



WYDZIAŁ ELEKTRYCZNY

KIERUNEK: AUTOMATYKA I ROBOTYKA

Szymon Wąchała

Nr albumu: 250442

**Eksperymentalny robot ramieniowy sterowany za pomocą
mikrokontrolera ARM**

Experimental arm robot controlled by ARM microcontroller

**INŻYNIERSKA PRACA
DYPLOMOWA**

Studia: stacjonarne

Opiekun: dr inż. Krzysztof Dyrcz

Katedra: Katedra maszyn, Napędów i Pomiarów Elektrycznych

.....

Ocena

.....

data, podpis opiekuna

Wrocław, 2021 r.

Spis treści

1.	Wstęp	4
2.	Cel i zakres pracy	5
3.	Roboty przemysłowe.....	5
3.1.	Wprowadzenie.....	5
3.2.	Rodzaje robotów.....	7
3.3.	Napędy stosowane w robotach	10
3.4.	Zastosowania robotów	11
4.	Projekt robota ramieniowego	13
4.1.	Założenia projektowe.....	13
4.2.	Koncepcja robota	13
4.3.	Określenie kinematyki ramienia.....	15
4.4.	Sposób doboru układu napędowego	21
4.4.1.	Obliczenia obciążeń dla napędów.....	24
4.4.2.	Opis teoretyczny wykorzystanych układów napędowych	31
4.5.	Konstrukcja ramienia	34
5.	Program sterujący robotem.....	59
5.1.	Omówienie aplikacji komputerowej do sterowania robotem	61
5.2.	Omówienie programu wykonawczego mikrokontrolera	80
6.	Badania laboratoryjne	86
7.	Podsumowanie	90
	Literatura	91
	Załącznik 1	93

1. Wstęp

Tematem niniejszej pracy inżynierskiej jest zaprojektowanie oraz wykonanie eksperymentalnego robota ramieniowego. Grupa robotów, do której można zaklasyfikować taką maszynę to roboty przemysłowe. Analiza rynku pod kątem dostępności tego typu rozwiązań pokazuje, że wielu wiodących producentów automatyki posiada w swoich ofertach roboty przemysłowe. Urządzenia te są konstruowane tak, aby zapewnić jak największą uniwersalność oraz sprostać wielu potrzebom w nowoczesnych procesach przemysłowych. Ze względu na te cechy, firmy produkujące roboty wraz z urządzeniami oferują również bardzo rozbudowane środowiska do obsługi robotów, co niestety przekłada się to bezpośrednio na wysokie koszty zakupu. Nabywca robota przemysłowego nie płaci wówczas wyłącznie za urządzenie, ale znaczna część kosztów przeznaczona jest na dedykowane specjalistyczne oprogramowanie, które zapewnia duże możliwości, jednak w przypadku prostych implementacji robotów nie jest w pełni wykorzystywane. W związku z tym, jako główny cel pracy przyjęto zaprojektowanie oraz wykonanie robota ramieniowego, który będzie charakteryzował się prostym układem sterowania, pozwalającym na wykonanie prostych operacji typu „*pick and place*”. Robot wraz z takim oprogramowaniem będzie mógł zostać wykorzystany w celu np. przenoszenia elementów z jednej taśmy produkcyjnej na inną lub wykonywania prostych zadań pakowania elementów. Jednym z podstawowych założeń pracy jest wykonanie robota oraz dedykowanego dla niego oprogramowania w taki sposób, aby całość była tańsza od spotykanych przemysłowych rozwiązań.

2. Cel i zakres pracy

Celem pracy dyplomowej jest opracowanie i wykonanie eksperymentalnego robota ramieniowego wykorzystującego w układzie sterowania mikrokontroler z rodziny ARM.

Zakres pracy obejmuje:

- zapoznanie się z zagadnieniem,
- opracowanie koncepcji robota ramieniowego,
- dobór elementów robota,
- wykonanie prototypu robota i napisanie oprogramowania sterującego,
- wykonanie badań laboratoryjnych,
- redakcję pracy.

Praca składa się z 7 rozdziałów. Pierwsze dwa rozdziały zawierają wstęp do pracy oraz jej cel i zakres. Trzeci rozdział zawiera informacje wprowadzające do zagadnienia robotów przemysłowych. W czwartym rozdziale zawarty został proces modelowania robota oraz jego docelowa budowa. Piąty rozdział opisuje sposób sterowania robotem, zawiera również opisy programów sterujących. Szósty rozdział ma charakter badawczy, poruszone są w nim zagadnienia poboru mocy oraz oceny jakości ruchu robota. Ostatnim rozdziałem jest część zawierająca podsumowanie całości pracy.

3. Roboty przemysłowe.

3.1. Wprowadzenie

Robotyka jest to dziedzina nauki obejmująca zagadnienia takie jak sterowanie ruchem układów, inteligencja maszynowa, analiza z wykorzystaniem licznych sensorów, mechanika oraz projektowanie robotów [1]. W miarę rozwoju robotyki ludzkość zaczęła zdawać sobie sprawę jakie niebezpieczeństwa ona ze sobą niesie. W celu ich zapobiegnięcia sformułowano pięć podstawowych praw robotyki [1]:

1. *Robot nie może ingerować w działanie człowieka, oprócz tych działań, które szkodzą człowiekowi.*
2. *Robot musi być posłuszny rozkazom wydawanym przez człowieka, oprócz tych rozkazów, które są sprzeczne z pierwszym prawem.*
3. *Robot musi chronić swoją egzystencję, oprócz tych przypadków, które są sprzeczne z pierwszym lub drugim prawem.*
4. *Robot musi ujawniać swoją naturę robota. W szczególności robot nie może udawać człowieka.*
5. *Im bogatsze jest wyposażenie robota w układy sensoryczne, zapewniające percepcję warunków otoczenia, a w szczególności możliwości autonomicznego określania działań przez jego układ sterowania, tym — do pewnego dopuszczalnego stopnia — może byćuboższa, mniej złożona jego konstrukcja. Ten dopuszczalny stopień jest zależny od celu, który został przed robotem postawiony oraz od możliwości zrealizowania tego celu przez robota.*

Ciekawostką jest fakt, że autorem pierwszych trzech praw był pisarz science fiction – Isaac Asimov. W swojej książce „*Runaround*” porusza zagadnienia takie jak rozwój świata przyszłości, którego nieodłączną częścią stały się roboty oraz wysoce rozwinięta technologia. Ze względu na obawy jakie niosło ze sobą obcowanie z takimi tworami, konieczne było wprowadzenie pewnych ograniczeń i praw, tak by ludzkość nie została zniszczona przez ich własny wytwór. Prawa które pierwotnie stworzył Asimov zostały z biegiem czasu zaadaptowane do bieżących potrzeb. Dodatkowo zostały sformułowane nowe prawa, które również zostały dostosowane do bieżącego postępu technologicznego [5].

