ROBOTYKA 2
Programowanie robota FANUC w środowisku symulacyjnym Roboguide

Autor: Emilia Szymańska 248975
śrTP 8:00

27.04.2021r.

1 Cel ćwiczeń

Celem ćwiczeń jest zapoznanie się podstawami programowania robotów manipulacyjnych firmy FANUC przy wykorzystaniu środowiska symulacyjnego Roboguide.

2 CELL1

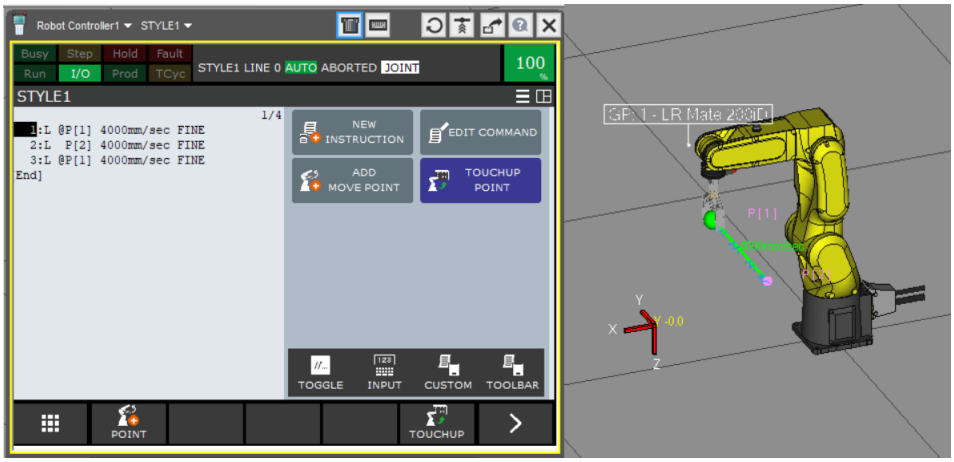
Filmiki pokazujące działanie programów znajdują się pod linkiem:
`https://www.youtube.com/playlist?list=PLshKwg0csHROvYNkQHfQUlnx7kwV8Tv3I`.
W przypadku każdego z zadań punkty definiowałam na jeden z dwóch sposobów - klikając na zieloną sferę w środku chwytaka i ręcznie manipulując osiami lub z panelu wybierając przycisk *POSN* i wpisując wartości współrzędnych. Ważnym było, by pamiętać o załączeniu odpowiednika przycisku DEADMAN SWITCH oraz wciśnięciu SHIFT przy funkcji TOUCHUP lub usuwaniu komend. Poniższe zadania rozwiązane zostały wykorzystując jedynie Teach Pendanta.

2.1 Zadanie 1

Głównym celem zadania nr 1 było wskazanie różnicy pomiędzy komendami ruchu w przestrzeni konfiguracyjnej (MOVE J) a ruchu w przestrzeni roboczej (MOVE L). W przypadku ruchu MOVE J robot nie optymalizuje przemieszczenia się biorąc pod uwagę przestrzeń Euklidesową, a optymalizując ruch przegubów. W przypadku MOVE L jest odwrotnie - ruch jest po linii prostej tworzącą najkrótszą drogę z punktu A do punktu B. Prędkość wykonania ruchu można było ustawić w jednostkach mm/sec.

