Sterowanie ramieniem chwytającym

Jakub Banaś, Michał Bizoń, Hubert Gajewski,

[1,3,5]*AGH Akademia Górniczo-Hutnicza im. Stanisława Staszica*
Wydział Inżynierii Mechanicznej i Robotyki,
Katedra Robotyki i Mechatroniki,
Al. Mickiewicza 30, 30-059 Kraków
mbizon@student.agh.edu.pl, jakbans@student.agh.edu.pl,
hgajewski@student.agh.edu.pl

SŁOWA KLUCZOWE: Robot SCARA, autonomiczne ramię, robot podający cukierki, oprogramowanie robota, kinematyka prosta, kinematyka robota, algorytm sterujący

STRESZCZENIE

Niniejszy artykuł ma na celu wyjaśnienie i omówienie kwestii związanych z powstaniem robota chwytającego, a dokładniej części odpowiedzialnej za algorytm sterujący całym obiektem mechatronicznym jakim jest ramię robota. Został w nim zawarty sposób rozumowania wiodący do powstania kodu sterującego oraz proces stojący za zaprogramowaniem prymitywnej sztucznej inteligencji, która umożliwiła kontrolę nad ramieniem. Z uwagi na obszerność całego zagadnienia, artykuł skupia się głównie wokół aspektów programistycznych. W dalszej części poruszone zostają kwestie użytego oprogramowania, implementacji kodu do robota, stworzenia algorytmu obliczeniowego oraz pomocy naukowych będących niezbędnymi do stworzenia odpowiedniego software'u.

1. WSTĘP

Sterowanie ramieniem robota może być w pełni autonomiczne lub zdalne. Jednakże zgodnie z definicją robot jest kierowany przez sztuczną inteligencję [1]. W tym celu należy go odpowiednio zaprogramować umożliwiając mu kontrolę nad silnikami i mechanizmami. Używa się do tego języków programowania, które są obsługiwane i odczytywane przez komputer, czyli mózg naszego robota. Dobrym przykładem jest Arduino [2] czyli prosty komputer z możliwością obsługi jednocześnie kilku serwomechanizmów, używany głównie w modelarstwie, który ma bardzo łatwy i przystępny język oparty na C (więcej języków [3]). Omawiany projekt opiera się o gotowy prototyp ramienia stworzony przez segment elektroniczny grupy projektowej. W przedstawionym ramieniu głównym wyzwaniem było napisanie programu, który na podstawie położenia przedmiotu, obliczy o jaki kąt poszczególne serva mają się obrócić aby ramię chwyciło przedmiot.

1. Treść właściwa

Tę część poświęcono na bliższe zapoznanie z programem, który wyszukuje optymalne położenie ramienia w układzie kartezjańskim. Zadaniem sekcji programistycznej było zmierzenie się z problemem napisania kodu do gotowego prototypu ramienia posiadającego pięć złączy obrotowych, w których dwa z nich kontrolowane są poprzez serva o zakresie ruchu od 0o do 180o, a trzecie usytuowane jest na łożysku przez co chwytak ustawiony jest w pozycji pionowej poprzez grawitację, czwarte kontrolowane servo o mniejszym zakresie ruchu odpowiedzialne jest za zaciskanie szczęk chwytaka oraz ostatnim sterowanym poprzez silnik krokowy obracający całość ramienia. W celu ułatwienia pracy podzielono ten układ na trzy części. Pierwsza operująca silnikiem krokowym liczącym odpowiedni kąt obrotu całego ramienia oraz liczącą odległość końca chwytaka od osi obrotu ramienia, odległość ta będzie potrzebna do działania części drugiej, która to z kolei operuje dwoma servami. Jej zadaniem jest wyznaczenie optymalnego położenia końca chwytaka w płaszczyźnie prostopadłej do płaszczyzny na której znajduje się cały obiekt mechatroniczny. Trzecia zajmująca się zaciskaniem oraz zwalnianiem chwytaka w odpowiednim momencie.