Słowo robot wywodzi się z Czech. Pierwotnie słowem „*robota*” Czesi nazywali trudną oraz ciężką pracę. Człowiekiem, który pierwszy raz posłużył się tym słowem był również Czeski malarz i poeta Josef Čapek. Jak widać etymologia tego słowa nie jest przypadkowa, ponieważ obecnie roboty są maszynami, które często są stosowane często do ciężkich i wymagających prac, gdzie człowiekowi mogłoby zagrażać bezpieczeństwo.

Jedną z bardzo istotnych kategorii robotów są roboty wykorzystywane w przemyśle. Grupa tych urządzeń nazywana jest robotami przemysłowymi.

Definicja robota według normy ISO/TR 8373:1988 (ang. *Manipulating industrial robots — Vocabulary*) brzmi: „*Manipulacyjny robot przemysłowy jest automatycznie sterowaną, programowaną, wielozadaniową maszyną manipulacyjną o wielu stopniach swobody, posiadającą własności manipulacyjne lub lokomocyjne, stacjonarną lub mobilną, dla ważnych zastosowań przemysłowych*”.

Innymi słowy, robot przemysłowy jest to specjalizowana maszyna zdolna do wykonywania konkretnych, często złożonych ruchów dzięki wielu stopniom swobody. Elementem wykonawczym robota jest efektor. Pojęciem tym określa się chwytaki oraz dodatkowe narzędzia jak np. laser, uchwyt spawalniczy czy pistolet do malowania. Robot wykonuje ruchy oraz dodatkowe działania według odpowiednio zaprogramowanych działań [2].

Aby maszynę można było nazwać robotem konieczne jest wyodrębnienie w jej budowie pewnych cech. Nieodłącznymi elementami każdego robota są [3]:

- Zespół napędowy – są to układy napędowe oraz odpowiednie układy przeniesienia napędu odpowiedzialne za ruch robota np. silniki wraz z odpowiednimi przekładniami.
- Zespół czujników – są to wszystkie czujniki potrzebne robotowi do prawidłowego poruszania się oraz reagowania na inne elementy otoczenia.
- Efektor – element wykonawczy robota.
- Układ sterujący – „mózg” robota, jest odpowiedzialny za:
 - odpowiednie wysterowanie układów napędowych,
 - zbieranie informacji z czujników,
 - komunikację z innymi maszynami współpracującymi podczas danego procesu,
 - interpretację zadanego programu oraz praca według niego.

Najważniejszymi cechami oraz parametrami jakie opisują roboty są [7]:

- własności geometryczne – jest to zbiór cech opisujących możliwości ruchowe robota w tym charakter ruchów, ilość stopni swobody,
- generacja robota,
- sposób programowania,

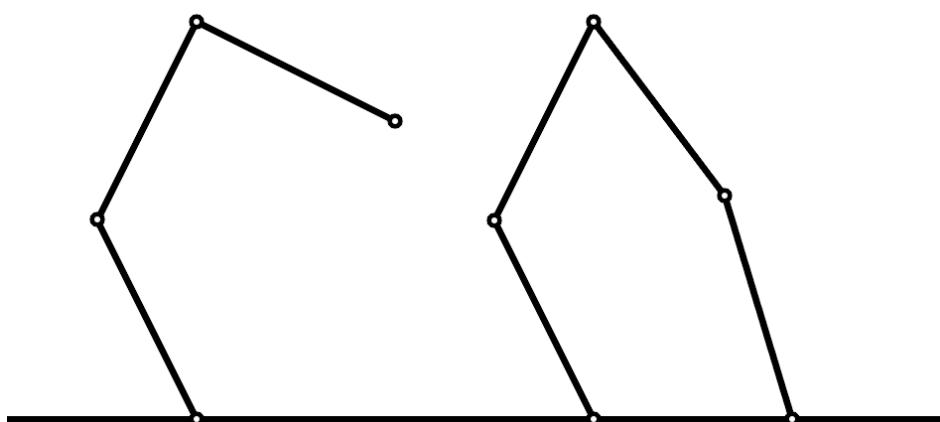
- udźwig – maksymalna dopuszczalna waga jaką może mieć obiekt podnoszony przez robota,
- możliwość swobodnego poruszania się całej konstrukcji,
- dokładność oraz powtarzalność ruchów – jest to precyza z jaką robot może osiągnąć zadany punkt przy wielokrotnym do niego powrocie,
- prędkość poruszania się efektora.

Możliwe jest wyróżnienie większej ilości cech, jednak wyżej wymienione wystarczą do wykonania najważniejszych klasyfikacji.

3.2. Rodzaje robotów

Pośród robotów można wyróżnić wiele różnych konstrukcji. Każda z nich charakteryzuje się innymi cechami, które pozwalają dopasować konkretny rodzaj robota do danego zadania.

Podstawowym podziałem robotów jest podział na roboty mobilne oraz stacjonarne. Struktura kinematyczna robota jest to najczęściej schemat przedstawiający łańcuch kinematyczny robota czyli zespół połączonych ze sobą par kinematycznych (para kinematyczna to połączenie ruchowe dwóch sąsiednich członów za pomocą złącza). W przypadku robotów stacjonarnych są one zbudowane z modułów połączonych ze sobą przez pary kinematyczne. W zależności od sposobu połączenia modułów uzyskuje się zamknięty (połączenie równoległe) lub otwarty (połączenie szeregowe) łańcuch kinematyczny. W przypadku robotów mobilnych można dokonać podziału na roboty autonomiczne, które same potrafią decydować jak będzie wyglądał tor ich ruchu oraz na roboty poruszające się po stałym, ścisłe określonym torze jezdny. Na rysunku 2.1 pokazano schematy łańcuchów kinematycznych:

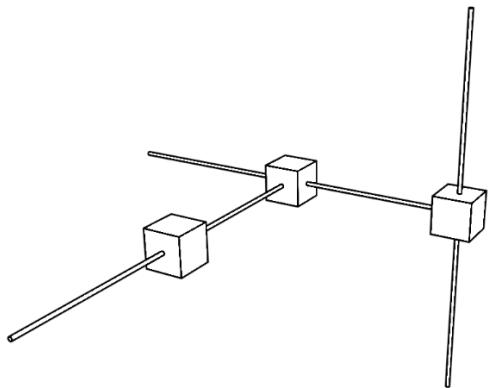


Rys. 3.1. Otwarty łańcuch kinematyczny po lewej stronie oraz zamknięty łańcuch kinematyczny po prawej stronie

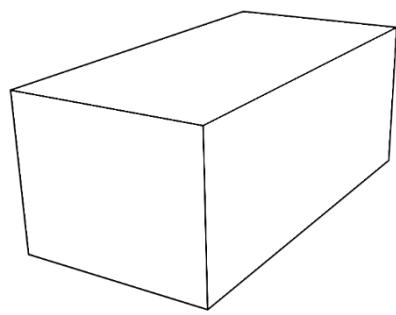
W grupie robotów stacjonarnych wyróżniamy [7]:

A. Robot w konfiguracji kartezańskiej (PPP)

a)



b)

