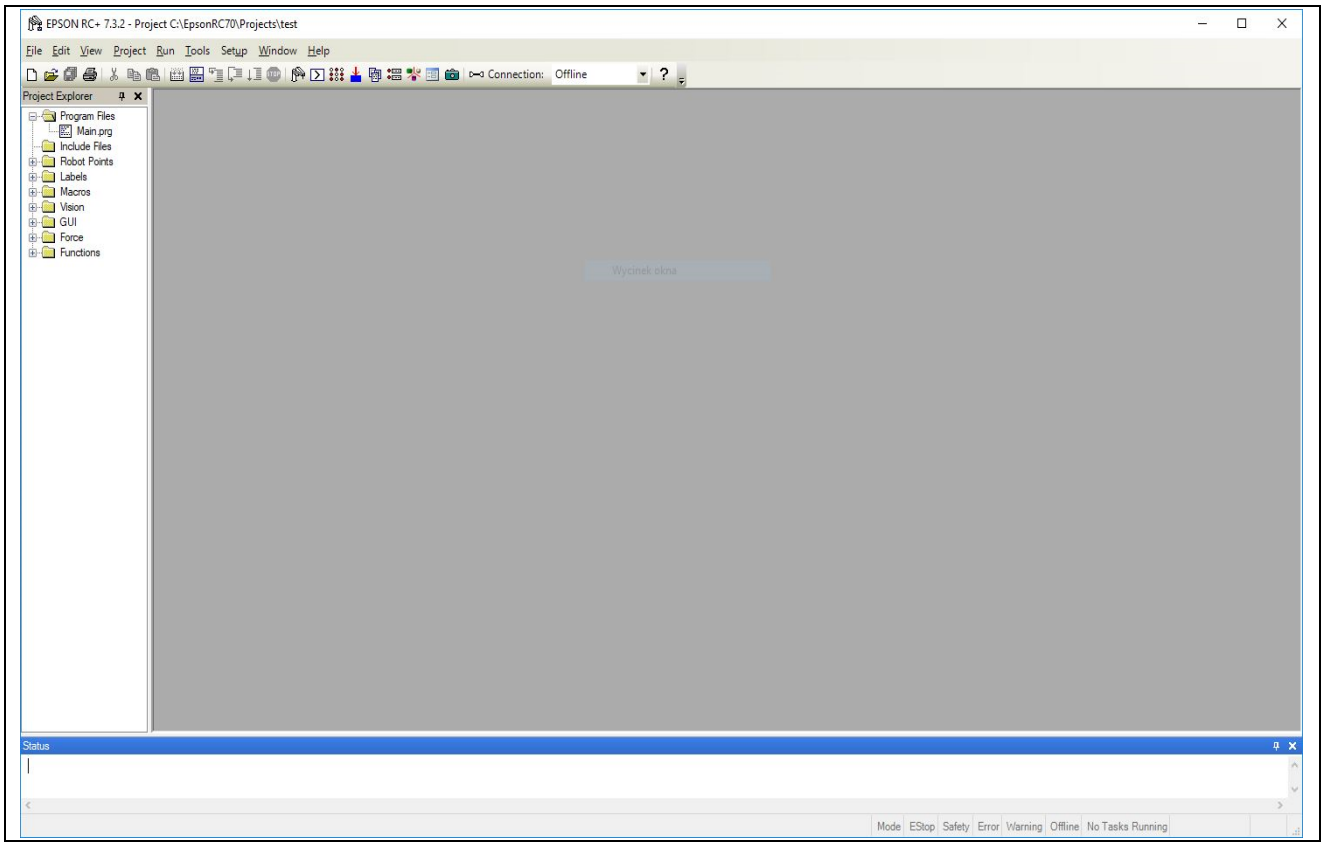
Układ narzędzia obraca się wraz z obrotem ostatniej osi robota.

Przebieg ćwiczenia:

Celem ćwiczenia jest zaznajomienie się ze środowiskiem EPSON RC+ a następnie stworzenie programu powodującego ruch robota pomiędzy punktami z wykorzystaniem języka SPEL+
Na wykonanie ćwiczenia składają się następujące etapy:

1. nawiązanie połączenia z robotem
2. uruchomienie napędów
3. pozycjonowanie poszczególnych ramion robota z poziomu ***Robot Managera***
4. napisanie programu realizującego ruch robota w języku SPEL+

Środowisko EPSON RC+ 7.3.2



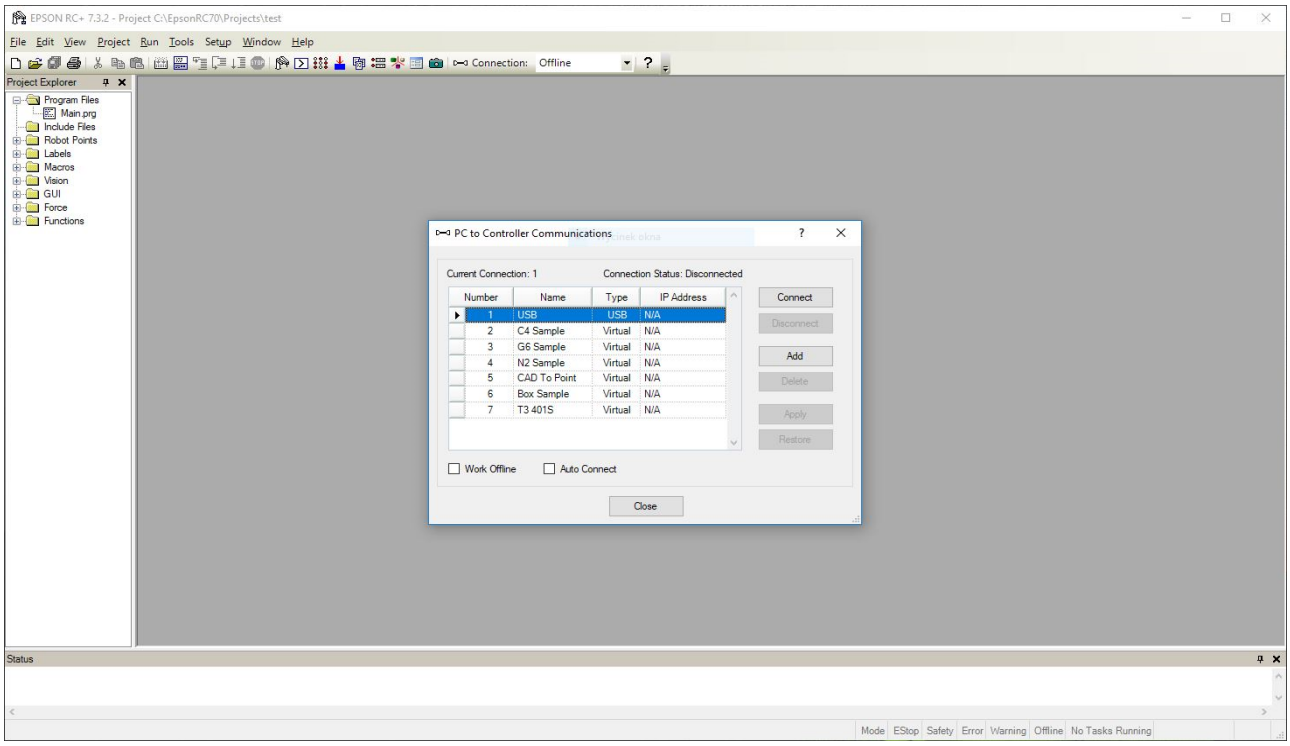
Rys.1 Okno główne programu EPSON RC+

Narzędzie EPSON RC+ jest środowiskiem programistycznym umożliwiającym programowanie robotów EPSON.
Programowanie realizowane jest metodą uczenia, na którą składają się dwa etapy:

- nauczenie i zapamiętanie punktów w przestrzeni roboczej, w których ma zostać spozycjonowany robot
- napisanie programu ruchu robota pomiędzy punktami za pomocą poleceń języka programowania SPEL+.