Część pierwsza:

Aby zapewnić ramieniu możliwości swobodnego ruchu w trójwymiarze dodano na podstawie silnik krokowy. Jego praca umożliwiła obrót pionową płaszczyzną ramienia, w której ono pracowało, co zapewniło w łatwy sposób możliwość osiągnięcia każdego punktu wokół ramienia. Prototyp miał zamontowany ostatni segment ramienia lekko odchylony od osi całego ramienia wobec czego niezbędna była poprawka na kąt o jaki ma się obrócić ramię aby dotarło we wpisane współrzędne. Do kalibracji posłużono się prostą trygonometrią oraz funkcjami cyklometrycznymi. Mając odpowiednie długości ramienia i odchylenia części z chwytakiem obliczono arcus tangens małego kąta, o który trzeba było zwiększyć lub zmniejszyć ruch obrotowy ramienia. Następnie podzielono płaszczyznę na 4 ćwiartki. Dla każdej z ćwiartki układu współrzędnych ułożono oddzielne instrukcje warunkowe, które obliczały kąt o jaki ramię ma się obrócić, następnie z postaci radialnej algorytm zamieniał kąt na stopnie, a z tego na kroki silnika. Skorzystano z silnika krokowego o 800 stopniach wobec czego dodawana lub odejmowana wartość 200 (rysunek poniżej) odpowiada 90 stopniom, a co za tym idzie zmianie ćwiartki.

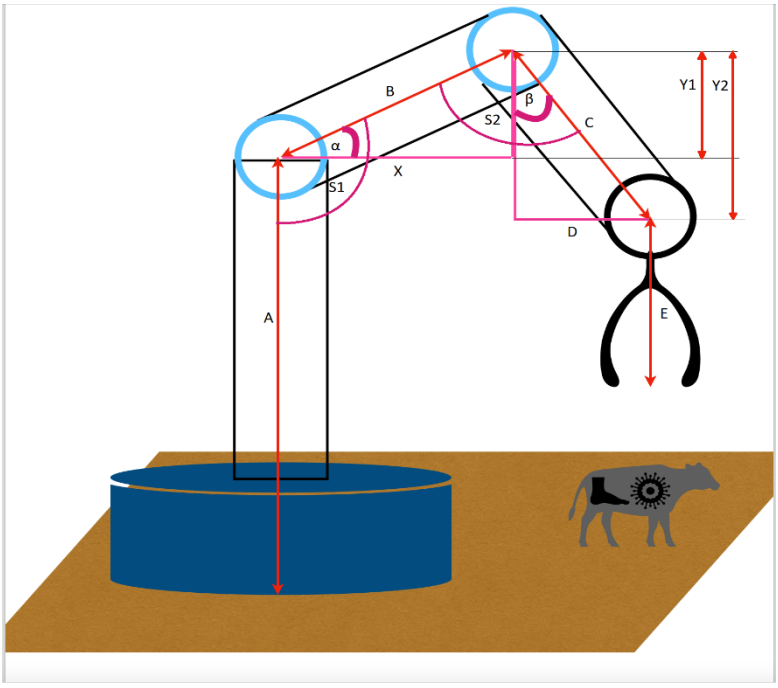
```
 przeciw=sqrt(x*x+y*y);

 if(x>0 && y<0){
      kat = atan(y1/x1 + male/przeciw);
katy=(kat*180)/PI;
      kroki = (katy*800/360);
 }
  if(x<0 && y<0){
      kat = atan(x1/y1 + male/przeciw);//sprawdzic
katy=(kat*180)/PI;
      kroki =200 + katy*800/360;
 }
 if(x>0 && y>0){
      kat = atan(y1/x1 - male/przeciw);
katy=(kat*180)/PI;
      kroki = -(katy*800/360);
 }
  if(x<0 && y>0){
      kat = atan(x1/y1 - male/przeciw);//sprawdzic
katy=(kat*180)/PI;
      kroki = -(katy*800/360)-200;
 }
```

Rysunek 1 Część kodu odpowiedzialna za obrót ramienia

Część druga:

Aby zrozumieć w jaki sposób określono zależność położenia ramienia od kątów poszczególnych serv, należy sobie dla ułatwienia przyjąć położenie chwytaka jako punktu w układzie kartezjańskim. Wyobrażeniu ułatwi schemat przedstawiony na rysunku poniżej (patrz rysunek 1) zawierający poglądowy schemat ramienia.



