Fundusze Europejskie Inteligentny Rozwój | Rzeczpospolita Polska | Narodowe Centrum Badań i Rozwoju | Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Politechnika Świętokrzyska

WYDZIAŁ MECHATRONIKI I BUDOWY MASZYN

PROJEKT BADAWCZY
POIR. 04.01.02-00-0045/18.

"Opracowanie i demonstracja zrobotyzowanego systemu murarsko-tynkarskiego (ZSMT) do zastosowania w przemyśle budowlanym" akronim ZSMT

Zakres obowiązków

1) Analiza wymagań projektowych dla oprogramowania zrobotyzowanego systemu murarskiego (ZSM).
2) Wykonanie i implementacja oprogramowania dla zrobotyzowanego systemu murarskiego (ZSM).

Wykonawca projektu
Artur Arciszewski

Kielce, 2019-2021 r.

I. Analiza wymagań projektowych dla oprogramowania zrobotyzowanego systemu murarskiego (ZSM).

Główny problem badawczy dotyczy algorytmów sterowania systemem jako całością oraz dostosowanie robota do uniwersalności (różny asortyment elementów konstrukcyjnych cegieł bloczków. Rozwiązanie problemu badawczego realizowane będzie w następujący sposób.

Prace zostaną rozpoczęte od zagadnień związanych z manipulatorem i mobilną platformą nośną manipulatora.

Konstrukcja manipulatora uzależniona jest od czynników technologicznych, takich jak: rodzaje cegieł lub bloczków i zaprawy w zależności od stosowanej technologii.

Prawidłowa praca robota wymaga: obsługi różnych rodzajów i wielkości cegieł i bloków, wykrywanie i kompensacja tolerancji materiałowych, kalibracji położenia cegły lub bloku w stosunku do środkowego narzędzia (TCP), zautomatyzowania dozowania materiału wiążącego,

Aby właściwie opisać sposób sterowania manipulatorem robota należy najpierw scharakteryzować jego strukturę kinematyczną. W założeniu, manipulator w który wyposażony będzie robot będzie miał sześć stopni swobody. Przewiduje się, że ruch narzędzia roboczego zamocowanego w końcówce roboczej będzie wynikał z ruchu całego manipulatora tj. narzędzie nie będzie zmieniało ilości stopni swobody manipulatora. Dodatkowo łańcuch kinematyczny jest otwarty. Dla części zdefiniowanych dla robota zadań, w zależności od zastosowanego narzędzia (uniwersalność robota pod względem wykonywanych prac budowlanych), manipulator nie będzie kinematycznie ani funkcjonalnie redundantny, ponieważ liczba stopni swobody to 6 więc nie może być on redundantny w sposób istotny. Aby zrealizować sterowanie manipulatorem rozwiązywane będzie zadanie kinematyki odwrotnej (ZO), przy czym kolejne iteracje obliczeń będą wykonywane z różną częstotliwością, ze względu na wymaganą płynność ruchu końcówki roboczej. Rozwiązanie zadania odwrotnego kinematyki będzie w przypadku robota budowlanego podporządkowane zadaniu planowania i wykonania trajektorii, gdyż ma ona istotny wpływ na jakość wykonywanych przez robota prac.. Równolegle do działania zadania odwrotnego kinematyki odpowiedni moduł oprogramowania będzie czuwał nad bezpieczeństwem użytkowym. Przez to pojęcie rozumie się takie mechanizmy sterowania, które zabezpieczą przed samouszkodzeniem, uszkodzeniem środowiska (wykonanych już np. elementów konstrukcji budowli) w którym robot się porusza oraz zabezpieczeniem operatora i osób postronnych. Samouszkodzenie manipulatora będzie

wyeliminowane przez dobór stref bezpieczeństwa w przestrzeni złączy dla aktualnej konfiguracji manipulatora.

Do rozwiązania zagadnienia wykorzystywane są również Metody bazujące na odczytach z urządzeń laserowych: dalmierzy czy skanerów bazujących na metodzie pomiaru czasu przelotu ToF (ang. Time of Flight). Dodatkowo odczyty laserowe nie są zakłócane przez zewnętrzne źródła światła co czyni je doskonałym czujnikiem do zastosowań zewnętrznych.

Wyniki niniejszego Projektu będą wykorzystywane przez szeroko pojętą branżę budowlaną. Należy zauważyć, że wiedza zdobyta przy realizacji Projektu pozwoli na rozwinięcie zakresu Projektu i umożliwi wykorzystanie wiedzy zdobytej podczas jego realizacji również w ramach projektów dotyczących Obronności i Bezpieczeństwa Państwa np. poprzez stworzenie uniwersalnej platformy działań inżynieryjnych.

II. Wykonanie i implementacja oprogramowania dla zrobotyzowanego systemu murarskiego (ZSM).

Zadanie, które należało wykonać to stworzenie algorytmu a następnie zaimplementowania go w programie sterującym pracą robota przemysłowego do budowania ściany z cegieł.