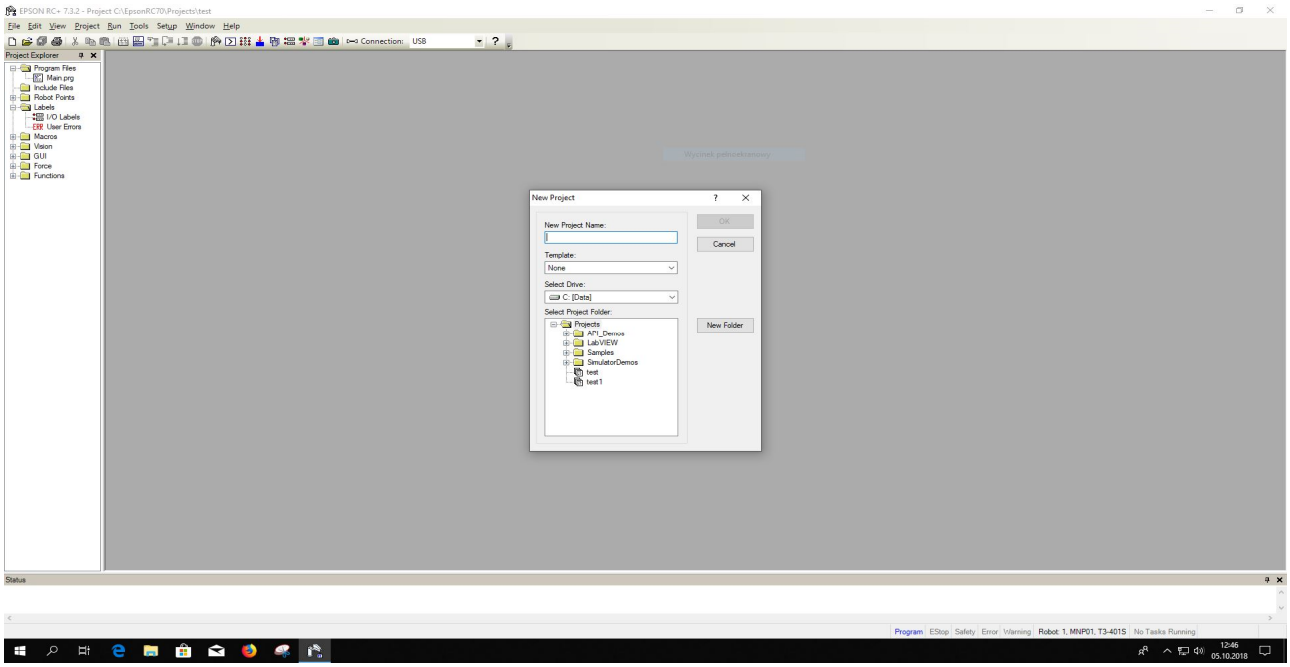
Robot EPSON T3-401S łączony jest z komputerem PC poprzez port USB.

1. Po uruchomieniu programu EPSON RC+ pojawia się okno pokazane na Rys. 1.
Dla uzyskania połączenia z robotem należy wybrać: ***Setup/ PC to Controller Communication***.
W otwartym oknie, pokazanym na Rys. 2 , wybrać ***USB*** i nacisnąć ***Connect***.
Po uzyskaniu połączenia zamknąć okno: ***Close***.



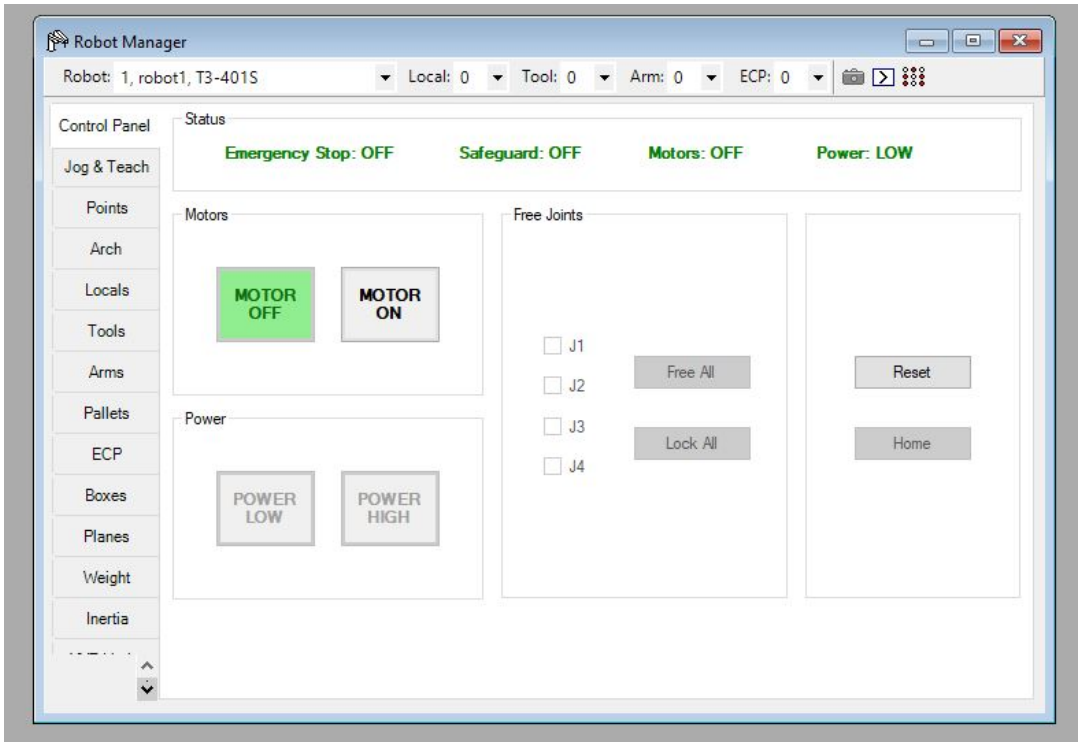
Rys.2 Okno *PC to Controller Communication*

2. Aby uzyskać dostęp do wszystkich opcji środowiska wybrać: ***Project/New*** i nadać nazwę projektu (Rys. 3).



Rys.3 Okno *Project/New*

3. Aby uruchomić napędy robota wybieramy: ***Tools/Robot Manager*** (Rys.4).
Przyciski ***MOTOR OFF/MOTOR ON:*** służą do włączania i wyłączania napędów.
POWER LOW/POWER HIGH: wybieranie trybu pracy; *POWER LOW*: ograniczona prędkość ruchu wykorzystywana podczas uczenia i testowania (ustawiana defaultowo); *POWER HIGH:* możliwość uzyskania pełnych prędkości.
Przycisk ***Free All***: zwolnienie hamulców wszystkich osi (brak możliwości wykonania ruchu).
Przycisk ***Lock All***: włączenie hamulców.
Przycisk ***Reset***: resetowanie kontrolera robota.
Przycisk ***Home***: ustawianie robota w pewnej, zdefiniowanej przez użytkownika, pozycji.