2.1.1 Ruch w przestrzeni konfiguracyjnej

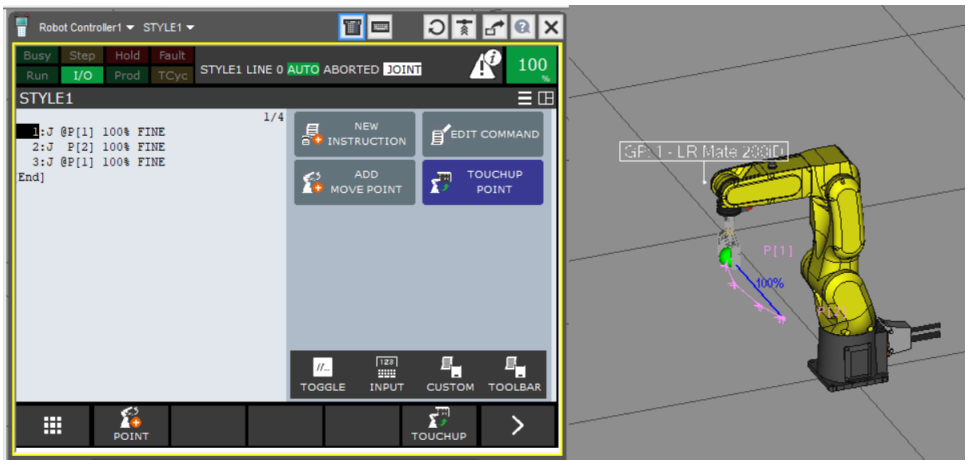
Zadaniem było napisanie programu przemieszczającego manipulator między dwoma punktami w przestrzeni konfiguracyjnej. W tym celu wykorzystano opcję MOVE J (rys. 1), wybierając punkt początkowy i końcowy. Ruch odbył się po łuku. Prędkość wykonania ruchu można było ustawić w wartości procentowej.



Rysunek 1: Ruch w przestrzeni konfiguracyjnej.

2.1.2 Ruch w przestrzeni zadaniowej

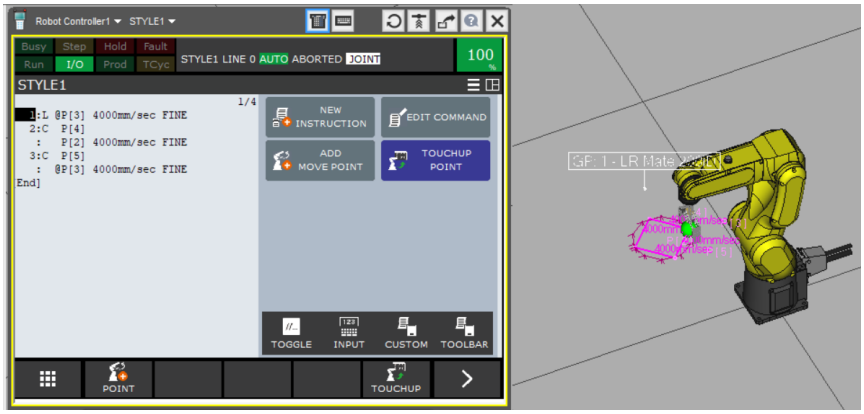
Zadaniem było napisanie programu przemieszczającego manipulator między dwoma punktami w przestrzeni zadaniowej/roboczej. W tym celu wykorzystano opcję MOVE L (rys. 2), wybierając punkt początkowy i końcowy. Ruch odbył się po linii prostej.



Rysunek 2: Ruch w przestrzeni zadaniowej.

2.2 Zadanie 2

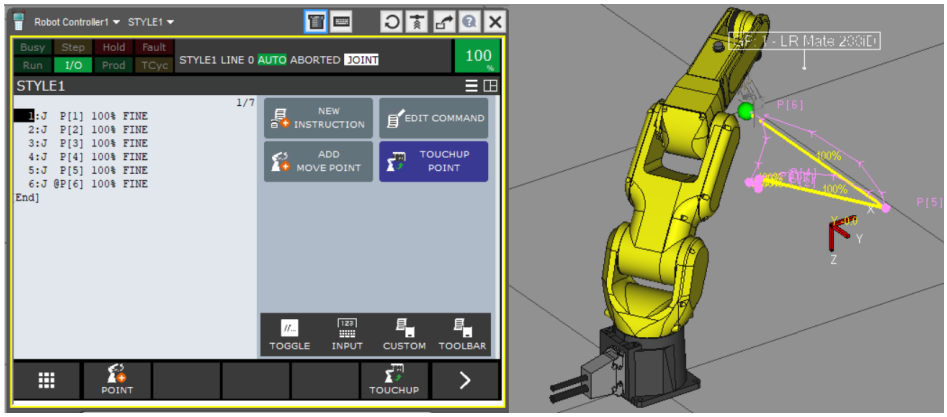
Następne zadanie dotyczyło stworzenia programu kreślącego okrąg końcówką roboczą manipulatora. W tym celu wykorzystano ruch MOVE C. Definicja ruchu po okręgu składa się z dwóch części - zdefiniowania punktów A-B tworzących średnicę okręgu oraz punktów pośrednich, przez które robot powinien przejść końcówką roboczą. Jako pierwszy krok zdefiniowałam dodatkowo liniowe przejście do punktu startowego. Rezultat wraz z kodem zaprezentowany jest na rys. 3.