Rysunek 2 Poglądowy schemat ramienia.

Podzielono następnie geometrie ruchu ramienia na trójkąty. Zakładając, że alfa jest równe wartości bezwzględnej alfa - 90 stopni. (Przypadek 1) Jeżeli S1 jest większe bądź równe kątowi 90 stopni, dodatkowo jeżeli S2 + alfa jest mniejsze od 180 stopni, wtedy beta jest równa wartości bezwzględnej S2 - 90 stopni + alfa. (Przypadek 2) Jeżeli S1 jest większe bądź równe kątowi 90 stopni, dodatkowo jeżeli S2 + alfa jest większe bądź równe 180 stopni, wtedy beta jest równa wartości bezwzględnej S2 - 180 stopni + alfa. (Przypadek 3) Jeżeli S1 jest mniejsze od kąta 90 stopni, oraz S2 jest mniejsze bądź równe alfa, wtedy beta równa jest wartości bezwzględnej alfa - S2.(Przypadek 4) Jeżeli S1 jest mniejsze od kąta 90 stopni, oraz S2 jest większe niż alfa, wtedy beta równa jest wartości bezwzględnej S2 - 90 stopni + alfa. Korzystając z tak obliczonych kątów jest się już w stanie obliczyć odpowiednio funkcjami sinus i cosinus odległości (patrz rysunek 1) X, D, y1 ,y1; używając z odpowiednich zmierzonych wcześniej długości modelu (patrz rysunek 1) A, B, C, E.

Część trzecia:

Trzecia część obejmuje kod sterujący servem zaciskającym łapki chwytaka. Musimy wziąć pod uwagę fakt, iż servo to ma ograniczony zakres zacisku oraz rozwarcia. Gdy zaciśniemy je zbyt mocno chwytany obiekt albo część chwytaka może ulec uszkodzeniu co uniemożliwi dalsze w pełni poprawne i zamierzone przez nas działanie całego prototypu. Zbyt duże rozwarcie również może uszkodzić części naszego ramienia. Stwierdziliśmy również fakt, że pełne rozwarcie nie jest dla nas optymalne i jedynie zajmowało by dodatkowy czas oraz zwiększało pobór prądu poprzez zupełnie niepotrzebny zakres ruchu, który musiałoby wykonać servo dwukrotnie podczas rozwarcia oraz zacisku. Optymalne położenie serva dla zacisku oraz rozwarcia wyznaczyliśmy empirycznie. Testując położenie łapek chwytaka po zadaniu servu odpowiednich kątów obrotu byliśmy w stanie zobaczyć jak zadane kąty obrazują odległość pomiędzy łapkami. Wykonując liczne próby doszliśmy do wniosku, że optymalne kąty dla rozwarcia i zacisku to odpowiednio 80 i 54. Cała operacja została zawarta w pętli for aby uniknąć uszkodzeń mechanicznych wcześniej przygotowanej konstrukcji prototypu. Zbyt szybki ruch elementów chwytaka może spowodować deformację struktury tej części obiektu mechatronicznego. Zbyt wolny zaś wiąże się ze spadkiem efektywności procesu. Zastosowana w pętli funkcja „delay" o odpowiedniej wartości ma na celu optymalizację ruchu. Możemy wyodrębnić dwie zasadnicze części jedna obejmuje zaciskanie, a druga zajmuje się rozszerzaniem łapek chwytaka. Istotną częścią kodu jest umiejscowienie formuły sterującej procesem chwytającym w odpowiednim miejscu co jest gwarantem poprawnie wydawanych instrukcji dla istniejących rozwiązań zastosowanych w prototypie.

```
if(nom<80){
   for(nom;nom<=80;nom++)
      {
      S3.write(nom);
      delay(30);
      } //koniec fora
      }
    check(przeciw, wysokosc, katServ);
  idzie(katServ[0],katServ[1],30);
   delay(500);
   if(nom>54){
   for(nom;nom>=54;nom--)
      {
      S3.write(nom);
      delay(30);
      } //koniec fora
      }
```