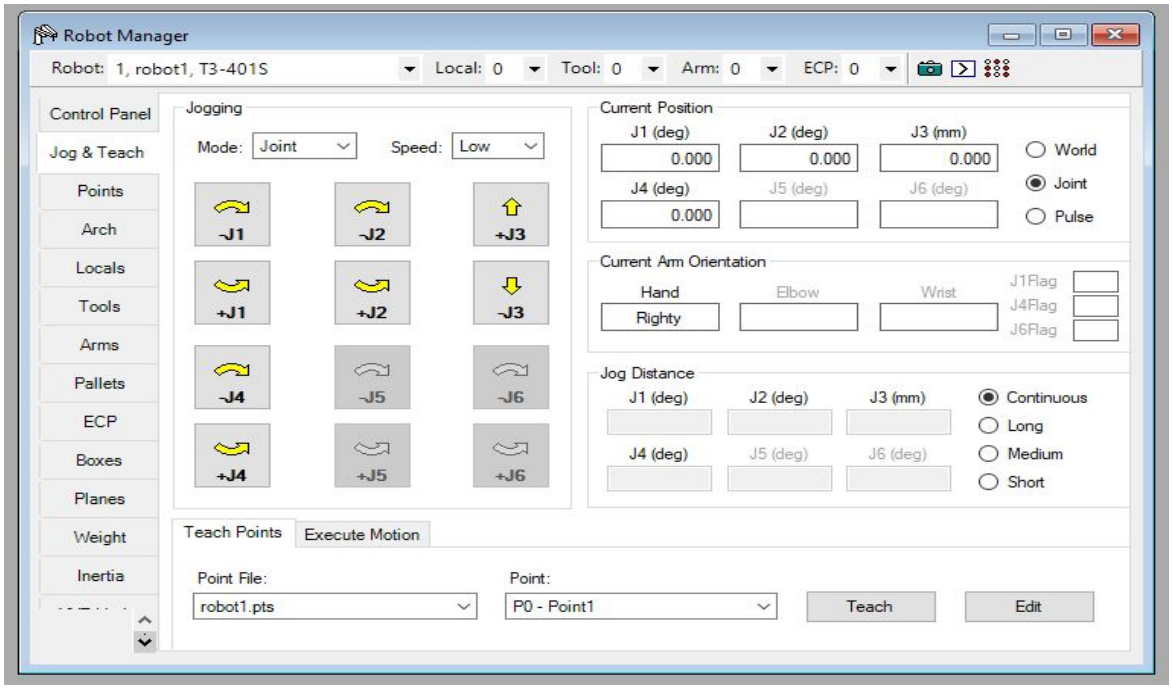
Rys.4 Okno *Tools/Robot Manager*

4. Proces uczenia punktów: w ***Robot Manager*** wybieramy zakładkę ***Jog&Teach*** (*Teach Points*) (Rys.5).

Mode: wybór układu współrzędnych, w którym uczymy robota: *Point* (układ złączowy), *World* (układ kartezjański), *Tool* (układ narzędzia), *Local* (układ lokalny)
Speed: ustawianie prędkości ruchu w trakcie uczenia (*Low, High*)
Current Position: wyświetlanie współrzędnych punktu (w układzie: *Joint, World lub Puls*)
Current Arm Orientation: pokazuje aktualną konfigurację ramion 1 i 2 (*Righty, Lefty*)
Jog Distance: ustawianie sposobu ruchu podczas uczenia: *Continuous* (ciągły), *Long* (skokowo co 10 mm), *Medium* (skokowo co 1 mm), *Short* (skokowo co 0.1 mm)

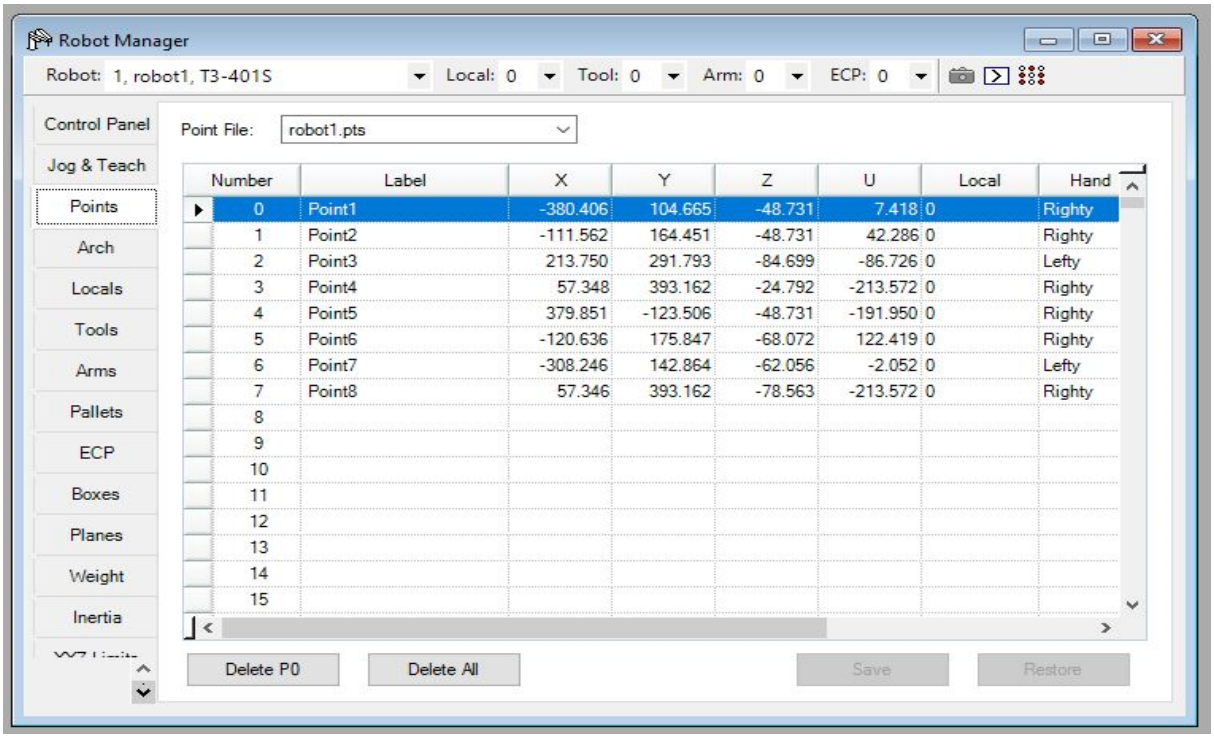
Używając przycisków ***J1, J2, J3 i J4*** ustawiamy punkt w przestrzeni roboczej, następnie z rozwijanego menu wybieramy numer punktu (***Point***) i naciskamy przycisk ***Teach***. Pojawia się okno, w którym należy nadać nazwę dla danego punktu, pod jaką będzie identyfikowany w projekcie.
Wszystkie nauczone punkty są zapisywane w pliku *robot1.pts*.

Gdy chcemy mieć inny zbiór punktów to wybieramy (w **EPSON RC+**) ikonę ***New***, a następnie w pojawiającym się oknie wybieramy z rozwijanego menu ***Point*** i nadajemy nazwę: *nazwa.pts*.
W polu ***Point File*** w rozwijanym menu pojawią się różne zbiory punktów.

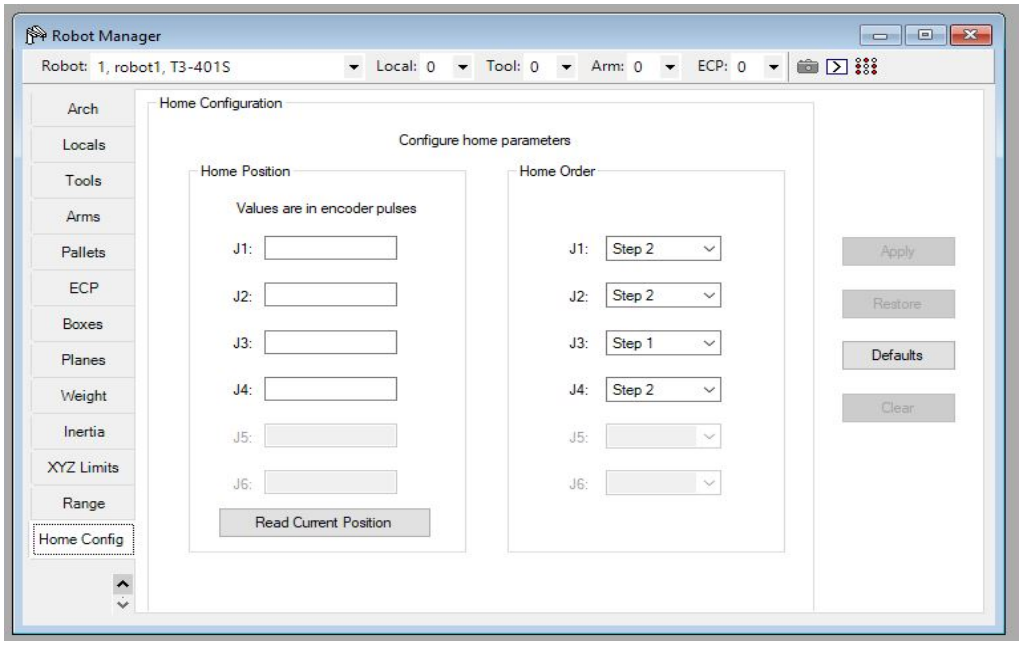


Rys.5 Okno *Robot Manager/Jog&Teach (zakładka Teach Points)*

Wybierając w ***Robot Manager*** zakładkę ***Points*** można zmieniać konfigurację ramion 1 i 2 dla zapisanych punktów (kolumna *Hand*).