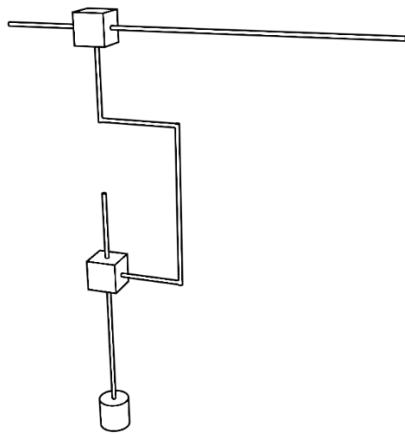


Rys. 3.2. Schemat robota PPP a) oraz kształt jego przestrzeni roboczej b)

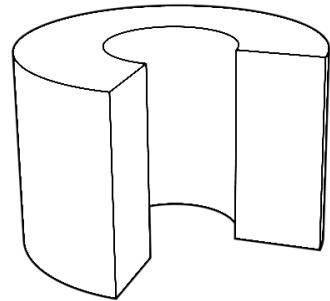
Skrótowy opis PPP oznacza, że budowa robota oparta jest o 3 przeguby przesuwne. Dzięki takiej konfiguracji robot ten porusza się według współrzędnych kartezańskich. Przestrzeń robocza takiego robota wyznaczona jest przez prostopadłościan. Układ ten charakteryzuje się zaletami w postaci 3 liniowych napędów, które przekładają się na dużą łatwość w programowaniu, tworzeniu wizualizacji pracy oraz dają dużą sztywność układu. Wadą tego układu jest fakt, że taki robot zajmuje dużo miejsca, ze względu na charakter pracy [7].

B. Robot w konfiguracji cylindrycznej (OPP)

a)



b)

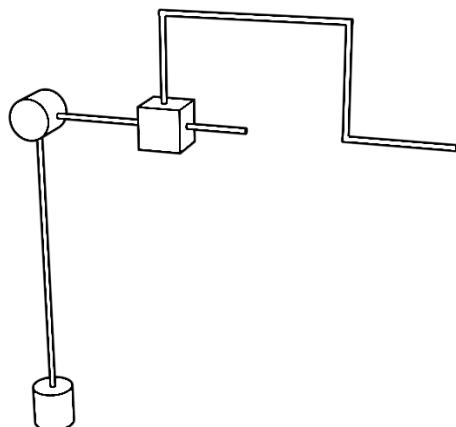


Rys. 3.3. Schemat robota OPP a) oraz kształt jego przestrzeni roboczej b)

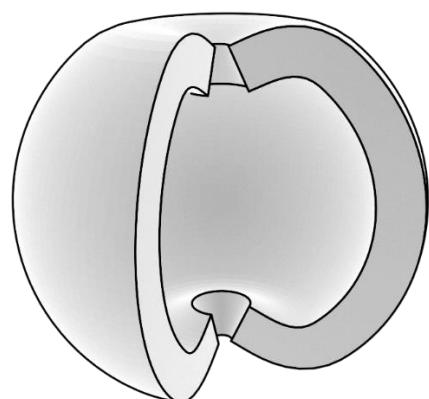
Skrótowy opis OPP oznacza, że budowa robota wykorzystuje 1 przegub obrotowy oraz 2 przeguby przesuwne. Taka konfiguracja pozwala na poruszanie się według współrzędnych walcowych, a przestrzeń robocza takiego robota wyznaczana jest poprzez walec z „wyciętym” środkiem. Dzięki wprowadzeniu jednego przegubu obrotowego możliwe staje się bezproblemowe poruszanie się po trajektorii okręgu wokół robota co pozwala na ograniczenie rozmiarów robota oraz przekłada się na mniejszą przestrzeń którą robot potrzebuje do pracy. W takiej konfiguracji niemożliwa jest jednak sytuacja, w której efektor znajduje się nad manipulatorem, w związku z tym układ jest problematyczny w omijaniu przeszkód [7].

C. Robot w konfiguracji sferycznej (OOP)

a)



b)

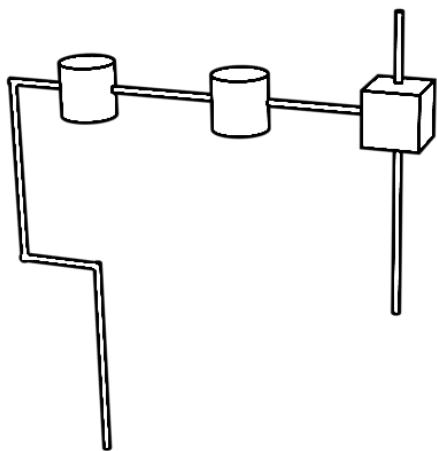


Rys. 3.4. Schemat robota OOP a) oraz kształt jego przestrzeni roboczej b)

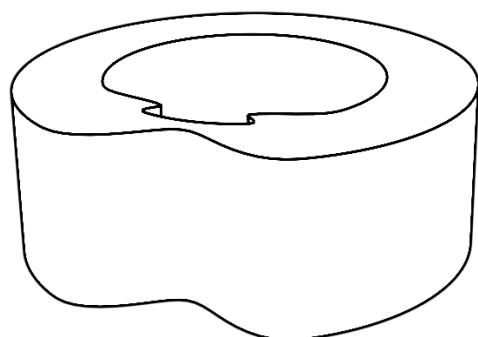
Skrótowy opis robota oznacza, że jego budowa wykorzystuje dwa człony obrotowe oraz jeden człon przesuwny. Zaletą takiego rozwiązania jest duży zasięg poziomy, ma to jednak swoje odzwierciedlenie w postaci małego zakresu pracy w pionie [7].

D. Robot w konfiguracji SCARA (OOP)

a)



b)

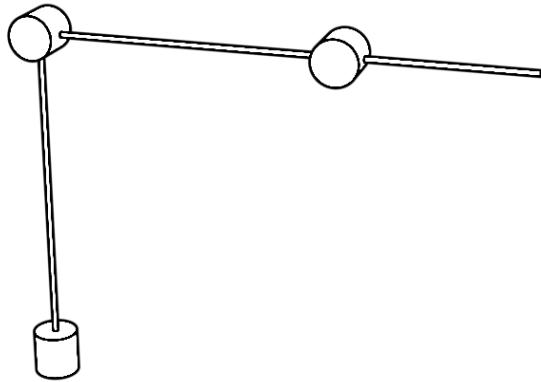


Rys. 3.5. Schemat robota SCARA a) oraz kształt jego przestrzeni roboczej b)

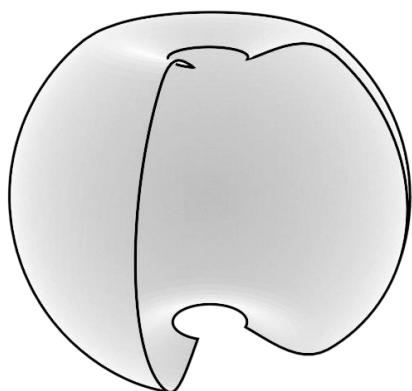
Skrótowy opis robota SCARA jest równoznaczny z opisem robota w konfiguracji sferycznej, jednak sposób połączenia oraz rozmieszczenia poszczególnych przegubów jest różny. Zastosowanie takiej konfiguracji pozwala na uzyskanie dużej przestrzeni roboczej, jednak poruszanie się po niej wymaga skomplikowanego sterowania ponieważ w każdym punkcie występują 2 możliwości ustawienia ramienia [7].