Do napisania programu zostało wykorzystane środowisko programistyczne RobotStudio firmy ABB Ltd. pozwalające tworzyć programy w języku RAPID oraz testować offline we wbudowanym symulatorze. Po wykonanych testach na symulatorze program przenoszony jest do sterownika rzeczywistego robota. Trajektorie i targety użyte w symulatorze są w pełni kompatybilne z rzeczywistymi. Podczas dalszej pracy online wszystkie zmiany i edycja programu wykonywane są nadal w RobotStudio.

Główne czynności robota wykonywane cyklicznie po starcie z punktu *Home* to:

1. Dojazd do podajnika cegieł.
2. Pobranie cegły.
3. Dojazd do punktu aplikacji zaprawy.
4. Aplikacja zaprawy.
5. Dojazd do punktu *cross* rozprowadzania cegieł na ścianę.
6. Dojazd do ściany.
7. Wstawienie cegły.
8. Powrót do punktu *cross*.

Główny szkielet programu w procedurze *main()* oparty jest na dwóch zagnieżdżonych pętlach *while()*, które przebiegają przez kolejne rzędy ściany układając w każdej kolejne cegły:

WHILE warunek dla wysokości DO
Zwiększ numer rzędu o 1
WHILE warunek dla długości DO

Wstaw cegłę
ENDWHILE
Zmień parzystość rzędu
ENDWHILE

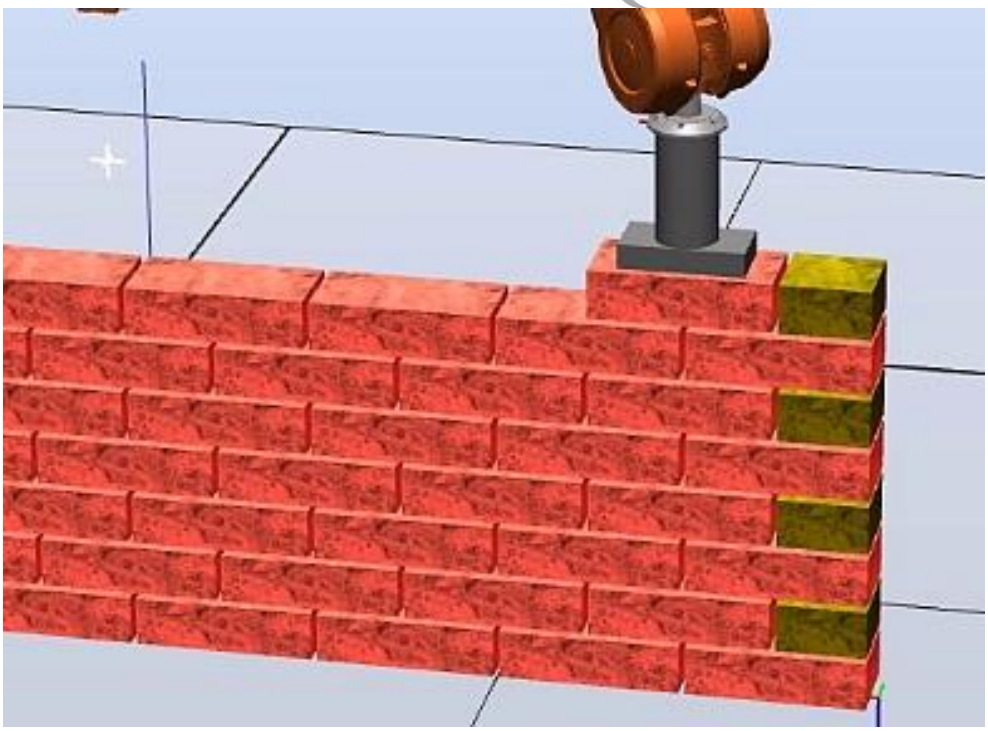
W powyższym pseudokodzie *warunek dla wysokości* sprawdza, czy dodanie nowego rzędu cegieł nie przekroczy założonej wysokości ściany:
WHILE wys_actual+zaprawa_level+brick_H<=wys_wall DO...
gdzie:
wys_actual - aktualna wysokość ściany,
zaprawa_level - grubość zaprawy w pionie,
brick_H - wysokość cegły,
wys_wall - założona wysokość budowanej ściany.
Natomiast *warunek dla długości* sprawdza, czy dodanie nowej cegły nie przekroczy założonej długości ściany:
WHILE dlug_actual+brick_L+zaprawa_vertical<=dlug_wall DO ...
gdzie:
dlug_actual - aktualna długość ściany,
zaprawa_vertical- grubość zaprawy w poziomie,
brick_L - długość cegły,
dlug_wall - założona długość budowanej ściany.
Instrukcja *Zmień parzystość rzędu* w powyższym pseudokodzie to :

row_is_par:=NOT row_is_par;

gdzie *row_is_par* jest zmienną typu *bool* odpowiadającą za parzystość budowanego rzędu zaczynając od rzędu najniższego o wartości *FALSE*. W zależności od wartości tej zmiennej rząd zaczynamy budować albo od całej cegły albo połówki cegły plus cała cegła. Jeśli wartość jest *TRUE* to zaczynamy od połówki cegły plus całej cegły. W przeciwnym razie od całej cegły. Odpowiada za to poniższy fragment kodu:

IF row_is_par=TRUE THEN
SetFirstHalfAndFullBrick;
...
ELSE
SetFirstFullBrick;
...
ENDIF

Za odmienne sposoby rozpoczynania rzędów cegieł odpowiadają procedury *SetFirstFullBrick()* oraz *SetFirstHalfAndFullBrick()*.