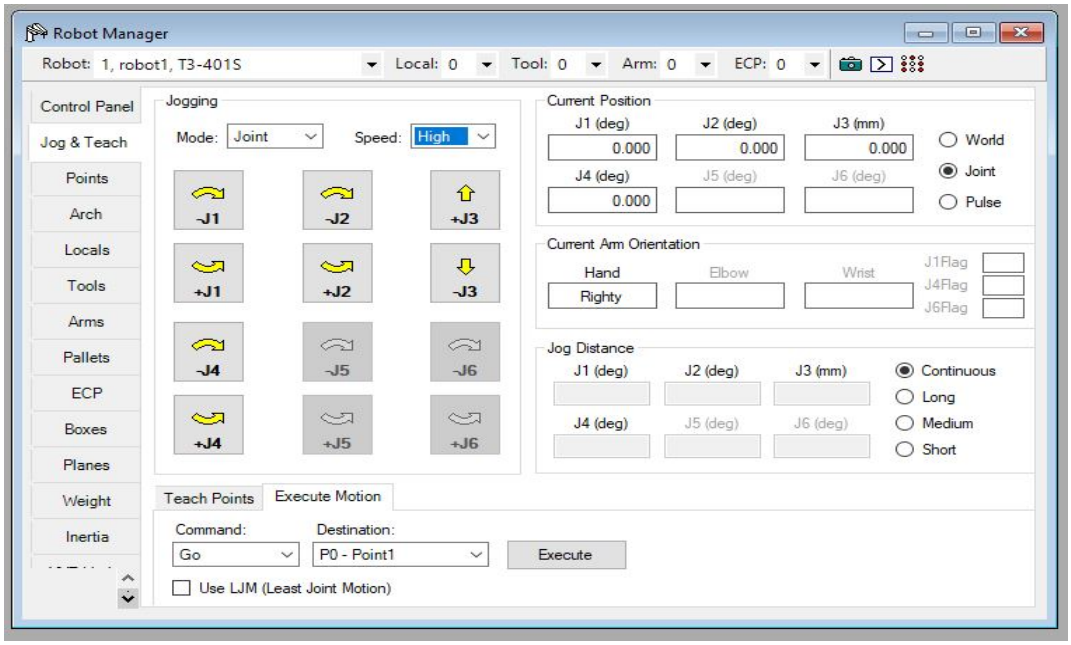
Aby zdefiniować pozycję HOME należy w ***Tools/Robot Manager*** otworzyć zakładkę ***Home Config*** (Rys.6). Ustawić robota w pozycji, która ma być pozycją HOME, wczytać pozycję (przycisk ***Read Current Position***) i w kolumnie ***Home Order*** ustawić kolejność wykonywania ruchów przez osie robota przy dochodzeniu do ustawienia *HOME*. Pozycja HOME jest podawana w impulsach z enkoderów.



Rys.6 Okno *Home Configuration*

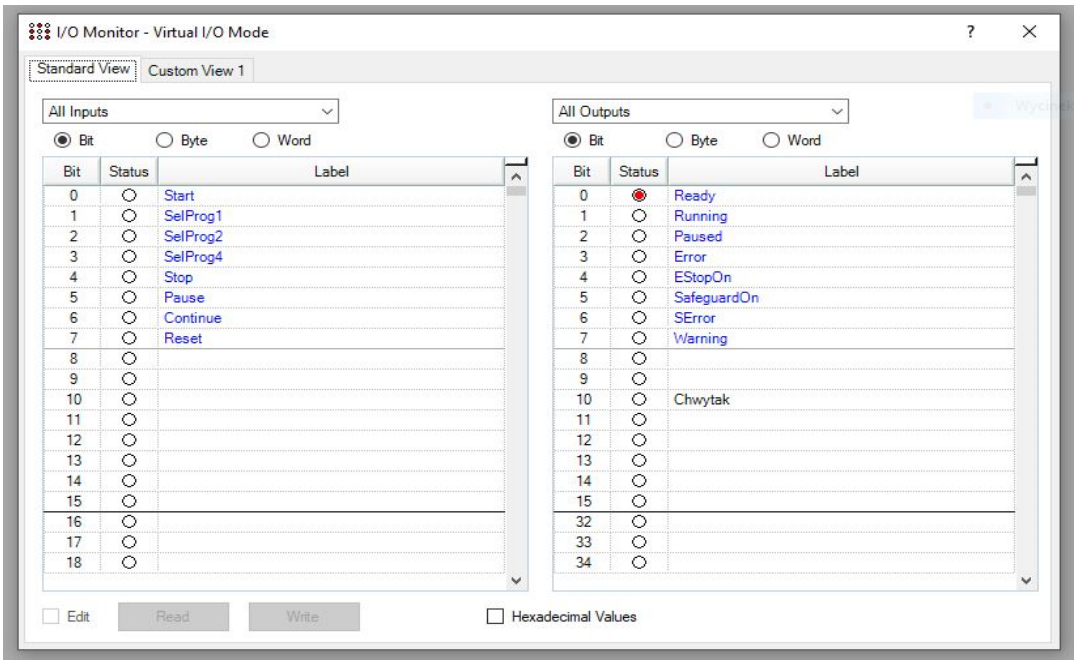
5. Testowanie sposobów ruchu.

Wybierając w ***Jog&Teach*** zakładkę ***Execute Motion*** (Rys.7) można przetestować sposoby ruchu między nauczonymi punktami. W polu ***Destination*** z rozwijanego menu wybieramy punkt do którego ma być zrealizowany ruch, a następnie w polu ***Command*** z rozwijanego menu wybieramy jeden ze zdefiniowanych dla robota sposobów ruchu: ***Go*** (ruch synchroniczny PTP w układzie złączowym), ***Jump*** (przy rozpoczęciu ruchu oś 4 jest podnoszona do pozycji zerowej, przy zakończeniu oś 4 ustawiana jest w pozycji dla danego punktu; reszta ruchu jak w Go), ***Move*** (ruch po linii prostej w układzie kartezjańskim), ***CRC*** (ruch po okręgu w układzie kartezjańskim).
Po naciśnięciu ***Execute*** ruch zostanie wykonany (pod warunkiem, że jest on fizycznie możliwy do wykonania między zadanymi punktami).
W przypadku wystąpienia błędu, wynikiem którego jest blokada napędów, należy zresetować kontroler (w *Control Panel*, przycisk *Reset*).



Rys.7 Okno *Robot Manager/Jog&Teach (zakładka Execute Motion)*

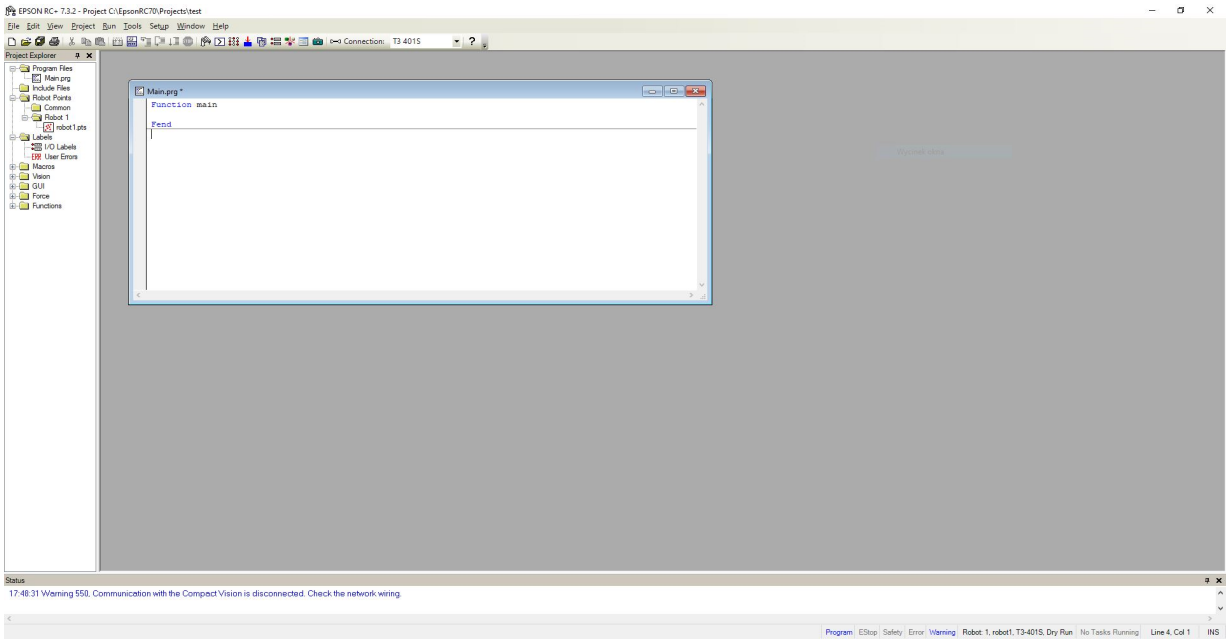
Robot wyposażony jest w chwytak pneumatyczny podpięty do wyjścia cyfrowego Out 10 (***Tools/IO Monitor/Outputs***) (Rys.8). Aby dostęp do chwytaka był możliwy w projekcie należy go zdefiniować nadając nazwę dla bitu Out 10. Nadawanie nazwy dla wyjść odbywa się w zakładce ***Tools/IO Label Editor/Outputs***) np. Chwytak Po zdefiniowaniu dostęp do chwytaka jest realizowany za pomocą poleceń języka SPEL+ (***On*** i ***Off***).