E. Robot w konfiguracji antropomorficznej (OOO)

a)



b)



Rys. 3.6. Schemat robota OOO a) oraz kształt jego przestrzeni roboczej b)

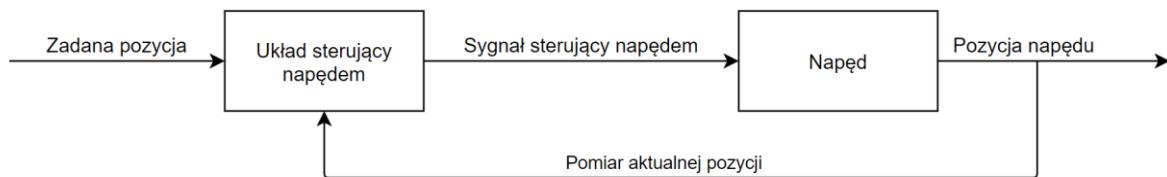
Skrótowy opis oznacza, że konstrukcja robota wykorzystuje 3 przeguby obrotowe. Wykorzystanie takiego układu pozwala na stosunkowo łatwe omijanie przeszkód, jednak wiąże się z tym konieczność wprowadzenia bardziej skomplikowanych układów sterowania od tych, których wymagają wcześniej wymienione konstrukcje. Jednym z czynników wpływających na trudność programowania jest fakt, że często istnieje więcej niż jeden sposób na osiągnięcie docelowej pozycji [7].

3.3. Napędy stosowane w robotach

Zarówno roboty przemysłowe jak i inne konstrukcje, w których duże znaczenie ma pozycjonowanie układu wykonawczego (np. efektora w robocie) muszą opierać się o napędy, które pozwalają na sterowanie położeniem elementu ruchomego w napędzie (np. położeniem wału silnika).

Współcześnie w zastosowaniach przemysłowych dominują napędy elektryczne. W przypadku kiedy istotna jest wysoka dynamika ruchu można spotkać również napędy hydrauliczne. Przykładem robota wykorzystującego takie rozwiązanie jest robot o nazwie Atlas wyprodukowany przez firmę Boston Dynamics [10]. Napęd z możliwością regulacji położenia nazywany jest serwonapędem.

Skupiając się na zagadnieniu serwonapędów elektrycznych można wyróżnić podstawową strukturę sterowania, pokazaną na rysunku 2.7.



Rys. 2.7. Schemat blokowy serwonapędu

Widoczne są dwa podstawowe układy:

- układ sterujący napędem (np. falownik), jest to układ odpowiedzialny za przetworzenie informacji o kącie, jaki ma zostać uzyskany na docelowy sygnał sterujący napędem uwzględniając przy tym wszelkie zakłócenia oraz aktualną pozycję napędu,
- napęd (np. silnik PMSM).

Dodatkowym układem który musi zostać wykorzystany jest element odczytujący bieżącą pozycję napędu. Często w tym celu wykorzystywany jest enkoder (np. inkrementalny).

Można wyróżnić trzy podstawowe grupy serwonapędów [4]:

- A. Serwonapędy prądu przemiennego. Są to układy wykorzystujące silniki prądu przemiennego. Wykorzystany silnik może być zarówno synchroniczny jak i asynchroniczny (indukcyjny). Wspólną cechą dla tych układów jest konieczność zastosowania falowników do sterowania silnikami.
- B. Serwonapędy prądu stałego. Układy o takiej budowie nie potrzebują do działania falownika, a ich sterowniki są prostsze w budowie od sterowników dla silników prądu przemiennego.
- C. Serwonapędy krokowe. Jest to specjalna grupa silników, których wirnik obraca się w ścisłe określony sposób. Obrót może zostać dokonany o minimalną wartość określoną jako krok silnika. Innymi słowy są to silniki o zdyskretyzowanym ruchu. Jest to jedyna grupa silników, która może pracować bez sprzężenia zwrotnego w postaci aktualnej pozycji wirnika zachowując możliwość sterowania położeniem. Taki rodzaj sterowania nie jest jednak odporny na zakłócenia. Grupa tych napędów ze względu na swoje właściwości stała się bardzo popularna w amatorskich zastosowaniach gdzie potrzebna jest regulacja położenia, jednak nie występują większe zakłócenia w pracy silników (przykładem są drukarki 3D).

3.4. Zastosowania robotów

W obecnych czasach wachlarz zastosowań robotów jest bardzo szeroki. Roboty przemysłowe, jak sama nazwa wskazuje, są nieodłącznym elementem przemysłu. W miejscach, gdzie wymagana jest automatyzacja danego procesu idealnym rozwiązaniem jest zastąpienie człowieka często narażonego na różne czynniki w otoczeniu podczas jego pracy. Dzięki takiemu zabiegowi można zyskać pewność pełnej powtarzalności procesu.

Przykładami zastosowań robotów w przemyśle są:

- Roboty spawalnicze (np. obsługujące linie składania samochodów). Spawanie jest procesem szkodliwym dla człowieka. Zastępując czynnik ludzki robotem uzyskamy pewność wykonania za każdym razem jednakowego łączenia danych elementów oraz zwiększymy prędkość produkcji poprzez niwelację przerw jakie są niezbędne dla człowieka do poprawnego funkcjonowania.
- Roboty pakujące. Jest to bardzo powszechnie zastosowanie robotów, które pozwala na zwiększenie prędkości pakowania oraz osiągnięcie wysokiej precyzji w pozycjonowaniu wraz z niezmienną powtarzalnością.

- Roboty do montażu płaskiego. Najczęściej do tego celu wykorzystuje się roboty typu SCARA. Kiedy zachodzi konieczność precyzyjnego umieszczenia danego elementu w obudowie lub element jest bardzo mały wówczas wykorzystanie do tego celu robota wielokrotnie zwiększa prędkość działań. Przykładem jest zastosowanie robotów działających na zasadzie „pick and place” do układania elementów montażowych na płytach PCB. Konieczna jest wówczas precyza w pozycjonowaniu oraz prędkość dla zapewnienia optymalnej prędkości produkcji.
- Roboty malarskie. Podczas etapu produkcji kiedy konieczne jest równomierne naniesienie farby na różne elementy firma musi posiadać pracownika mającego duże doświadczenie oraz umiejętności. Jednak kiedy mamy do czynienia z produkcją seryjną (jak np. w fabrykach samochodów) wówczas automatyzacja procesu malowania bardzo usprawnia ten proces.
- Roboty inspekcyjne. Na końcowym etapie produkcji kiedy konieczna jest kontrola jakości wprowadzenie automatyzacji z wykorzystaniem robotów pozwala na precyzyjne oraz powtarzalne sprawdzenie mierzonego parametru w konkretnym punkcie produkowanego elementu. Dzięki wykorzystaniu robotów proces ten jest szybszy przez co większa ilość komponentów może być jemu poddana bez zmiany tąpa produkcji.