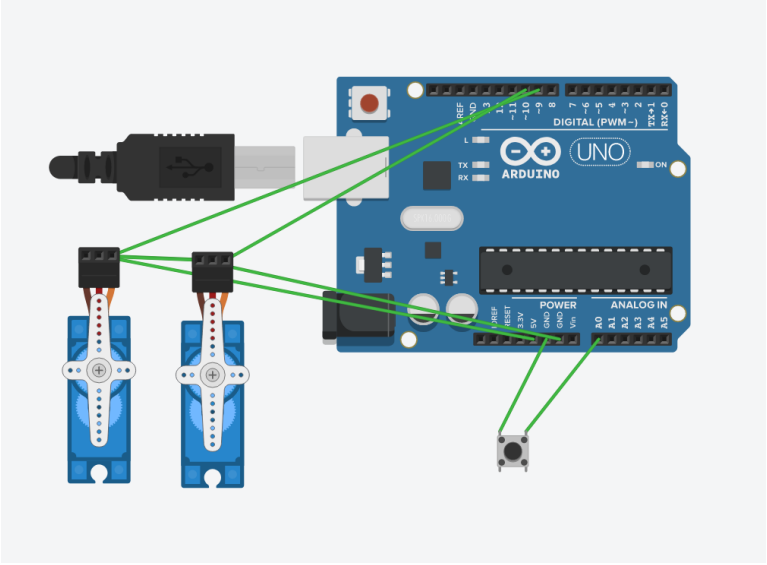
Rysunek 3 Część kodu odpowiedzialnego za ruch chwytaka.

TABLICA MORFOLOGICZNA									
Nr	Problem/Funkcja	Rozwiązanie możliwe							
Ramię chwytające cukierka - Część programistyczna									
1.	**Język programowania**	C	C++	Java	Pan Python	Arduino	kod blokowy (Tinkercad)		
	Kryterium 1	Niska	Niska	Niska	Niska	Niska	Niska		
	Kryterium 2	Wysoka	Średnia	Średnia	Średnia	Średnia	Niska		
	Kryterium 3	Wysoka	Wysoka	Średnia	Średnia	Średnia	Wysoka		
	Kryterium 4	Wysoka	Wysoka	Średnia	Średnia	Wysoka	Niska		
	Kryterium 5	Wysoka	Wysoka	Wysoka	Wysoka	Wysoka	Niska		
2.	**System operacyjny**	Android	Linux	Windows	FreeRTOS				
	Kryterium 1	Niska	Niska	Wysoka	Niska				
	Kryterium 2	Niska	Niska	Średnia	Niska				
	Kryterium 3	-	-	-	-				
	Kryterium 4	Wysoka	Wysoka	Średnia	Wysoka				
	Kryterium 5	Wysoka	Średnia	Średnia	Wysoka				
3.	**Kompilator kodu**	VisualStudi	Notatnik	Notepadd++	Arduiono IDE	Codeblock	Dev	GNU Compiler collection	
	Kryterium 1	Niska	Niska	Niska	Niska	Niska	Niska	Niska	
	Kryterium 2	Średnia	Niska	Niska	Średnia	Średnia	Średnia	Średnia	
	Kryterium 3	-	-	-	-	-	-	-	
	Kryterium 4	Wysoka	Niska	Niska	Niska	Średnia	Średnia	Niska	
	Kryterium 5	Wysoka	Niska	Średnia	Średnia	Średnia	Średnia	Niska	
	Kryterium 6	Wysoka	Średnia	Średnia	Średnia	Średnia	Średnia	Średnia	
4.	**Środowisko współpracy**	GitHub	Poczta Email	Spotkania IRL	Discord	Teams	Skype		
	Kryterium 1	Niska	Niska	Średnia	Niska	Średnia	Wysoka		
	Kryterium 2	Niska	Średnia	Wysoka	Niska	Niska	Średnia		
	Kryterium 3	-	-	-	-	-	-		
	Kryterium 4	Wysoka	Niska	Niska	Średnia	Niska	Niska		
	Kryterium 5	Wysoka	Niska	Niska	Średnia	Niska	Niska		
5.	**Sterowania**	Manualne	Automatyczne	Pół automatyczne					
	Kryterium 1	Niska	Wysoka	Średnia					
	Kryterium 2	Niska	Wysoka	Średnia					
	Kryterium 3	-	-	-					
	Kryterium 4	Niska	Wysoka	Średnia					
	Kryterium 5	Niska	Wysoka	Średnia					
6.	**Medium tranmisyjne**	Przewody	Bluethooth	Wi-FI	Podczerwień	Port Seryjny	Sterowanie dźwiękowe		
	Kryterium 1	Niska	Średnia	Średnia	Wysoka	Niska	Wysoka		
	Kryterium 2	Niska	Średnia	Średnia	Wysoka	Niska	Wysoka		
	Kryterium 3	-	-	-	-	-	-		
	Kryterium 4	Wysoka	Średnia	Średnia	Niska	Wysoka	Niska		
	Kryterium 5	Niska	Wysoka	Wysoka	Niska	Średnia	Średnia		