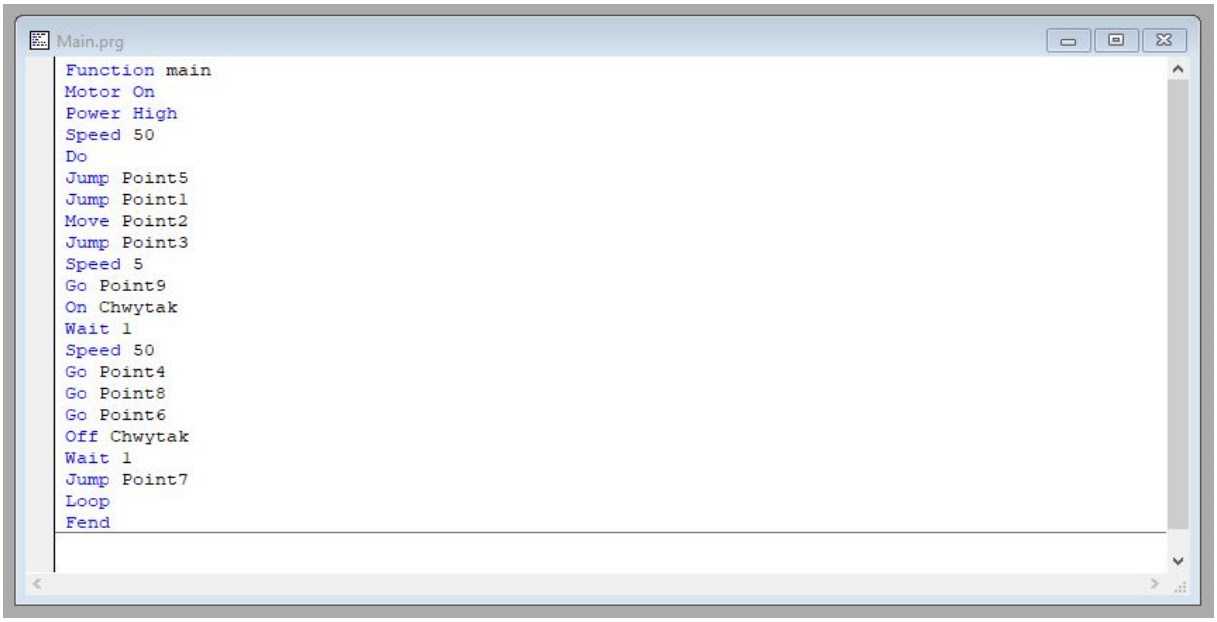
Rys.8 Definiowanie chwytaka w systemie

6. Edycja programu.

Po zdefiniowaniu punktów i przetestowaniu ruchów można przystąpić do edycji programu. Aby otworzyć okno edycji programu należy w ***Project Explorer*** dwukrotnie kliknąć na ***main.prg*** (Rys.9).
Pomiędzy dwie linie widoczne w oknie wpisuje się kolejne linie poleceń języka SPEL+.



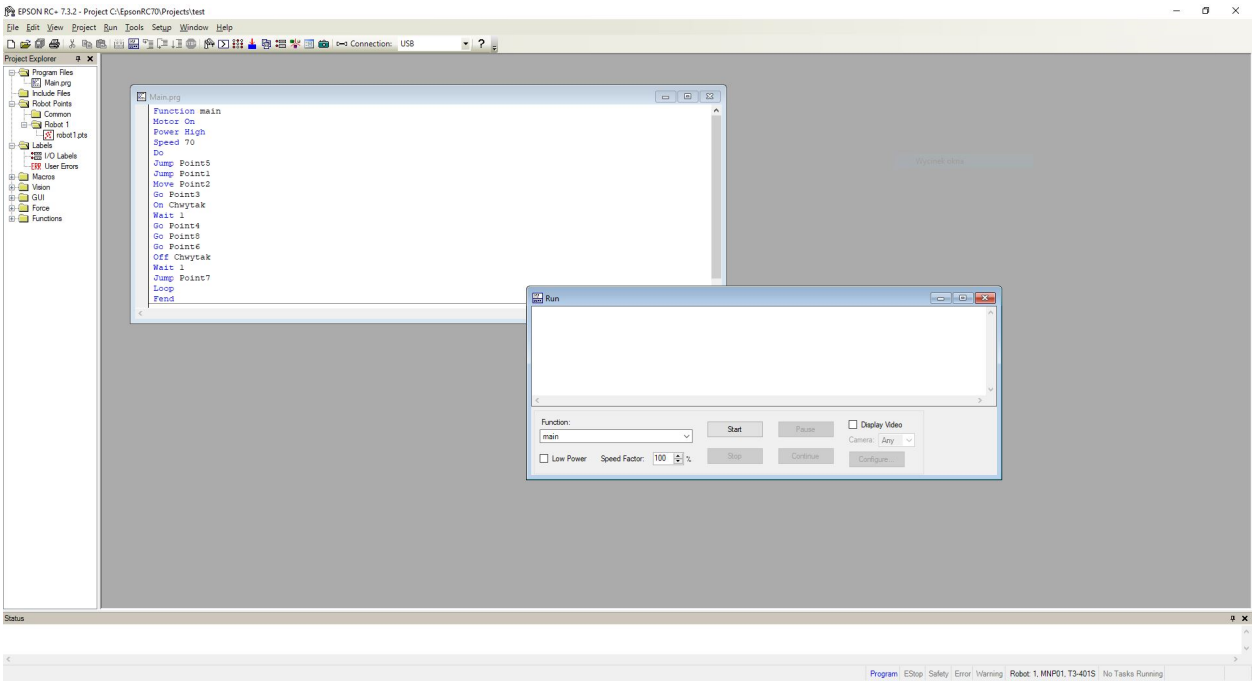
Rys.9 Okno edycji programu



Rys.10 Przykładowy program ruchu w nieskończonej pętli

7. Uruchomienie programu.

Uruchomienie programu następuje po wybraniu ***Run/Run Window***. W pojawiającym się oknie (Rys.11) dostępne są przyciski ***Start/Stop*** do uruchomienia i zatrzymania programu, ***Pause/Continue*** do wstrzymania i kontynuowania programu.



Rys.11 Okno uruchomienia programu

Jeżeli w trakcie wykonywania programu pojawi się błąd skutkujący blokadą napędów należy zatrzymać program (przycisk ***STOP***) a następnie zresetować kontroler (***Tools/Robot Manager/Reset***).

! Po zakończeniu pracy z robotem należy wyłączyć napędy robota (***Tools/Robot Manager/MOTOR OFF***) oraz rozłączyć komunikację z komputerem – w tym celu należy z menu głównego wybrać ***Setup/ PC to Controller Communication)*** po czym nacisnąć ***Disconnect***.

Język SPEL+

Program w języku SPEL+ jest zbiorem funkcji, zmiennych i makr.
Jest to język proceduralny składnią przypominający język C.