Podsumowując, można stwierdzić, że wszędzie tam, gdzie wymagana jest precyza ruchu, wysoka powtarzalność oraz duża wydajność procesu wprowadzenie automatyzacji w postaci zamiany czynnika ludzkiego na robota jest częstą praktyką. Bardzo duża część procesów produkcyjnych jest prowadzona w oparciu o wykorzystanie robotów, a wyżej wymieniona została jedynie część z nich. Poza przemysłem istnienie wiele różnych dziedzin, w których nieodłączną rolę pełnią roboty, przykładem są roboty chirurgiczne.

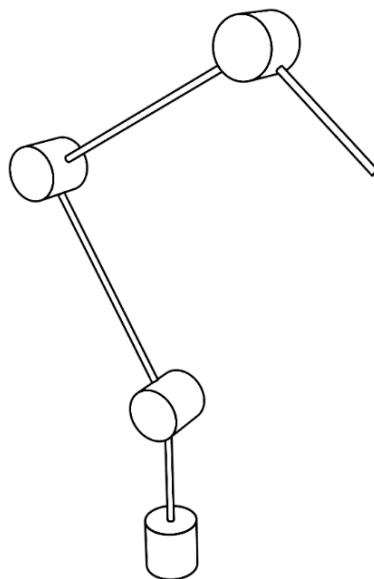
4. Projekt robota ramieniowego

4.1. Założenia projektowe

Przed przystąpieniem do projektowania oraz docelowej budowy robota, konieczne jest przyjęcie pewnych założeń, które będą narzucały dalsze decyzje. Ze względu na charakter pracy, której celem jest budowa eksperymentalnego ramienia nie przeznaczonego do produkcji masowej ważnym czynnikiem, który będzie decydował o przyjętych rozwiązańach jest cena zakupu lub wykonania poszczególnych elementów. Kolejnym ważnym założeniem jest określenie rozmiaru ramienia tak, aby zachować możliwość przenoszenia robota. Wiąże się z tym również waga całego urządzenia, która musi pozwolić na swobodny transport robota. Robot wraz z dedykowanym oprogramowaniem powinien zapewnić użytkownikowi możliwość swobodnego sterowania ramieniem oraz wprowadzić możliwość podstawowej automatyzacji procesów, w których będzie brał udział. Mając na uwadze powyższe założenia projektowe, kolejnym krokiem jest opracowanie koncepcji budowanego robota.

4.2. Koncepcja robota

Początkowy etap budowy ramienia jest związany z podjęciem wielu decyzji, które określają ogólny zarys ramienia. Pierwszą decyzją która w największym stopniu rzutowała na projektowane ramie jest wybór ilości stopni swobody. W tym przypadku wybór padł na 4 stopnie swobody. Decyzja ta jest uwarunkowana przez kilka czynników. Pierwszym z nich jest założenie, że ramię powinno dawać możliwość kontroli położenia efektora wzdłuż osi układu współrzędnych (przyjętym układem jest kartezjański układ XYZ). Minimalna liczba osi swobody, która na to pozwoli przy wykorzystaniu wyłącznie obrotowych przegubów to 3. Dodając do takiego układu dodatkowy stopień swobody zwiększymy jego możliwości ruchowe, przez co możliwe jest poruszanie się ramienia zgodnie z przyjętymi założeniami oraz dodatkowo możliwa jest kontrola nachylenia efektora względem płaszczyzny podłożu. Innym czynnikiem jest ograniczony dostęp do napędów ze względu na koszty jakie ze sobą niosą.

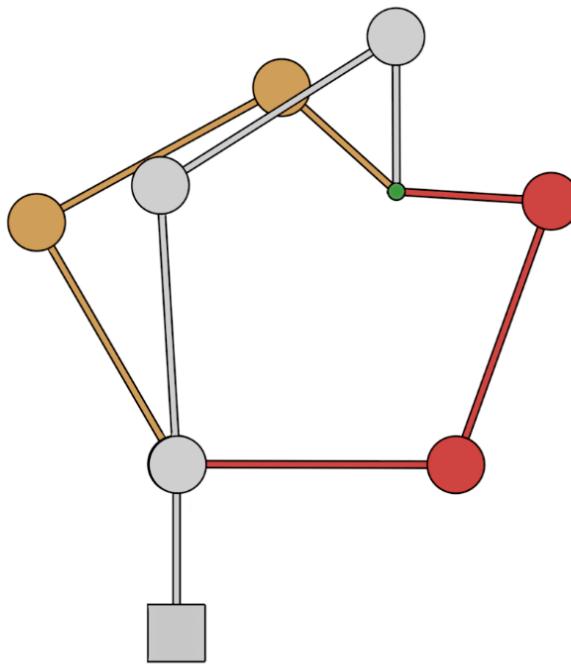


Rys. 4.1. Schematyczne przedstawienie zamysłu budowanego ramienia robota

Na rysunku 4 przedstawiono sposób, w jaki rozmieszczone zostaną przeguby w ramieniu. Jak wynika z rysunku, schemat ten odpowiada robotowi o kinematyce RRR powiększonemu o dodatkowy przegub, który również można opisać literą „R”. Konstrukcja taka cechuje się licznymi zaletami, takimi jak:

- Zdolność do poruszania się zgodnie z osiami układu kartezjańskiego XYZ,
- Łatwość w omijaniu przeszkód,
- Stosunkowo duży zasięg,
- Niewielka minimalna wymagana powierzchnia do usytuowania robota.

Jak widać wymienione wyżej cechy pokazują, że taka konstrukcja umożliwia swobodne wykorzystanie ramienia w różnych celach. Niestety posiada ona też wady. Jedną z nich jest fakt, że dla większości punktów w przestrzeni w których ma znaleźć się efektor nie jest jednoznacznie określone w jakim ustawieniu musi znaleźć się ramię. Ogólnie mówiąc kinematyka ramienia nie jest prosta. Niejednoznaczność kinematyki ramienia pokazano na rysunku 4.2.



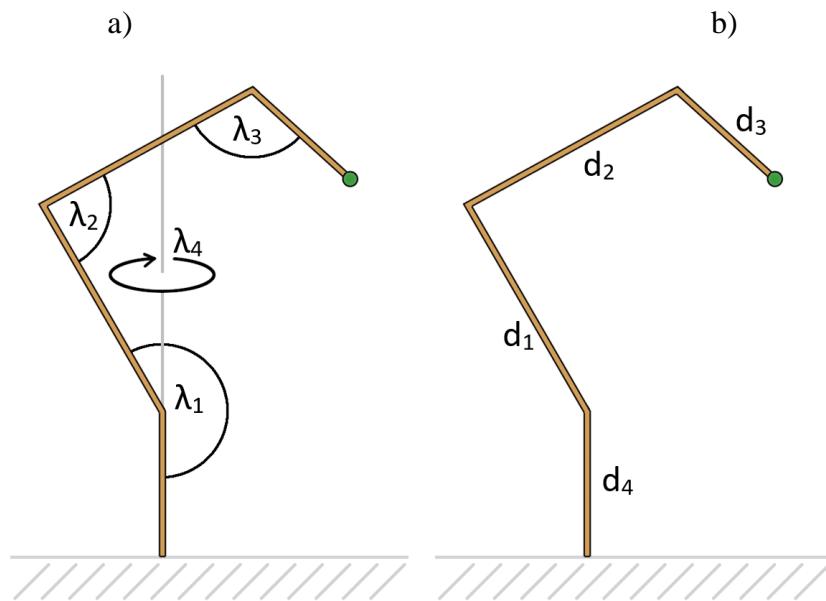
Rys. 4.2. Przedstawienie niejednoznaczności ustawienia ramienia dla jednakowej docelowej pozycji efektora (bez uwzględnienia jego rotacji)