Rys.1 Dwa sposoby rozpoczynania rzędu

W rzędach parzystych zaczynamy od połówki i całej cegły, gdyż dopiero wtedy możemy uzyskać powtarzalność układania pozostałych cegieł. W rzędach nieparzystych powtarzalność uzyskujemy już po wstawieniu pierwszej całej cegły. Wynika to z geometrii. Każdą z pozostałych cegieł układamy w ścianie za pomocą procedury *SetRestBricks()*.

Wykonuje to fragment kodu:

```
WHILE dlug_actual+brick_L+zaprawa_vertical<=dlug_wall DO
    SetRestBricks;
    dlug_actual:=dlug_actual+brick_L+zaprawa_vertical;
ENDWHILE
```

W programie umożliwiony jest wybór jednej z 5 rodzajów cegieł, z których układana jest ściana. W procedurze *InitVariables()* rodzaj cegły wybieramy za pomocą instrukcji *TEST*:

```
brick_nr:=1;
TEST brick_nr
        CASE 1:
                brick_L:=373;
                brick_W:=250;
                brick_H:=236;
        CASE 2:
...
```

ENDTEST

...

gdzie::

brick_nr - numer wybrnej cegły,

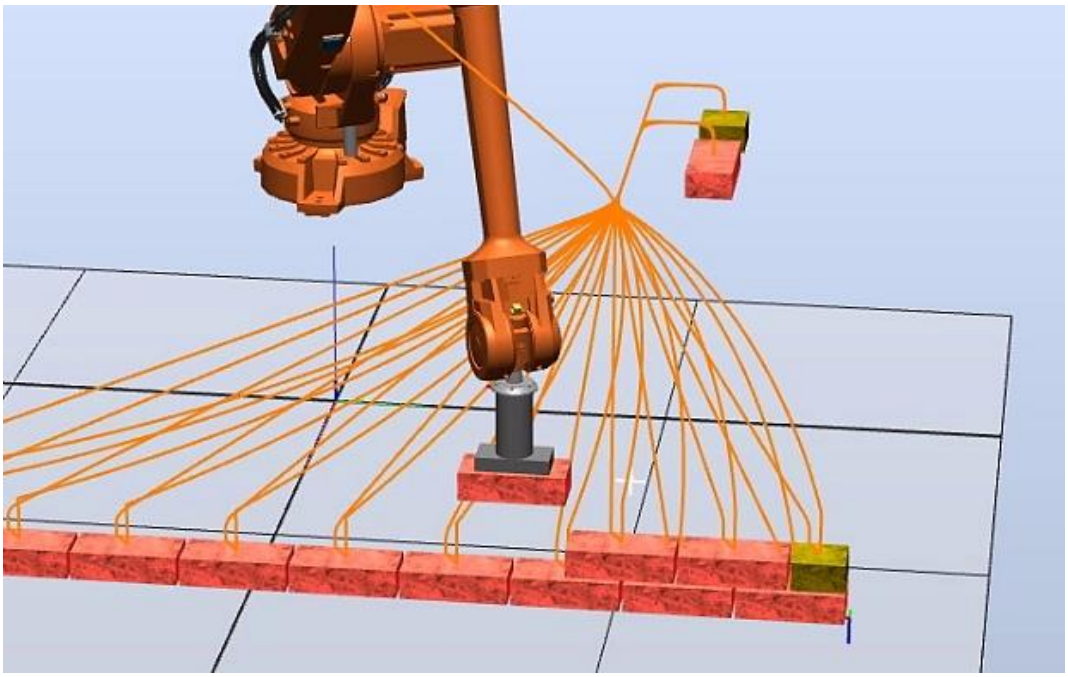
brick_L, *brick_W*, *brick_H* - długość, szerokość i wysokość wybranej cegły.

Ponadto w tej procedurze ustalamy wartości grubości zaprawy w poziomie i pionie:

zaprawa_vertical:=5;

zaprawa_level:=3;

Podczas układania cegieł chwytak robota porusza się po dwóch rodzajach trajektorii: stałej i zmiennej. Stała trajektoria jest pomiędzy targetem *cross* a magazynem całych cegieł oraz połówek cegieł. Natomiast trajektoria zmienna pomiędzy targetem *cross* a zmiennym targetem aktualnie wstawianej cegły.



Rys.2 Trajektorie stałe i zmienne

Cały cykl pobierania i wstawiania cegieł w ścianę podzielony jest na wiele procedur. Taki podział pozwala łatwiej testować i wyszukiwać błędy w programie. Ponadto wydzielony zestaw instrukcji w procedurze pozwala wywołać ją z poziomu FlexPendanta. Oto te procedury:

1. *CrossToBrick()* - przejście z punktu cross do magazynu całych cegieł i pobranie cegły.
2. *CrossToHalfBrick()* - przejście z punktu cross do magazynu połówek całych cegieł i pobranie połówki cegły.
3. *Move_cement()* - przejście do punktu z zaprawą i nałożenie zaprawy na cegłę.