Tryb zmiennych w SPEL+

W języku SPEL+ występują następujące typy zmiennych:

Typ danych	Rozmiar	Zakres
Boolean	2 bajty	Prawda lub Fałsz
Byte	2 bajty	-128 do +127
Double	8 bajtów	-1.79E+308 do +1.79E+308 (14 cyfr znaczących)
Integer	2 bajty	-32768 do +32767
Long	4 bajty	-2147483648 do +2147483647
Real	4 bajty	-3.4E+38 do +3.4E+38 (6 cyfr znaczących)
String	256 bajtów	Wszystkie znaki ASCII (255 znaków)

Operatory w SPEL+

W języku SPEL+ występują następujące operatory:

+	dodawanie	>=	większość lub równość
-	odejmowanie	<=	mniejszość lub równość
*	mnożenie	<>	różność
/	dzielenie	And	iloczyn logiczny
**	potęgowanie	Mod	reszta z dzielenia
=	równość	Not	logiczna negacja
>	większość	Or	suma logiczna
<	mniejszość	Xor	alternatywa wykluczająca

Zmienne w SPEL+

W języku SPEL+ występują trzy typy zmiennych:

- **lokalne**
- **modułowe**
- **globalne**

Zmienne lokalne – ich zasięg ogranicza się tylko do wnętrza funkcji, w której zostały zadeklarowane; w różnych funkcjach mogą występować zmienne o tej samej nazwie, jednak nie wpływają one na siebie z uwagi na ograniczony zasięg.

Zmienne lokalne deklaruje się na początku ciała funkcji podając ich typ i nazwę.

Przykład:

```
Function test
    Integer zmienna1
    Real zmienna2
    ...
```

Zmienne w SPEL+

Zmienne modułowe – swoim zasięgiem obejmują wszystkie funkcje wewnątrz pliku z programem.

Zmienne modułowe deklaruje się na początku pliku z programem przed definicjami funkcji, podając ich typ i nazwę. Aby odróżnić ten rodzaj zmiennych od pozostałych, nazwę zmiennej poprzedza się znakami m_

Przykład:

```
Integer m_zmienna1
String m_zmienna2
Function main
    ...
Fend
```

Zmienne globalne – swoim zasięgiem obejmują wszystkie funkcje wewnątrz projektu.

Zmienne globalne deklaruje się na początku pliku z programem przed definicjami funkcji, podając słowo Global a następnie ich typ i nazwę. Aby odróżnić ten rodzaj zmiennych od pozostałych, nazwę zmiennej poprzedza się znakami g

Zmienne w SPEL+

Przykład:

Plik main.prg:

```
Global Integer g_zmienna1
Function main
    ...
    Call funkcja2
Fend
```

Plik fun2.prg:

```
Function funkcja2
    ....
Fend
```

Szczególnym przypadkiem zmiennych globalnych są zmienne Global Preserve, które przechowywane są w pamięci SRAM kontrolera, ich deklaracja jest następująca:

```
Global Preserve Integer g_zmienna1
```

Tablice w SPEL+

Dla wszystkich rodzajów zmiennych (lokalnych, modułowych i globalnych) oraz typów (Boolean, Byte, Double, Integer, Long, Real, String) można tworzyć tablice o wymiarach 1, 2 lub 3; sposób deklaracji jest następujący:

typ_danych nazwa(rozm1 [,rozm2 [,rozm3]])

Przykłady:

Integer position(10)

Real coords(20,20,20)

Pierwszy element tablicy ma indeks równy 0 (zero)

Maksymalny rozmiar tablicy lokalnej lub global preserve typu łańcuchowego wynosi 100, a dla wszystkich pozostałych typów 1000.

Maksymalny rozmiar tablicy modułowej lub globalnej typu łańcuchowego wynosi 1000, a dla wszystkich pozostałych typów 10000.

Aby zmienić rozmiar tablicy w trakcie wykonywania programu należy użyć polecenia Redim

Redim position(20)

Typy zmiennych w SPEL+

Mamy do dyspozycji następujące typy zmiennych:

-Integer – liczby całkowite16 bit

-Real – liczby zmiennoprzecinkowe 32 bit

-Boolean – zmienne boolowskie (0 ,1)

-Long – 32 bit

-String – ciągi znaków, ASCII (do 255)

-Double – 64 bit, max 14 znaków

-Preserve – wartość zmiennej jest przechowywana w pamięci kontrolera

Ponadto zmienne możemy deklarować jako lokalne lub globalne:

Np.: ***Global Preserve String zmienna$***

Poniżej przedstawione zostały wybrane polecenia języka SPEL+.

Instrukcja FOR...NEXT

Instrukcja For … Next ma następującą składnię:

```
For var = initValue To finalValue [Step increment ]
    statements
Next [var]
```

var – zmienna wyliczeniowa, zazwyczaj typu całkowitego (integer), ale może też być zmienną rzeczywistą (real)

finalValue – wartość końcowa zmiennej wyliczeniowej; gdy zmienna wyliczeniowa osiągnie wartość końcową, pętla For … Next jest kończona i wykonywane są instrukcje znajdujące się po słowie Next

increment – parametr określający przyrost zmiennej wyliczeniowej w kolejnych krokach; przyrost może być dodatni lub ujemny, ale w tym drugim przypadku wartość początkowa musi być większa od końcowej; w przypadku nie podania parametru increment przyjmowana jest jego domyślna wartość wynosząca 1

Instrukcja FOR...NEXT

Instrukcja For ... Next służy do wykonywania zestawu poleceń określoną ilość razy.

Jeśli przyrost ma wartość dodatnią to kolejne kroki pętli wykonywane są dopóki wartość aktualna zmiennej wyliczeniowej nie przekracza wartości końcowej, natomiast jeśli przyrost jest liczbą ujemną, to pętla wykonywana jest do momentu, aż zmienna wyliczeniowa nie przestanie być większa od wartości końcowej.

Przykład użycia:

```
For counter = 1 To 10
        Go Pctr
Next counter
For counter = 10 to 1 Step -1
        Go Pctr
Next counter
```

Pętla DO...LOOP

Pętla Do ... Loop może mieć jedną z dwóch składni:

```
Do [{While | Until} condition]
        [statements]
[Exit Do]
        [statements]
Loop
```

```
Do
        [statements]
[Exit Do]
        [statements]
Loop [{While | Until} condition]
```

condition – wyrażenie, którego wartość logiczna wynosi True (prawda) lub False (fałsz); w przypadku gdy wartości warunku nie można ustalić (Null) wartość przyjmowana jest jako False

statements – wyrażenia lub funkcje, które będą wykonane w przypadku spełnienia warunku (True)

Pętla DO...LOOP

Pętla Do ... Loop wykonywana jest cyklicznie dopóki warunek condition jest spełniony lub dopóki nie zostanie napotkane polecenie Exit Do

W przypadku pominięcia warunku polecenie jest pętlą nieskończoną (może jednak zostać przerwana przez wystąpienie Exit Do lub inne zdarzenia takie jak Reset, Halt, inne).

Przykład użycia:

```
Do While Not Lof(1)
        Line Input #1, tLine$
        Print tLine$
Loop
```

Instrukcja SELECT

Instrukcja Select ma następującą składnię:

```
Select selectExpr
    Case caseExpr
        statements
    [Case caseExpr
        statements]
    [Default
        statements]
Send
```

selectExpr – wyrażenie typu liczbowego lub znakowego

caseExpr – wyrażenie liczbowe lub łańcuch znaków będące tego samego typu co selectExpr

statements – wyrażenia lub funkcje, które będą wykonane w przypadku gdy caseExpr w którym są zagnieżdżone ma tę samą wartość co selectExpr

Instrukcja SELECT

Polecenie sprawdza wartość wyrażenia selectExpr i przyrównuje ją po kolei do wartości poszczególnych wyrażeń caseExpr; w przypadku stwierdzenia równości dalsze sprawdzanie kolejnych warunków jest przerywane i wykonywane są instrukcje (statements) zagnieżdżone w bieżącym warunku (dla którego nastąpiło pozytywne porównanie).

Jeśli żaden z warunków Case nie jest spełniony (żadne z wyrażeń caseExpr nie jest równe wyrażeniu selectExpr) wykonywane są instrukcje zagnieżdżone wewnątrz Default.

Polecenie musi zawierać przynajmniej jeden warunek Case, natomiast pozostałe warunki, jak również Default są opcjonalne.