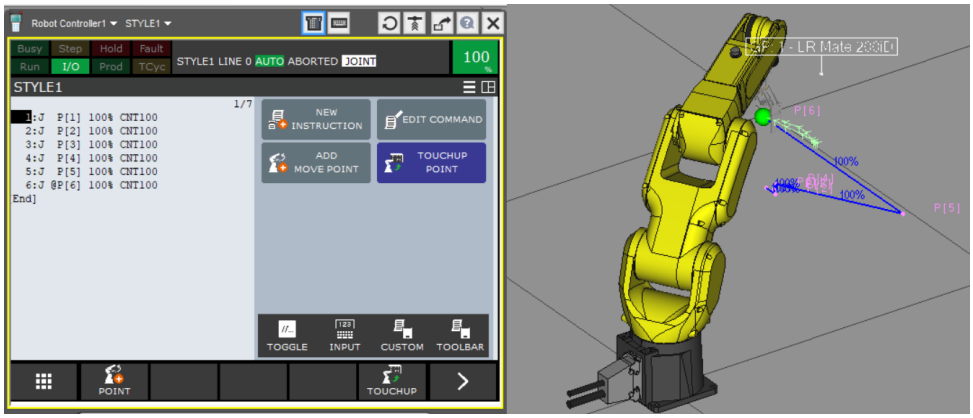
Rysunek 3: Ruch końcówką roboczą manipulatora po okręgu.

2.3 Zadanie 3

Zadanie składało się z dwóch części - jednej używającej opcji FINE, drugiej wykorzystującej ustawienie CNT100. Dokładność FINE definiuje, że robot powinien przejść końcówką roboczą manipulatora przez wszystkie zdefiniowane punkty. CNT pozwala na odstępstwo od przejścia przez punkty, a liczba przy nim stojąca definiuje, jak duże może być odstępstwo od zaplanowanej przez nas trajektorii. W obu przypadkach wybrane zostało sześć punktów (cztery z nich były blisko siebie, dwa pozostałe względnie daleko). Efekt działania dla FINE widać na rys. 4, wynik dla CNT100 zaprezentowany jest na rys. 5.



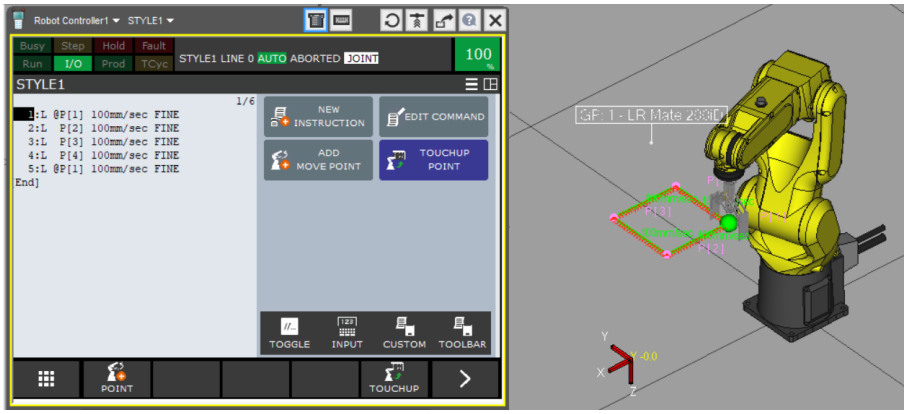
Rysunek 4: Ruch końcówką manipulatora z dokładnością FINE.