Jak łatwo wywnioskować z powyższego rysunku, istnieje nieskończoność możliwych ustawień ramienia, która pozwala na osiągnięcie pożdanego punktu w przestrzeni roboczej. Część z tych pozycji jest niekorzystna dla niektórych zadań ramienia. Dlatego jednym z podstawowych założeń dalszych rozważań będzie przyjęcie, że ramię nie może znaleźć się poniżej efektora. Innymi słowy, pozycja zaznaczona kolorem czerwonym na rysunku 3.2 będzie niedozwolona.

4.3. Określenie kinematyki ramienia

Chcąc jednoznacznie wiązać położenie efektora wraz z ustawieniem poszczególnych sekcji ramienia przyjęto uproszczenie, które pozwala operatorowi na odgórne ustalenie kąta nachylenia efektora względem płaszczyzny poziomej. Dzięki takiemu podejściu każdemu punktowi, w którym znajdzie się efektor odpowiadał będzie wyłącznie jeden zestaw kątów określających wzajemne ustawienie względem siebie dwóch sąsiadujących sekcji ramienia. Wyjątkiem jest sytuacja, kiedy efektor ma znaleźć się w osi pionowej robota. Wówczas również istnieje nieskończoność wielu układów ramienia pozwalających osiągnąć daną pozycję. W celu ograniczenia takiego zachowania, również ruch przez oś pionową robota jest niedozwolony.

Do określenia kątów, które mają wystąpić pomiędzy poszczególnymi parami sekcji ramienia konieczna jest znajomość parametrów ramienia m.in. długości ramion. Kąty te przyjmą nazewnictwo według wzorca: λ_i , gdzie i oznacza numer przegubu. Długości ramion zostaną opisane według wzorca: d_i , gdzie i również oznacza numer przegubu. Powyższa nomenklatura będzie wykorzystana w dalszych obliczeniach.

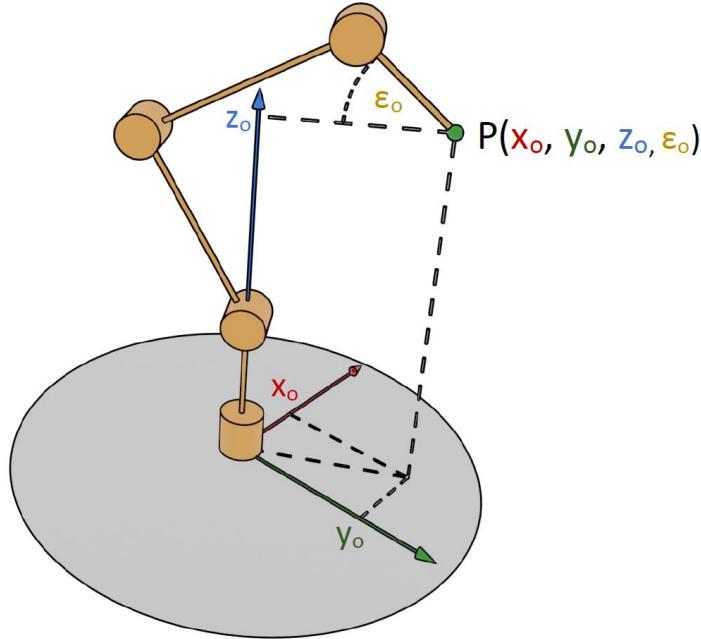


Rys. 4.3. Kąty potrzebne do sterowania ramieniem a) oraz długości niezbędne do obliczeń b)

Analizując zagadnienie kinematyki ramienia możemy wyróżnić jej dwa rodzaje: kinematykę prostą oraz odwrotną. Kinematyka prosta w omawianym przypadku zajmuje się odpowiedzią na pytanie: jaka jest pozycja efektora kiedy znamy wartości kątów λ_i . Kinematyka odwrotna natomiast realizuje trudniejsze zadanie, którego celem jest określenie kątów λ_i na podstawie współrzędnych położenia efektora.

A. Kinematyka prosta

Rozważając zagadnienie kinematyki jako funkcję, która przyjmuje jako argumenty poszczególne kąty λ_i , otrzymujemy w wyniku jej wykonania odpowiednie koordynaty położenia.



Rys. 4.4. Rysunek pomocniczy dla analizy kinematyki prostej

Uogólniony zapis działania kinematyki prostej przedstawiają poniższe równania:

$$x_o = f_x(\lambda, d) \quad [1.0]$$

$$y_o = f_y(\lambda, d) \quad [1.1]$$

$$z_o = f_z(\lambda, d) \quad [1.2]$$

$$\varepsilon_o = f_\varepsilon(\lambda, d) \quad [1.3]$$

Głębsza analiza pozwala na wyznaczenie dokładnych wzorów dla każdej z współrzędnych. W tym celu należy posłużyć się właściwościami trygonometrycznymi. Poniżej znajdują się końcowe równania będące odpowiedzią na zagadnienie kinematyki prostej

$$x_o = \cos(\lambda_4) \cdot \left[d_1 \cdot \cos\left(\lambda_1 - \frac{\pi}{2}\right) + d_2 \cdot \cos\left(\lambda_1 + \lambda_2 + \frac{\pi}{2}\right) + d_3 \cdot \sin(\lambda_1 + \lambda_2 + \lambda_3) \right] \quad [2.0]$$

$$y_o = \sin(\lambda_4) \cdot \left[d_1 \cdot \cos\left(\lambda_1 - \frac{\pi}{2}\right) + d_2 \cdot \cos\left(\lambda_1 + \lambda_2 + \frac{\pi}{2}\right) + d_3 \cdot \sin(\lambda_1 + \lambda_2 + \lambda_3) \right] \quad [2.1]$$

$$z_o = d_1 \cdot \sin\left(\lambda_1 - \frac{\pi}{2}\right) + d_2 \cdot \sin\left(\lambda_1 + \lambda_2 + \frac{\pi}{2}\right) + d_3 \cdot \cos(\lambda_1 + \lambda_2 + \lambda_3) + d_4 \quad [2.2]$$

$$\varepsilon_o = \frac{\pi}{2} - \lambda_1 - \lambda_2 - \lambda_3 \quad [2.3]$$