Instrukcja SELECT

Przykład użycia:

```
Select N                          'sprawdzaj zmienna N
    Case 0                        'gdy jest rowna 0
        Off 1;On 2;Jump P1
    Case 3                        'gdy jest rowna 3
        On 1;Off 2
        Jump P2;Move P3;On 3
Case 7                            'gdy jest rowna 7
        On 4
Default                           'gdy żaden z powyzszych warunkow
        On 7                      'nie jest spelniony (default)
 Send
```

Instrukcja warunkowa IF

Instrukcja If ma następującą składnię:

```
If condition1 Then
    stmtT1
[ElseIf condition2 Then]
    stmtT2
[Else]
    stmtF
EndIf
```

condition – wyrażenie, którego wartość logiczna wynosi True (każde wyrażenie o wartości różnej od zera) lub False (wyrażenie o wartości równej zero)

stmtT – instrukcje wykonywane gdy warunek condition jest prawdziwy

stmtF – instrukcje wykonywane gdy warunek condition jest fałszywy

Instrukcja warunkowa IF

Polecenie If/Then/Else może również być zapisane w nie w postaci blokowej, tylko w jednej linii.

If condition Then stmtT1 [; stmtT2...] [Else stmtF1 [; stmtF2...]]

W przypadku użycia powyższej formy zapisu znacznik zamykający EndIf nie jest potrzebny.

Jeśli warunek (condition1) jest spełniony wykonywane są polecenia występujące bezpośrednio po tym warunku, natomiast jeśli warunek nie jest spełniony to sprawdzany jest kolejny warunek (condition2), itd.

Jeśli żaden z warunków nie jest spełniony to wykonywane są polecenia występujące po słowie Else.

Dodatkowe warunki (condition2 itd.) oraz Else są opcjonalne i nie muszą występować w instrukcji If.

Polecenie If/Then/Else może być wielokrotnie zagnieżdżane wewnątrz innego polecenia If/Then/Else, jak również innych instrukcji (Select, Do ... Loop, For).

Instrukcja warunkowa IF

Przykład użycia:

```
If Sw(1) = 1 Then
        If Sw(2) = 1 Then
                Print "Input1 On and Input2 ON"
        Else
                Print "Input1 On and Input2 OFF"
        EndIf
Else
        If Sw(2) = 1 Then
                Print "Input1 Off and Input2 ON"
        Else
                Print "Input1 Off and Input2 OFF"
        EndIf
EndIf
```

Polecenie POWER

Power {Low | High}

Low – włącza tryb Low Power, w którym moc ograniczona jest do 30%, prędkość do 250mm/s, a sztywność serwomechanizmów ustawiona tak, aby po uderzeniu w obiekt moment obrotowy serwomechanizmów był wyłączany; trybu Low Power używa się podczas testów i uczenia robota punktów

High – wyłącza tryb Low Power co oznacza, że ograniczenia prędkości robota i sztywności serwomechanizmów są usuwane; High Power Mode jest normalnym trybem pracy robota

W przypadku pominięcia parametrów polecenie Power zwraca aktualnie ustawioną wartość (0 = Power Low, 1 = Power High)

Tryb Low Power włączany jest automatycznie w przypadku następujących zdarzeń:

- resetu
- wydania komendy Motor On
- wstrzymaniu wszystkich zadań
- włączeniu trybu nauczania (Teach mode)

Polecenie SPEED

Speed percent, [departSpeed], [approSpeed]

percent – liczba całkowita z zakresu 1 – 100 określająca prędkość ramienia wyrażoną jako procent prędkości maksymalnej

departSpeed – liczba całkowita z zakresu 1 – 100 określająca prędkość ramienia podczas odchodzenia od punktu wyrażoną jako procent prędkości maksymalnej

approSpeed – liczba całkowita z zakresu 1 – 100 określająca prędkość ramienia podczas dochodzenia do punktu wyrażoną jako procent prędkości maksymalnej

Dwa ostatnie parametry są opcjonalne.

Polecenie może być też wywołane bez żadnych parametrów i wtedy zwraca aktualnie ustawioną wartość prędkości.

Ustawiona za pomocą polecenia Speed prędkość odnosi się do wszystkich ruchów ramienia podczas przemieszczania się pomiędzy wyznaczonymi punktami, a więc wywołanych instrukcjami: Go, Jump i Pulse

Polecenie ACCEL

Accel accel, decel [departAccel, departDecel, approAccel, approDecel]

accel – liczba całkowita z zakresu 1 – 100 określająca przyspieszenie ramienia wyrażone jako procent przyspieszenia maksymalnego

decel – liczba całkowita z zakresu 1 – 100 określająca opóźnienie ramienia wyrażone jako procent opóźnienia maksymalnego

departAccel – liczba całkowita z zakresu 1 – 100 określająca przyspieszenie ramienia podczas odchodzenia od punktu wyrażone jako procent przyspieszenia maksymalnego

departDecel – liczba całkowita z zakresu 1 – 100 określająca opóźnienie ramienia podczas odchodzenia od punktu wyrażone jako procent opóźnienia maksymalnego

approAccel – liczba całkowita z zakresu 1 – 100 określająca przyspieszenie ramienia podczas dochodzenia do punktu wyrażone jako procent przyspieszenia maksymalnego

approDecel – liczba całkowita z zakresu 1 – 100 określająca opóźnienie ramienia podczas dochodzenia do punktu wyrażone jako procent opóźnienia maksymalnego

Polecenie ACCEL

Cztery ostatnie opcjonalne parametry (departAccel, departDecel, approAccel, approDecel) mają znaczenie jedynie podczas wykonywania instrukcji Jump w jej początkowej i końcowej fazie.

Polecenie może być też wywołane bez żadnych parametrów i wtedy zwraca aktualnie ustawione wartości przyspieszeń i opóźnień.

Polecenie WEIGHT

Weight payloadWeight, [distance]

payloadWeight – sumaryczna waga detalu i efektora końcowego w kg

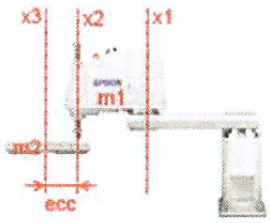
distance – odległość osi obrotu drugiego ramienia od środka ciężkości efektora końcowego i detalu mierzona w mm

Polecenie może być też wywołane bez żadnych parametrów i wtedy zwraca aktualnie ustawioną wartość wagi

Polecenie INERTIA

Inertia [loadInertia], [eccentricity]

loadInertia – sumaryczny moment bezwładności (w kgm2) detalu i drugiego ramienia względem drugiej osi robota (moment bezwładności elementów m1 i m2 względem osi x1 - rysunek)



eccentricity – odległość (w mm) pomiędzy środkiem ciężkości detalu, a osią efektora końcowego robota (ecc na rysunku)

Podanie momentu bezwładności i ekscentryczności pozwala na lepszą kompensację przyspieszeń, opóźnień i prędkości, co wpływa pozytywnie na dokładność pozycjonowania.

Oba parametry są opcjonalne, w przypadku ich niepodania polecenie Inertia zwróci aktualne wartości tego parametru.

Polecenie MOVE

Move *destination*

destination – punkt docelowy ruchu

Polecenie Move wykonuje ruch flanszy z punktu bieżącego do punktu docelowego po linii prostej (używając liniowej interpolacji do określenia trajektorii).

W trakcie działania polecenia wszystkie osie kończą swój ruch w tym samym czasie.

Trajektoria ruchu obliczana jest na bieżąco (nie jest znana przed rozpoczęciem działania polecenia) więc może się zdarzyć, że robot nagle zatrzyma się z powodu przekroczenia zakresu ruchu; aby uniknąć uszkodzenia serwomechanizmu należy zatem testować wykonanie polecenia Move przy niedużych prędkościach.

Polecenie Move może zostać także wywołane z przełącznikiem CP.

Polecenie GO

GO *point* [CP] [SearchExpr] [!...!]

Wykonuje ruch robota do zadanej pozycji, tak że wszystkie osie rozpoczynają i kończą pracę w tym samym czasie.

Odpowiednik ruchu JOINT.

Może być używana na kilka sposobów:

- GO P1
- GO P1 +X(30)
- GO P!1 :X(30)
- GO XY(30,0,0,0)

Polecenie JUMP

Polecenie Jump wykonuje skok do zdefiniowanego punktu w przestrzeni.