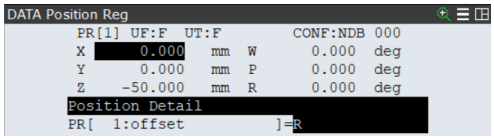
Rysunek 5: Ruch końcówką manipulatora z dokładnością CNT100.

2.4 Zadanie 4

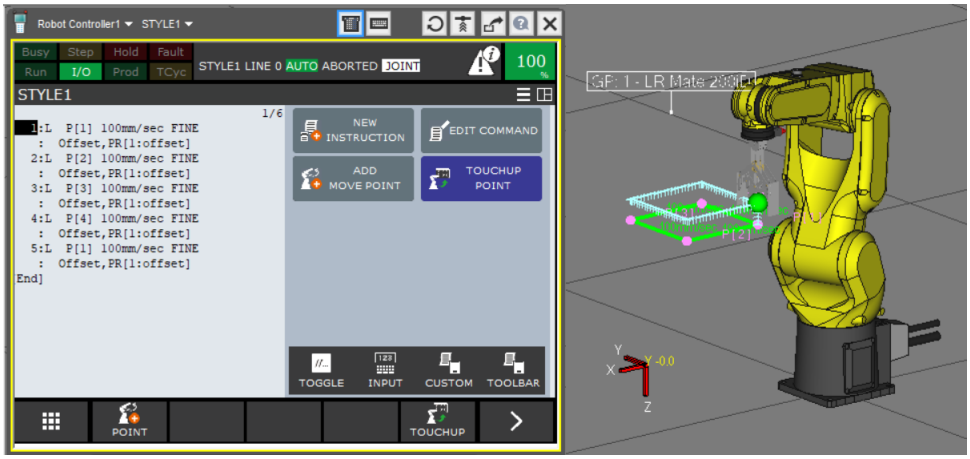
Zadanie nr 4 dotyczy napisania programu kreślącego kwadrat końcówką chwytaka. Na początku zdefiniowałam ruch L pomiędzy czterema określonymi punktami (zdefiniowałam je ręcznie wpisując żądane współrzędne po wybraniu opcji *POSN*). Po uruchomieniu tego programu (rys. 6) i poprawnym wykonaniu, należało go zmodyfikować wykorzystując Offset. W tym celu zdefiniowałam swoją wartość w rejestrze Position Register jak na rys. 7. Po wykorzystaniu funkcji Offset z tym parametrem rysowany jest kwadrat przesunięty wzdłuż osi Z o -50mm (rys. 8).



Rysunek 6: Kreślenie kwadratu.



Rysunek 7: Wartość rejestru PR.



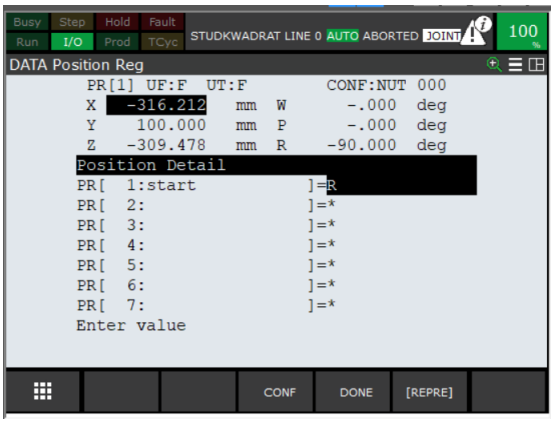
Rysunek 8: Kreślenie kwadratu z offsetem.

2.5 Zadanie 5

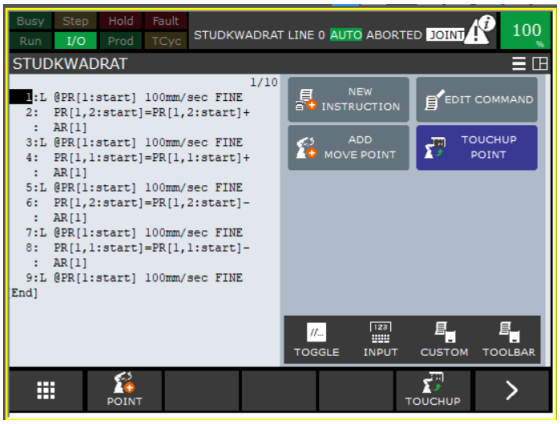
Ostatnim zadaniem z komórki CELL jest modyfikacja zadania nr 4 w taki sposób, by utworzony podprogram StudKwadrat kreślący kwadrat końcówką manipulatora w płaszczyźnie XY, wywoływany był z argumentem wielkości kwadratu.