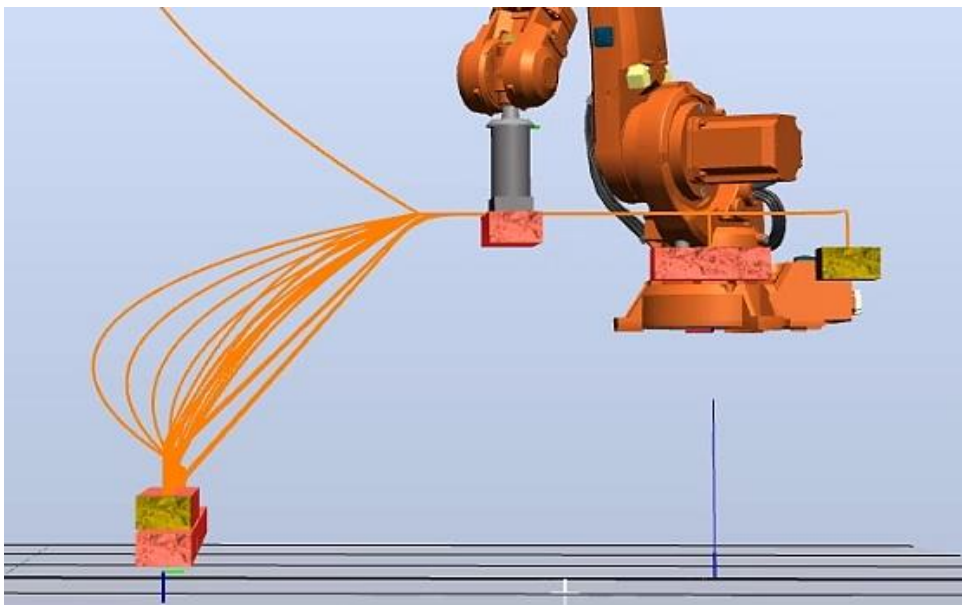
4. *BrickToCross()* - przejście z cegłą z cementem do punktu cross przed rozprowadzeniem jej na ścianę.
5. *HalfBrickToCross()* - przejście z połówką cegły z cementem do punktu cross przed rozprowadzeniem jej na ścianę
6. *SetFirstFullBrick()* - wstawienie pełnej cegły w ścianę.
7. *SetFirstHalfAndFullBrick()* - wstawienie połówki cegły w ścianę.
8. *SetRestBricks()* - wstawienie pozostałych cegieł w ścianę.
9. *SetLastHalfBrick()* - procedura mająca na celu wstawienie jako ostatniej połówki cegły w wypadku gdy cała cegła już się nie zmieści w zadanej długości ściany.

W procedurach, które kończą się umieszczeniem cegły (lub połówki) w ścianie zostały zastosowane dwa sposoby ich zakończenia w zależności od tego, czy aktualny target umieszczenia cegły w ścianie znajduje się powyżej czy poniżej punktu *cross*.

1. Gdy aktualny punkt znajduje się powyżej punktu *cross*, to chwytak z cegłą zatrzymuje się obok ściany, wsuwa się w ścianę a następnie dociska cegłę.

2. Gdy aktualny punkt znajduje się poniżej punktu *cross*, to chwytak z cegłą zatrzymuje się nad ściany, opuszcza się i dociska cegłę.