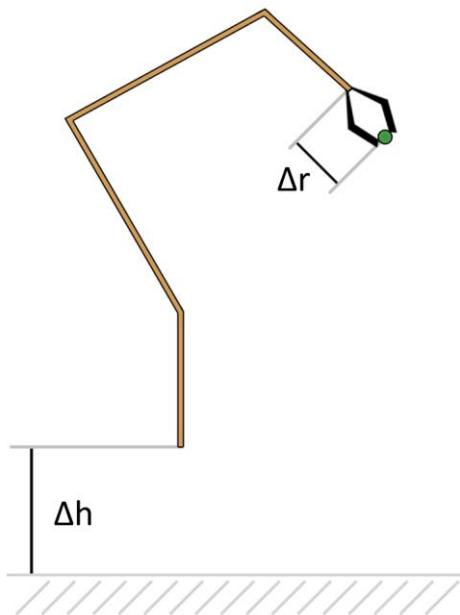
Powyższe równania pozwalają na obliczenie współrzędnych punktu przy znajomości parametrów robota oraz jego bieżącego ustawienia. W sytuacji, kiedy robot zostanie umieszczony na podniesieniu, wówczas chcąc zachować układ współrzędnych mający punkt (0,0,0) w płaszczyźnie podłoża konieczna jest modyfikacja parametru d_4 . Analogiczne przekształcenie jest wymagane wówczas, kiedy do efektora dołączony zostanie dodatkowy element (np. chwytak). Wówczas po to, żeby algorytm kinematyki prostej zwracał współrzędne punktu będącego nowym punktem wodzącym ramienia należy skorygować wartość d_3 o określzoną odległość.

$$d_{4\ nowe} = d_4 + \Delta h \quad [3.1]$$

$$d_{3\ nowe} = d_3 + \Delta r \quad [3.2]$$

Gdzie:

- $d_{3\ nowe}$ oraz $d_{4\ nowe}$ to nowe wartości parametrów d_3 oraz d_4 które należy wykorzystać do obliczeń,
- Δh to wysokość na jaką został podniesiony robot,
- Δr to odległość o jaką przesunięty został punkt wodzący ramienia wzdłuż osi wyznaczonej przez odcinek d_3 .



Rys. 4.5. Graficzne przedstawienie przesunięć w celu korekcji obliczeń algorytmu kinematyki prostej

B. Kinematyka odwrotna

Rolą algorytmu kinematyki odwrotnej jest zwrócenie wartości poszczególnych kątów λ_i na podstawie znajomości parametrów robota oraz pozycji punktu wodzącego ramienia. W celu poprawnego funkcjonowania algorytm potrzebuje 4 zmiennych przechowujących informacje o współrzędnych punktu wodzącego ramienia oraz kąta nachylenia efektora względem płaszczyzny podłoża. Uproszczony model matematyczny może zostać zapisany jako:

$$P = (x, y, z, \varepsilon) \quad [4.0]$$

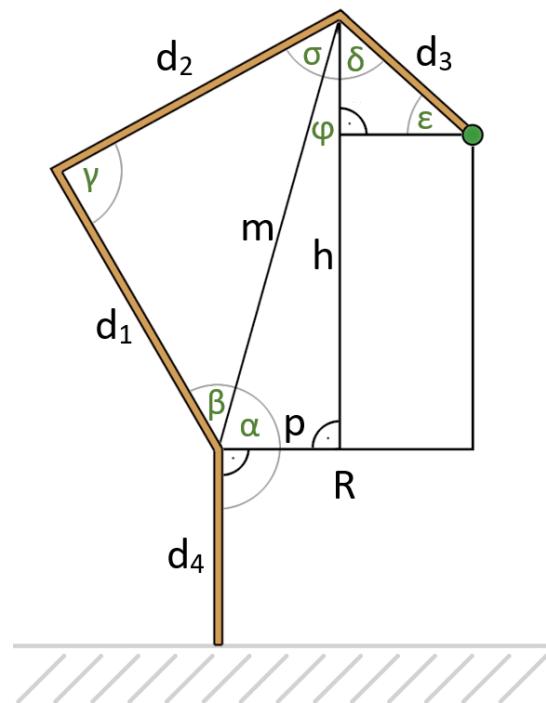
$$\lambda_1 = f_1(P, d) \quad [4.1]$$

$$\lambda_2 = f_y(P, d) \quad [4.2]$$

$$\lambda_3 = f_z(P, d) \quad [4.3]$$

$$\lambda_4 = f_\varepsilon(P, d) \quad [4.4]$$

Rysunek 4.6 przedstawia najważniejsze kąty oraz długości poszczególnych odcinków pomocniczych wraz z długościami ramion. Dzięki takiemu podejściu do kinematyki odwrotnej możliwe będzie określenie jej algorytmu bez pomocy bardziej skomplikowanych metod.



Rys. 4.6. Zestawienie ważnych kątów oraz długości poszczególnych elementów ramienia

Rozważając powyższe zależności, oraz opierając je na własnościach trygonometrycznych można przeprowadzić następującą analizę:

- I. Odległość rzutu punktu wodzącego na płaszczyznę poziomą od jej środka można opisać posługując się twierdzeniem Pitagorasa oraz współrzędnymi x i y punktu wodzącego:

$$R = \sqrt{x_o^2 + y_o^2} \quad [5.0]$$

- II. Wiedząc, że wartość kąta nachylenia efektora względem płaszczyzny poziomej jest wprowadzana przez operatora, możemy określić zależność na zmienną p :

$$p = R - d_3 \cdot \cos(\varepsilon) \quad [5.1]$$

- III. Długość h można wyznaczyć z następującej zależności :

$$h = z_o - d_3 + d_3 \cdot \sin(\varepsilon) \quad [5.2]$$

- IV. Długość m można wyznaczyć z twierdzenia pitagorasa:

$$m = \sqrt{p^2 + h^2} \quad [5.3]$$

- V. Kąt α można wyznaczyć za pomocą funkcji odwrotnego tangensa:

$$\alpha = \tan^{-1} \left(\frac{h}{p} \right) \quad [5.4]$$

- VI. Kąty β oraz γ można wyznaczyć z twierdzenia cosinusów:

$$\beta = \cos^{-1} \left(\frac{d_1^2 + m^2 - d_2^2}{2 \cdot m \cdot d_1} \right) \quad [5.5]$$

$$\gamma = \cos^{-1} \left(\frac{d_1^2 + d_2^2 - m^2}{2 \cdot d_1 \cdot d_2} \right) \quad [5.6]$$