Na początku ustawiłam końcówkę manipulatora w wybranym przez siebie miejscu i wywołując DATA-¿TYPE-¿POSITION REGISTER-¿RECORD zapisałam w rejestrze Start POSITION REGISTER wybraną pozycję (9). Wcześniej próbowałam ręcznie przepisać do rejestru wartości widoczne w *POSN*, jednak wywoływanie programu kończyło się w takim przypadku błędem dotyczącym wyjścia poza zakres ruchu.

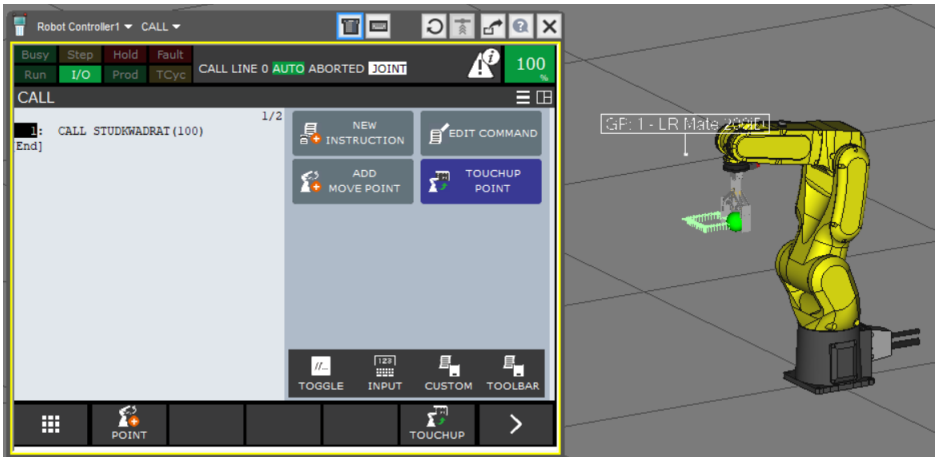
Program StudKwadrat składa się z instrukcji naprzemiennie ruchu L do punktu Start i modyfikacji współrzędnych tego punktu (rys. 10) o wartość rejestru AR (w którym zapisywane są parametry wywołania programu). Program ten jest wywoływany w programie CALL (rys. 11).



Rysunek 9: Wartość rejestru Start dla pozycji początkowej.



Rysunek 10: Program StudKwadrat.



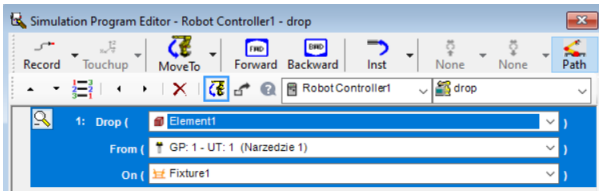
Rysunek 11: Wywołanie programu StudKwadrat w nadrzędnym programie z parametrem.

3 CELL2

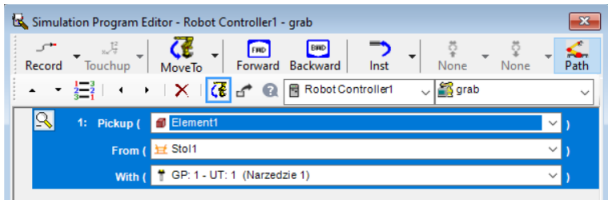
Filmik pokazujący działanie programu znajduje się pod linkiem:
`https://www.youtube.com/playlist?list=PLshKwg0csHRO9eVUV4n5px61dH1esjmGZ`.