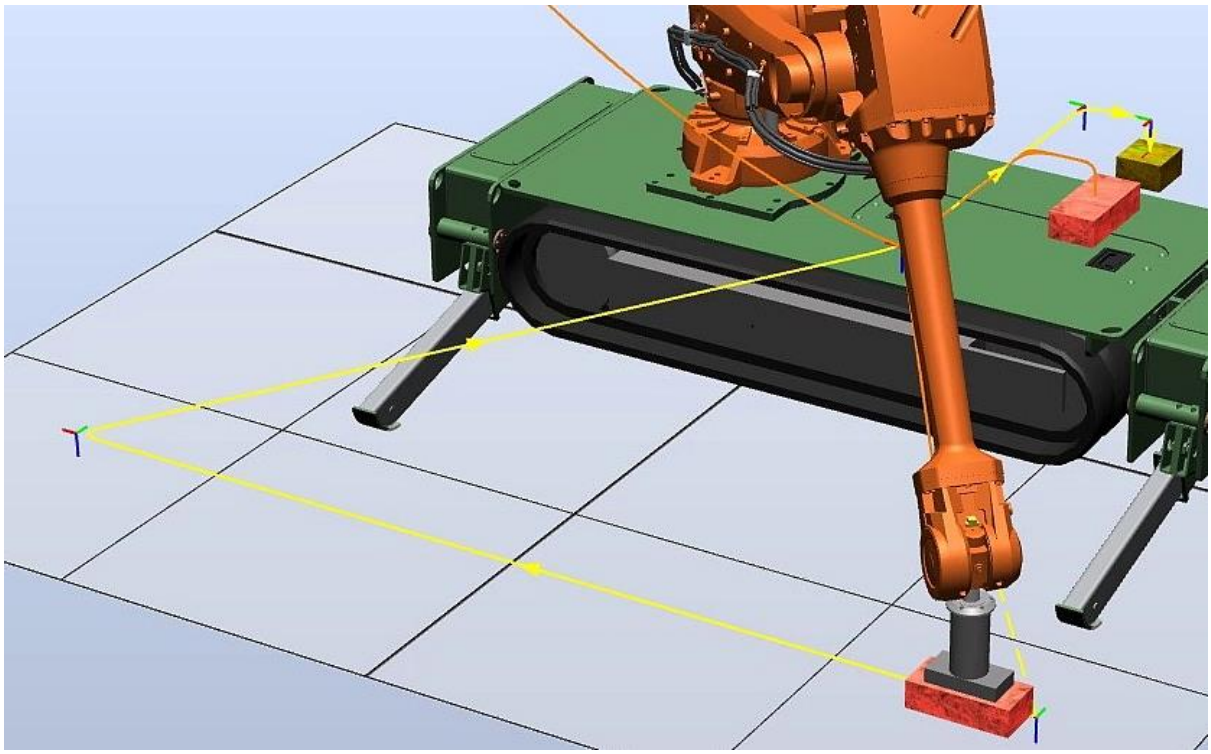
Powyższe dwa sposoby powstały jako rezultat wykonania wielu testów na symulatorze. Jeśli ściana znajduje się zbyt blisko robota, to dla cegieł poniżej punktu *cross* następuje zatrzymanie ruchu robota. Wynika to z ograniczeń spowodowanych kinematyką robota. Gdy wprowadzimy dwa sposoby wstawiania cegieł, to odległość robota od ściany może być mniejsza.



Rys.3 Cegły poniżej punktu cross

Budowana ściana nie musi być równoległa do osi OY. Może być odchylona o pewien kąt *alfa*, którego wartość musimy znaleźć, ponadto musimy znać wartość długości budowanej ściany *dlug_wall*. Wartości zmiennych *alfa* oraz *dlug_wall* możemy znaleźć na dwa sposoby:

1. Pierwszy sposób polega na pobraniu wartości dwóch targetów (pierwszej i ostatniej cegły) za pomocą FlexPendanta i umieszczeniu ich w tablicy jako elementów *point{1}* i *point{2}*.

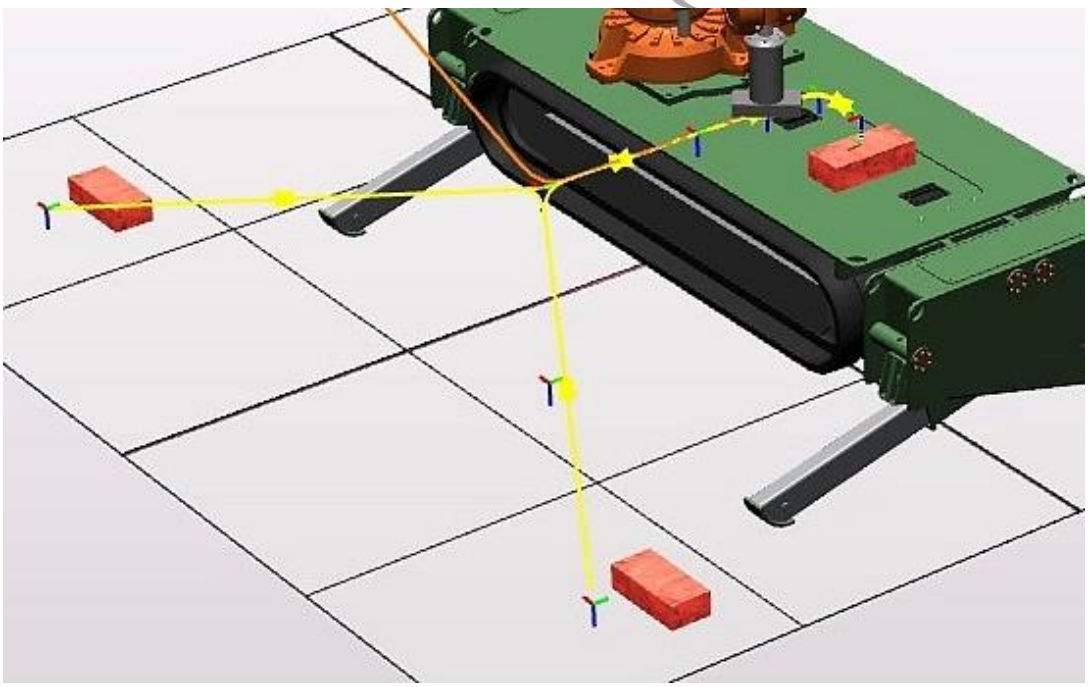


Rys.4 Pobranie współrzędnych pierwszej cegły

Następnie w procedurze *InitTargets()* obliczamy szukane wartości tych zmiennych:

```
IF opt_pomiar=OPT_TWO_POINTS THEN
        delta_z:=point{2}.trans.z-point{1}.trans.z;
        dlug_wall:=(point{1}.trans.x-point{2}.trans.x) *        point{1}.trans.x-
point{2}.trans.x) + (point{1}.trans.y-  point{2}.trans.y) * (point{1}.trans.y-point{2}.trans.y);
        dlug_wall:=Sqrt(dlug_wall)+1.5*brick_L+10;
        alfa:=ASin((point{2}.trans.x-point{1}.trans.x)/dlug_wall);
        ...
ELSE
        ...
```

Ponadto w tej procedurze obliczamy wartość zmiennej *delta_z* czyli różnicy położenia pierwszej i ostatniej cegły wzdłuż osi OZ. Wartość tej zmiennej ma wpływ na współrzędną *z* aktualnie układanej cegły.