Jump destination [CarchNumber] [LimZ zLimit] [CP] [searchExpr] [!...!]

destination – docelowy punkt w przestrzeni, do którego ma być wykonane przemieszczenie; może być uzyty numer punktu lub jego etykieta

archNumber – numer łuku po którym ma być wykonane przemieszczenie; numer musi być poprzedzony literą C i być z zakresu C0 – C7; parametr ten jest opcjonalny

zLimit – maksymalna pozycja w osi Z na jakiej może przebywać flansza podczas wykonywania skoku; parametr ten jest opcjonalny

CP – ustawia ciągły ruch flanszy po wyznaczonej trajektorii; parametr ten jest opcjonalny

searchExpr – jedno z wyrażeń Sense, Till lub Find; parametr ten jest opcjonalny

!...! - służy do wstawiania innych poleceń, które mogą być wykonywane równolegle z instrukcją skoku do punktu (np. sterowanie portami I/O); parametr ten jest opcjonalny

Polecenie JUMP

Przykład użycia:

Jump P0 C0 LimZ -30

Wartość parametru zLimit w określonym skoku nie może być mniejsza niż współrzędne Z punktów, pomiędzy którymi odbywa się skok (żaden z punktów trajektorii ruchu nie może znajdować się powyżej wartości zLimit); w takim przypadku pojawi się stosowne ostrzeżenie w oknie Status.

Również znalezienie się flanszy w położeniu powyżej zLimit spowodowane innymi czynnikami (np. ręczne przestawienie położenia) spowoduje wyświetlenie ostrzeżenia i zablokuje możliwość wykonania ruchu.

Polecenie BGo

BGo destination

destination – punkt docelowy ruchu

Polecenie BGo wykonuje ruch względny z punktu bieżącego do punktu docelowego w lokalnym układzie współrzędnych; jeśli lokalny układ współrzędnych nie zostanie określony to ruch odbywa się w bazowym układzie współrzędnych.

Przykład:

BGo XY(0,50,0,0) /1 'ruch względny w układzie lokalnym nr 1

BGo XY(0,50,0,0) 'ruch względny w układzie bazowym

Polecenie może zostać także wywołane z przełącznikiem CP, który oznacza ruch ciągły (brak wyraźnych zatrzymań w punktach docelowych w trakcie wykonywania serii ruchów): BGo destination CP

Polecenie ARCH

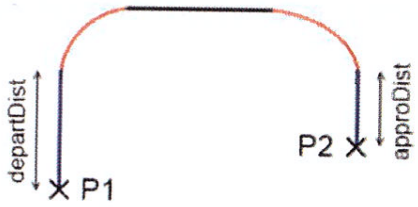
Arch archNumber, departDist, approDist

archNumber – liczba całkowita z zakresu 0 – 6 określająca numer definiowanego łuku

departDist – liczba określająca w dystans w milimetrach, jaki ma pokonać flansza w osi Z przy odchodzeniu od punktu zanim rozpocznie ruch w poziomie

approDist – liczba określająca dystans w milimetrach, jaki ma pokonać flansza w osi Z przy dochodzeniu od punktu

Polecenie Arch służy do definiowania łuków trajektorii ruchu; dzięki łukom skraca się czas potrzebny na przejście z jednego punktu do drugiego.



Polecenie ARCH

Odwołanie się do zdefiniowanego wcześniej łuku odbywa się przez podanie jego numeru poprzedzonego literą C (np. C0, C2).

Przykład użycia:

```
Arch 0, 15, 15          'definicja luku C0
Arch 1, 30, 50          'definicja luku C1
Jump P1 C0              'skocz do P1 po łuku C0
Wait 2
Jump P2 C1              'skocz do P2 po łuku C1
```

Gdy polecenia związane z ruchem robota wywoływane są bez podania numeru łuku, wykonywany jest ruch prostokątny.

Parametry poszczególnych łuków przechowywane są w tablicy w pamięci kontrolera i raz zdefiniowane pozostają niezmienne aż do kolejnego odwołania się do ich definicji (nie są kasowane przez reset i inne zdarzenia).

Polecenie ARCH

Każdy z 7 dostępnych łuków ma predefiniowane wartości, które można zmienić; predefiniowane wartości są następujące:

Arch Number	Depart Distance	Approach Distance
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

Aby sprawdzić aktualne ustawienia łuków należy w oknie Command (CTRL + M lub Tools → Command Window) wpisać polecenie Arch bez podawania żadnych dodatkowych parametrów.

```
arch
    arch0 =    15.000      15.000
    arch1 =    40.000      40.000
    arch2 =    50.000      50.000
    arch3 =    60.000      60.000
    arch4 =    70.000      70.000
    arch5 =    80.000      80.000
    arch6 =    90.000      90.000
>
```

Polecenie PALLET

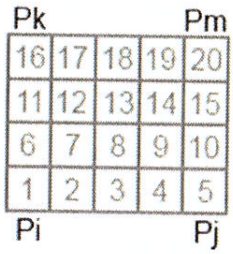
Pallet – polecenie definiuje rozmieszczenie elementów na palecie.

Pallet [Outside,] [palletNumber, Pi, Pj, Pk [,Pm], columns, rows]

palletNumber – numer palety (liczba całkowita z zakresu 0 do 15)

Pi, Pj, Pk – punkty reprezentujące trzy narożniki palety

Pm – dodatkowy (opcjonalny) punkt reprezentujący czwarty punkt palety



columns – liczba całkowita z zakresu 1-32767 reprezentująca liczbę kolumn palety (bok Pi – Pj)

rows – liczba całkowita z zakresu 1-32767 reprezentująca liczbę wierszy palety (bok Pi – Pk)

Polecenie PALLET

Odwołanie do elementów tak zdefiniowanej palety wygląda następująco:

Pallet(palletNumber, position)

position – pozycja w palecie (liczba całkowita z zakresu 0 do columns · rows)

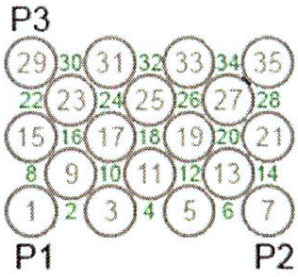
Przykład:

```
Pallet 0, P0, P1, P2, 5, 4        'paleta o numerze 0, 5 wierszach i 4 kolumnach,
                                  'wierzchołkach w punktach P0, P1, P2

Jump Pallet(0, 3)                 'odwołanie do 3 elementu palety o nr 0
```

Polecenie PALLET

W przypadku paletyzacji elementów o przekroju innym niż prostokątny (np. okrągły lub wielościenny) często zachodzi potrzeba utworzenia palety o następującej postaci:



Przy konstruowaniu powyższej palety należy uwzględniać zarówno miejsca, gdzie mają być umieszczane obiekty jak również przestrzenie pomiędzy nimi; natomiast obiekty na palecie umieszczać należy w co drugim punkcie.

Polecenie WAIT

WAIT *time*

Polecenie to umożliwia ustawianie czasu oczekiwania przed wykonaniem kolejnego polecenia.

Instrukcje do sterownia sygnałami

Bitami wejścia/wyjścia/pamięci starować można także za pośrednictwem poleceń języka SPEL+

In	odczytuje jeden bajt bitów wejściowych
InBCD	odczytuje jeden bajt bitów wejściowych w formacie BCD
InW	odczytuje jedno słowo bitów wejściowych
Oport	odczytuje jeden bit wyjściowy
Sw	odczytuje jeden bit wejściowy
Off	wyłącza jeden bit wyjściowy
On	włącza jeden bit wyjściowy
OpBCD	ustawia jeden bajt bitów wyjściowych w formacie BCD
Out	ustawia/odczytuje jeden bajt bitów wyjściowych
OutW	ustawia/odczytuje jedno słowo bitów wyjściowych
MemOff	wyłącza jeden bit pamięci
MemOn	włącza jeden bit pamięci
MemOut	ustawia/odczytuje jeden bajt bitów pamięci
MemSw	odczytuje jeden bit pamięci

Instrukcje do sterownia sygnałami

Funkcje:

SW(bitNumber) – zwraca wartość logiczną wejścia

Oport(bitNumber) - zwraca wartość logiczną wyjścia

In(bytePortNumber) – zwraca wartość dziesiętną na postawie 8 kolejnych wejść

Wyrażenia:

On {bitNumber|Label}, [time] - ustawia bit wyjściowy w stan wysoki [na czas]

Off {bitNumber|Label}, [time] - ustawia bit wyjściowy w stan niski [na czas]

Out bytePortNumber, outData – ustawia wartość dziesiętną na 8 kolejnych bitach wyjściowych

Analogicznie Memory bits:

Mem On, MemOff, MemOut, MemSW