W komórce CELL2 znalazły się dwa stoły, element oraz robot. Zadaniem było przenoszenie elementu ze stołu pomarańczowego na niebieski przez pięć cykli. Element został tak skonfigurowany, aby w miejscu pobranego z pomarańczowego stołu detalu po 8 sekundach pojawiał się nowy, a odłożony na niebieski stół element znikał po 5 sekundach. Dostosowałam tak prędkość ruchu robota, by nie musiał czekać na pojawienie się elementu.

Na początku w Simulation Program Editor zdefiniowałam dwa programy - drop (odłożenie elementu - rys. 12) oraz grab (chwycenie elementu - rys. 13).

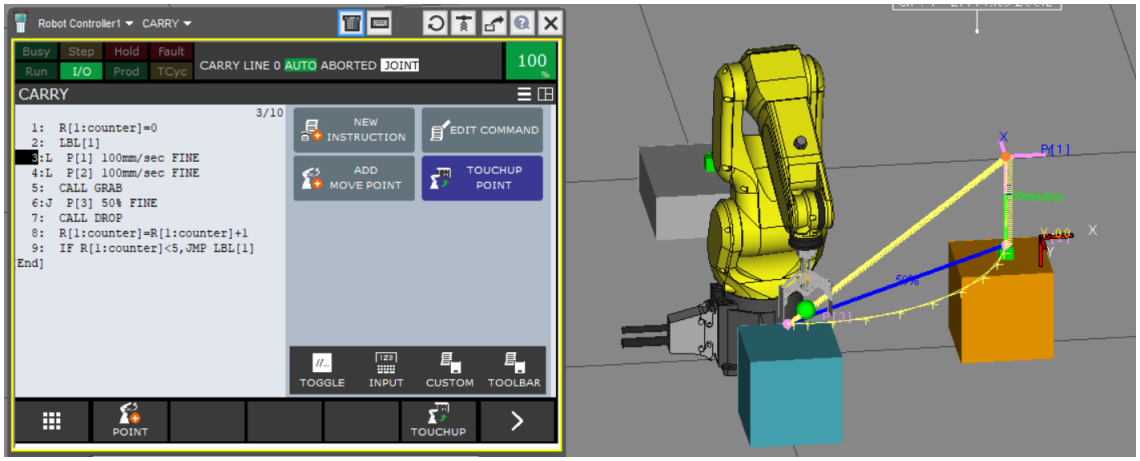


Rysunek 12: Program *drop* w Simulation Program Editor.



Rysunek 13: Program *grab* w Simulation Program Editor.

Następnie w rejestrze Register utworzyłam zmienną całkowitoliczbową zliczającą ilość wykonanych cykli.Na początku programu zeruję zmienną zliczającą cykle i dodaję Label. Przechodzę po kolejnych punktach na trajektorii, pobierając i odkładając element, zwiększam licznik i jeśli jego wartość jest mniejsza od 5, wówczas przechodzę do kroku ze zdefiniowanym Labelem. W przeciwnym wypadku zadanie jest wykonane i można zakończyć program. Kod programu znajduje się na rys. 14



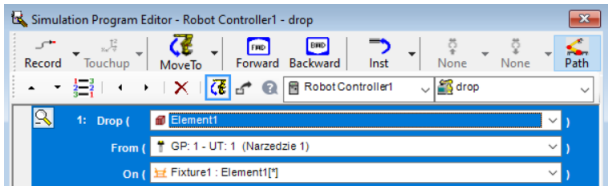
Rysunek 14: Program wykorzystujący pętlę.

4 CELL3

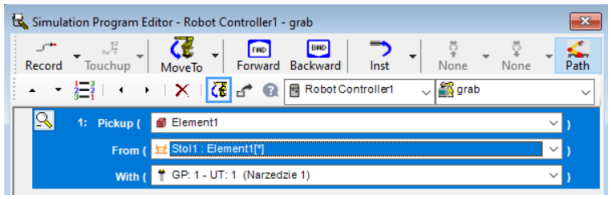
Filmik pokazujący częściowe działanie programu (oraz fragment z błędem) znajduje się pod linkiem: `https://www.youtube.com/playlist?list=PLshKwgOcsHRNLroVwYx2sKuG3upjbWTOK`.