Rys.5 Pierwsza i ostatnia cegła, których współrzędne pobieramy

2. Druga metoda polega na pobraniu za pomocą FlexPendanta współrzędnych pierwszej cegły *point{1}* i jednocześnie dokonaniu pomiaru laserem odległości do punktu końcowego muru. Kąt obrotu wokół osi OZ pierwszej cegły oraz długość ściany odczytujemy wykorzystując instrukcje w procedurze *InitLaser()*:

alfa:=EulerZYX(\Z,point{1}.rot);

...

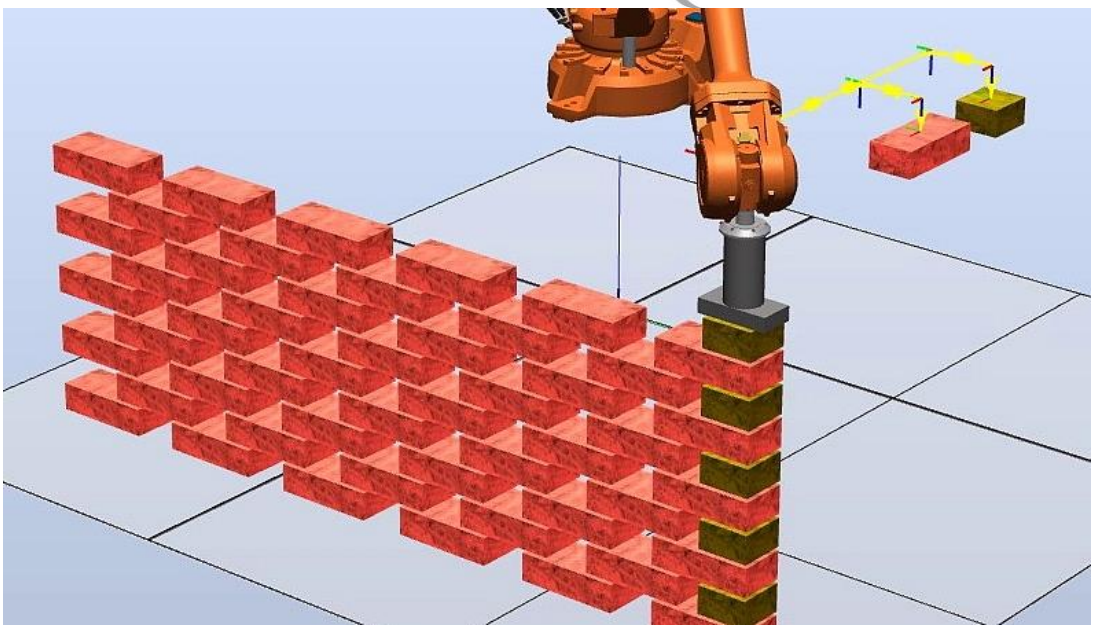
*dlug_wall:=dev_1_ai1_value*1000;*

...

Gdzie *dev_1_ail_value* to wartość typu *num* odczytana z pomiaru laserem.

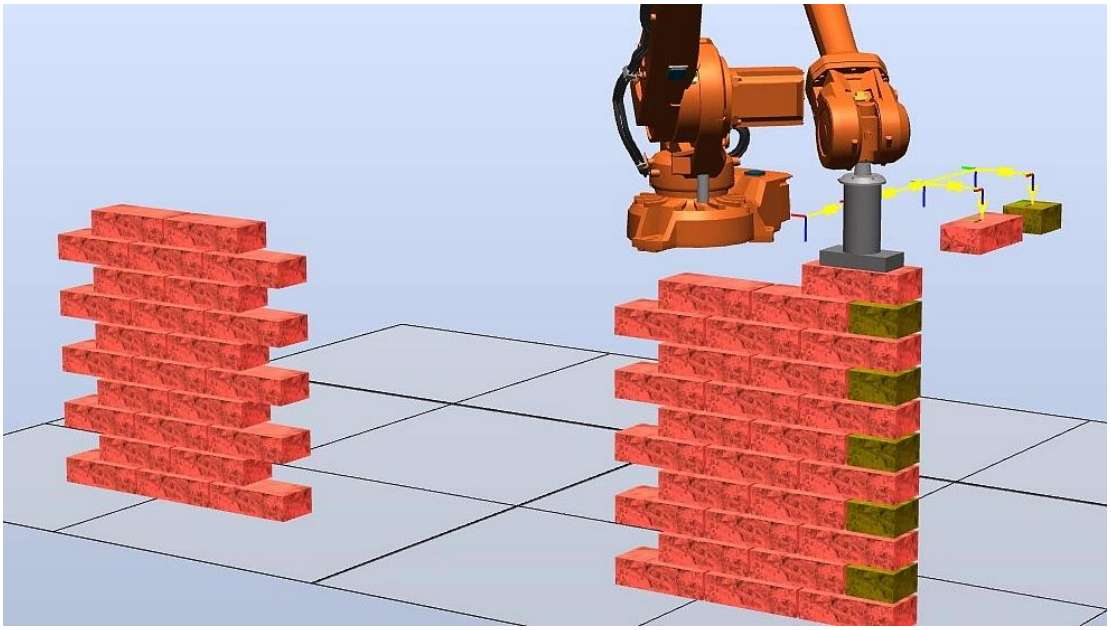
Kod programu został napisany z myślą o budowie typowej pełnej ściany w kształcie prostokąta. Można jednak zmieniać wartości różnych zmiennych w taki sposób, aby otrzymać ścianę o nietypowym, innym kształcie.