Rysunek 4 Tablica morfologiczna

Stworzenie tablicy morfologicznej ułatwiło podjęcie decyzji, a także jasno pokazuje powód, dla którego zdecydowano się na dokładnie takie rozwiązanie dotyczące wyboru odpowiedniego środowiska programowania czy też konkretnych podzespołów. Dzięki dodaniu kilku najważniejszych kryteriów cześć programistyczna jasno określiła najbardziej znaczące aspekty podczas doboru podzespołów.

Przeprowadzono symulację na platformie Tinkercad.



Rysunek 5 Schemat dwóch serv w Tinkercadzie.

W wyniku wielokrotnie przeprowadzonych testów poprzez manipulowanie zmiennymi oraz wymianą kodu udało się dojść do działającej wersji programu, która została przekazana do dalszych prac sekcji elektronicznej, a także została skalibrowana i przystosowana do poprawnego działania z prototypem.

Uwagi końcowe

Dalszy rozwój oprogramowania może się skupić na optymalizowaniu ruchu ramienia robota, w celu dobrania najbardziej korzystnej drogi do pokonania. Planowana jest także implementacja kodu odpowiedzialnego za kontrolę nad pracą podajnika do cukierków, a także obsługa zestawu wyświetlacz + klawiatura w celu łatwiejszego i szybszego wprowadzania danych do ramienia, co za tym idzie zwiększenia efektywności przeprowadzonych eksperymentów. Po kilku pierwszych próbach zrezygnowaliśmy także z prowadzenia symulacji na platformie Tincercad. Bardzo przydała się na początku, aby sprawdzić czy dobrze wszystko jest podpięte, a także aby przetestować działania poszczególnych funkcji z biblioteki. Jednakże, jest to program działający w przeglądarce, co za tym idzie ma bardzo mało pamięci i skutecznie spowalniał pracę nad programem, gdyż obliczenia zajmowały lwią część czasu pracy.

Bibliografia

[1]Robot - Wikipedia: https://pl.wikipedia.org/wiki/Robot odwiedzono w dniu: 07.05.2021

[2]Arduino dokumentacja: https://www.arduino.cc/reference/en/ odwiedzono w dniu: 07.05.2021

[3]Lista języków do porogramowania AI: https://www.analyticsinsight.net/what-are-the-best-programming-languages-for-artificial-intelligence/ odwiedzono w dniu: 07.05.2021

Stosowane oprogramowanie:

- Autodesk Fusion 360: https://www.autodesk.com/products/fusion-360/overview
- Libre Office: https://pl.libreoffice.org
- Inkscape: https://inkscape.org
- Gimp: https://www.gimp.org
-Tinkercad: https://www.tinkercad.com
-Arduino: https://www.arduino.cc
-Github: https://github.com

Za wyjątkiem Autodesk Fusion 360 (darmowe na 3 lata w wersji edukacyjnej i darmowe na 30 dni w wersji komercyjnej), oraz Autodesk Eagle (darmowa wersja do użytku dla hobbystów, z ograniczoną powierzchnią obszaru roboczego), pozostałe programy są darmowe do użytku zarówno komercyjnego jak również edukacyjnego.