Ostatni program ma za zadanie sterować robotem tak, by przeniesiona została (element po elemencie) cała paleta detali z pomarańczowego stołu na niebieski.

Na początku w Simulation Program Editor zdefiniowałam dwa programy - drop (odłożenie elementu - rys. 15) oraz grab (chwycenie elementu - rys. 16). W przeciwieństwie do analogicznych programów w CELL2, tutaj należy wybrać element ze znakiem gwiazdki - oznacza to, że interesuje nas najbliższy element względem chwytaka.

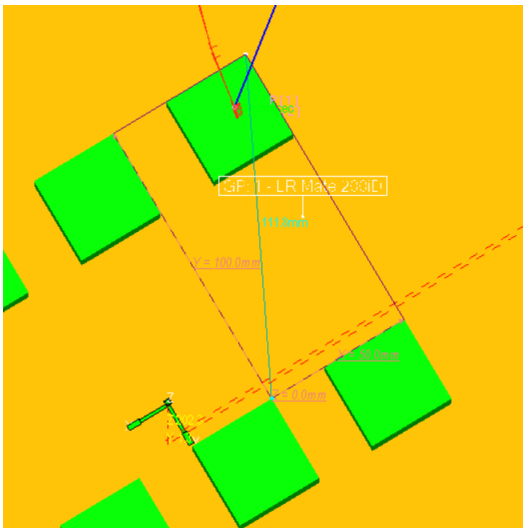


Rysunek 15: Definicja programu *drop*.

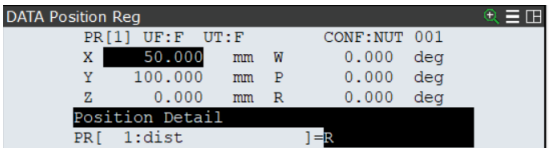


Rysunek 16: Definicja programu *grab*.

W następnym kroku należało zmierzyć odległości między detalami. Przy pomocy odpowiedniego narzędzia otrzymałam, że wzdłuż osi X przesunięcie wynosi 50mm, a wzdłuż osi Y - 100mm *rys. 17). Te wartości wpisałam do rejestru dist Position Register jak w 18.



Rysunek 17: Pomiar odległości między detalami.



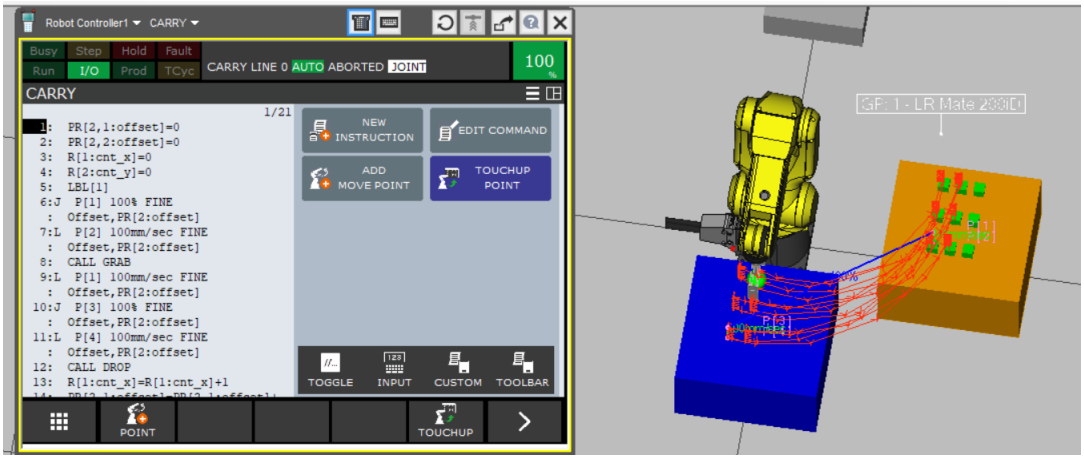
Rysunek 18: Rejestr przetrzymujący wartości przesunięcia między detalami.