1. Jeśli do każdego przesunięcia układanej cegły dodamy pewną stałą wartość wzdłuż osi OY to otrzymamy ścianę ażurową:



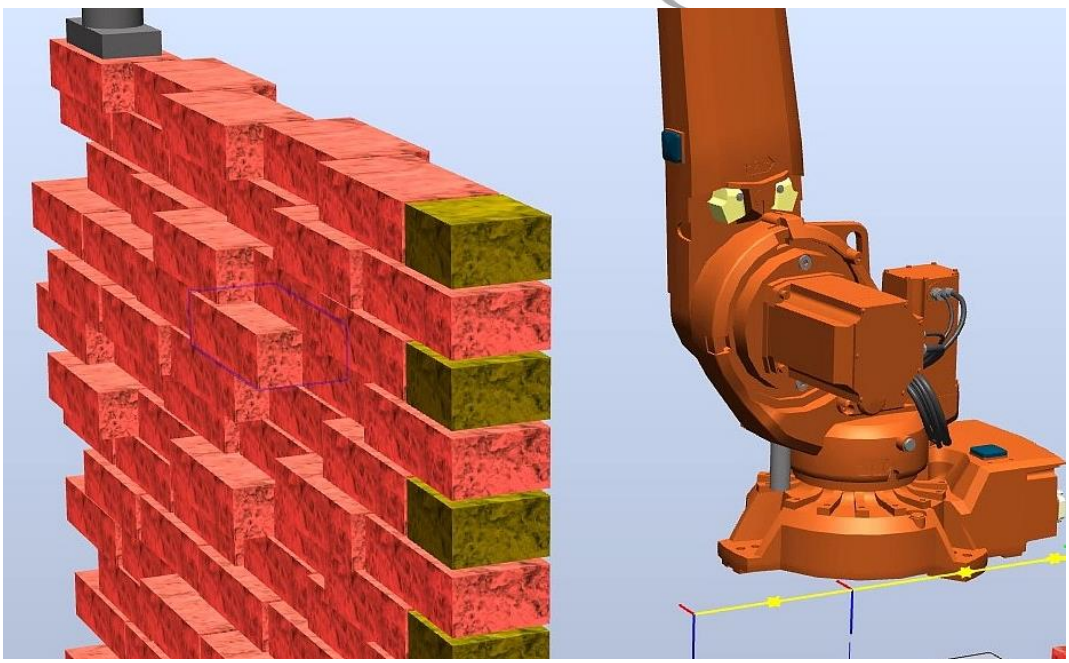
Rys.6 Ściana ażurowa

2. Jeśli umieścimy warunki logiczne, w których określimy dla jakich wartości współrzędnej *y* cegła ma być umieszczana w murze, to możemy zostawić prostokątny otwór np. na drzwi:



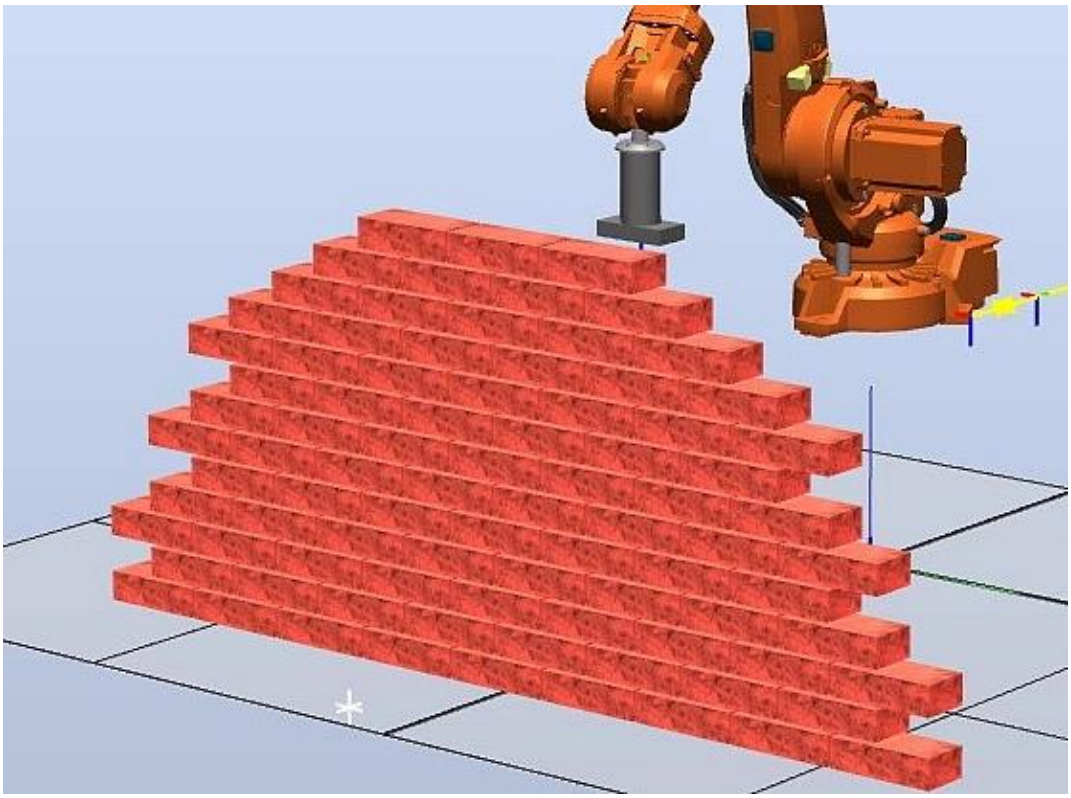
Rys.7 Ściana z prostokątnym otworem

3. Jeśli dla każdego punktu umieszczenia cegły dodamy małe losowe przesunięcie wzdłuż osi OX, to otrzymamy ścianę dekoracyjną:



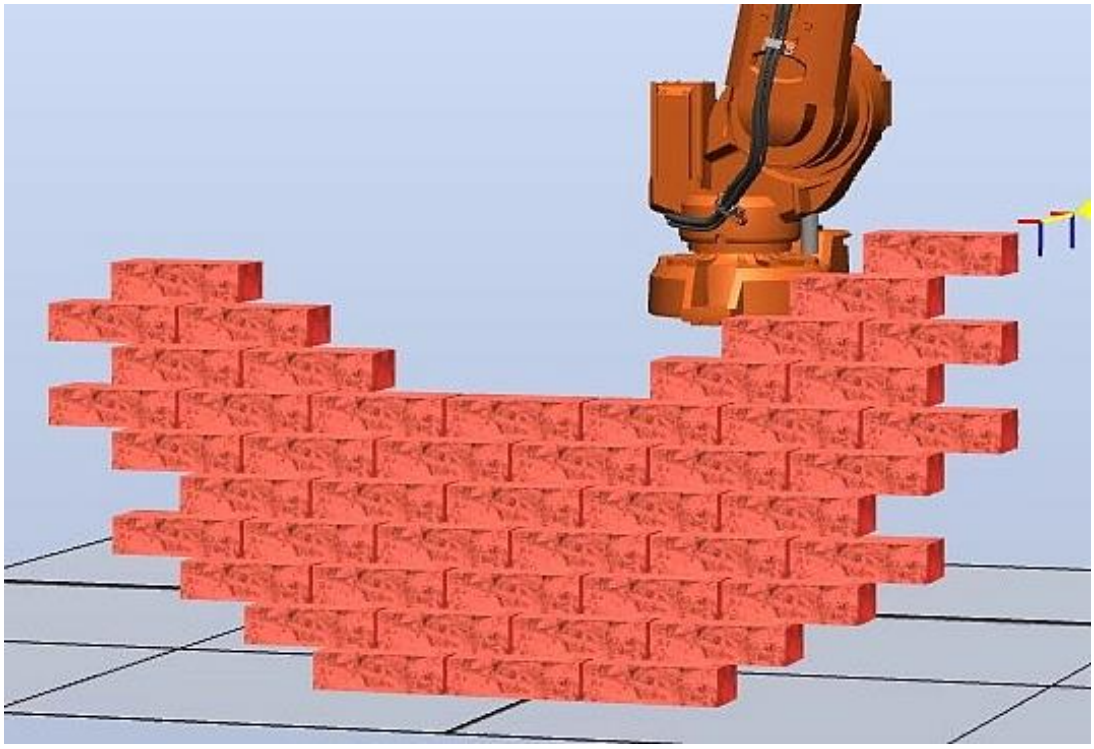
Rys.8 Ściana dekoracyjna z losowymi przesunięciami

4. Jeśli punkty wstawienia pierwszej i ostatniej cegły będą funkcją wysokości aktualnego rzędu, to możemy zbudować ścianę ograniczoną jakąś krzywą. Tutaj jest ściana w półkolu:



Rys.9 Ściana w półkolu

5. Inny przykład zastosowania ściany o kształcie krzywoliniowym z zastosowaniem odpowiednich warunków logicznych:



Rys.10 Półkole z wyciętym półkolem