Dodatkowymi zmiennymi zapisanymi w rejestrach typu Register są cnt_x oraz cnt_y - służą one do sprawdzenia warunków w zagnieżdżonych pętlach. Na początku programu zerowane są wartości obecnego offsetu oraz wartości zmiennych „pętlowych". Dwa Labele pozwalają na wykorzystanie odpowiednika dwóch zagnieżdżonych funkcji FOR. Działanie programu sprowadza się do przejścia nad pierwszy element, pobrania go, odłożenia na drugi stół, a następnie powrotu nad stół poprzedni, przy czym dodawane są wartości offsetu (w późniejszych krokach offset jest odpowiednio modyfikowany). Treść prgramu znajduje się na rys. 19.

```
 1:  PR[2,1:offset]=0
 2:  PR[2,2:offset]=0
 3:  R[1:cnt_x]=0
 4:  R[2:cnt_y]=0
 5:  LBL[1]
 6:J  P[1] 100% FINE
  :  Offset,PR[2:offset]
 7:L  P[2] 100mm/sec FINE
  :  Offset,PR[2:offset]
 8:  CALL GRAB
 9:L  P[1] 100mm/sec FINE
  :  Offset,PR[2:offset]
10:J  P[3] 100% FINE
  :  Offset,PR[2:offset]
11:L  P[4] 100mm/sec FINE
  :  Offset,PR[2:offset]
12:  CALL DROP
13:  R[1:cnt_x]=R[1:cnt_x]+1
14:  PR[2,1:offset]=PR[2,1:offset]+
  :  PR[1,1:dist]
15:  IF R[1:cnt_x]<2,JMP LBL[1]
16:  PR[2,1:offset]=0
17:  R[1:cnt_x]=0
18:  R[2:cnt_y]=R[2:cnt_y]+1
19:  PR[2,2:offset]=PR[2,2:offset]-
  :  PR[1,2:dist]
20:  IF R[2:cnt_y]<3,JMP LBL[1]
```

Rysunek 19: Lista kroków programu.

Na rys. 20 zostało przedstawione działanie programu. Niestety w trakcie jego wykonywania napotkałam błąd (limitation error), który nie pozwala mi się dostać do trzech elementów. Robot chcąc pobrać jeden z nich, kończy działanie programu ze wspomnianym błędem.



Rysunek 20: Efekt działania programu.

5 Wnioski

- Ze wszystkich programów nie udało się wykonać części ostatniego z CELL3 - pobieranych jest 6 elementów i przy próbie pobrania 9-ciu program kończy się z błędem „limitation error”. Pomimo prób obejścia tego błędu, nie udało mi się znaleźć dobrego rozwiązania.
- Wybór pomiędzy ruchami J, L oraz C (ich opis w podpunkcie *CELL1*) pozwala na decyzję w kwestii rodzaju optymalizacji danego ruchu, a tym samym jego trajektorii.
- Widoczna była także zmiana zachowania robota przy wykonywaniu instrukcji z dokładnością FINE oraz CNT100. W zależności od ograniczeń (lub ich braku) możemy dopasować cechy programu do zastosowania.
- Możliwym jest definiowanie pętli przy programowaniu robotów FANUC - w tym celu używana jest instrukcja Label.
- Instrukcja Offset jest przydatna, jeśli chcemy nieznacznie modyfikować powtarzalną czynność/ruch przy ograniczeniu liczby linii kodu.
- Rejestry służą do przechowywania wartości zmiennych. Jak w przypadku klasycznego programowania, mogą mieć one różne typy (np. całkowitoliczbowe, pozycyjne).
- Przydatnym narzędziem jest tzw. MeasureTool - pozwala ono w prosty sposób na dokładny pomiar odległości między elementami w trzech wymiarach jednocześnie.
- Mimo pojawiającego się miejscami braku intuicyjnych rozwiązań przy programowaniu (np. zatwierdzaniem wpisanych wartości klawiszem EXIT), Roboguide pozwala na stosunkowo proste programowanie robotów przemysłowych.