- VII. Znając powyższe kąty możemy obliczyć kąty σ , φ oraz δ :

$$\sigma = \pi - \beta - \gamma \quad [5.7]$$

$$\varphi = \frac{\pi}{2} - \alpha \quad [5.8]$$

$$\delta = \frac{\pi}{2} - \varepsilon \quad [5.9]$$

- VIII. Znając wszystkie potrzebne długości oraz kąty należy wyznaczyć kąty λ_i :

$$\lambda_1 = \alpha + \beta + \frac{\pi}{2} \quad [5.10]$$

$$\lambda_2 = \gamma \quad [5.11]$$

$$\lambda_3 = \sigma + \varphi + \delta \quad [5.12]$$

$$\lambda_4 = \tan^{-1} \left(\frac{y_o}{x_o} \right) \quad [5.13]$$

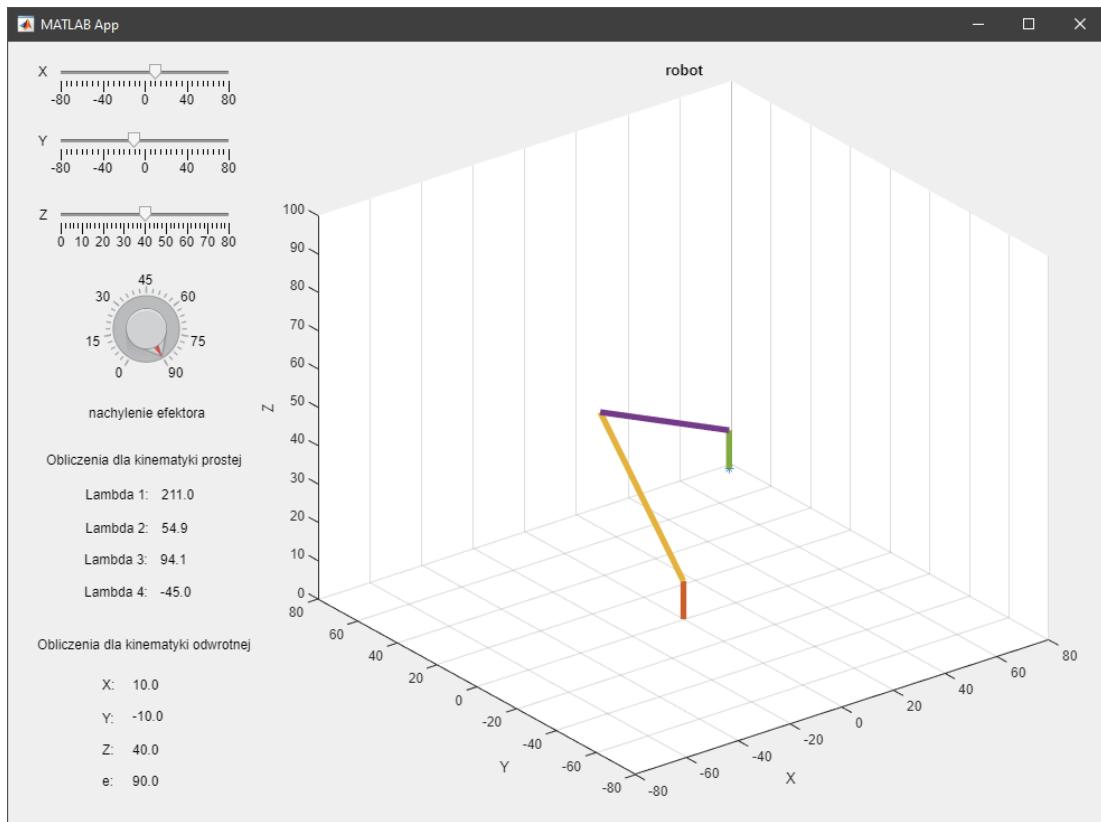
Powyższy algorytm obliczeń pozwala na uzyskanie konkretnych wartości kątów dla zadanej pozycji efektora. Równania [5.4], [5.5], [5.6] oraz [5.13] zawierają w swojej składni iloraz dwóch wartości. Niesie to ze sobą niebezpieczeństwo niepowodzenia obliczeń numerycznych w przypadku wystąpienia wartości zerowej w mianowniku. O ile w przypadkach [5.5] oraz [5.6] wartość taka nie może wystąpić (ponieważ wartości długości poszczególnych sekcji ramienia są niezerowe) tak, dla pozostałych równań są to realne przeszkody dla programu, które należy odpowiednio wykluczyć.

Chcąc poprawić działanie algorytmu w tych miejscach konieczne będzie uwzględnienie faktu, że dla $y_o > 0$ oraz dla $x_o = 0$, wartość kąta λ_4 ma wynosić $\frac{\pi}{2}$, natomiast dla $y_o < 0$, wartość kąta λ_4 ma być równa $-\frac{\pi}{2}$. Dzięki temu uzyskamy wartość liczbową dla przypadku [5.13].

W ostatnim przypadku, w którym z równania wynika, że w mianowniku znajduje się zmienna p , dla jej zerowej wartości ramię znajdowałoby się w pozycji, którą można nazwać kolizyjną. Oznacza to, że dwie sekcje ramienia robota wchodziłyby ze sobą w kolizję. Można więc uznać, że jest to zabezpieczenie przeciw mechanicznemu uszkodzeniu ramienia.

Dzięki połączeniu kinematyki prostej oraz odwrotnej możliwe jest dowolne przechodzenie pomiędzy sterowaniem przy pomocy kątów λ_i lub za pomocą współrzędnych punktu wodzącego ramienia.

W celu sprawdzenia poprawności obliczeń została zaprojektowana aplikacja w środowisku MATLAB. Jest to aplikacja, która dzięki graficznej prezentacji obliczeń pozwala na zobrazowanie wyników algorytmu kinematyki odwrotnej. Dodatkowo, na podstawie wyników tego algorytmu oraz wykorzystaniu ich jako dane wejściowe dla algorytmu kinematyki prostej, uzyskujemy dane o współrzędnych punktu wodzącego ramienia. Użytkownik ma do dyspozycji 4 elementy pozwalające na regulację współrzędnych efektora. Przez porównanie wprowadzonych danych przez użytkownika oraz danych zwróconych przez algorytm kinematyki prostej możliwe jest określenie poprawności przeprowadzonych obliczeń. GUI aplikacji sprawdzającej poprawność obliczeń pokazano na rysunku 4.7.



Rys. 4.7. GUI aplikacji sprawdzającej poprawność obliczeń algorytmów kinematyki

Jak można zauważyć, graficzny model ramienia dla wprowadzonych współrzędnych punktu wodzącego $P(10, -10, 40, 90)$ przyjmuje pozycję, która dobrze oddaje jego pożądane położenie. Kolejność wykonywania działań przez program jest następująca:

- I. Odczytanie współrzędnych wprowadzonych przez użytkownika
- II. Wykonanie obliczeń kinematyki odwrotnej
- III. Narysowanie symbolicznego modelu ramienia, za pomocą wektorów przy uwzględnieniu wyłącznie długości poszczególnych części ramienia d_i oraz kątów obliczonych w punkcie II
- IV. Realizacja obliczeń związanych z kinematyką prostą, dla której danymi wejściowymi są dane z punktu II
- V. Wyświetlenie rezultatów obliczeń

Porównując rezultaty obliczeń z współrzędnymi wprowadzonymi przez użytkownika można dojść do wniosku, że wartości te pokrywają się, co potwierdza poprawność wykonanych obliczeń.

4.4. Sposób doboru układu napędowego

Chcąc wprawić ramię w ruch konieczne jest dobranie odpowiedniego zespołu napędowego. W realizowanym robocie wybór padł na silniki krokowe. Ze względu na charakter pracy, w którym ważne jest dokładne pozycjonowanie efektora napędy te są wyposażone w pętle sprzężenia zwrotnego. Dobre zostały 3 rodzaje napędów, wszystkie spełniające wyżej

wymienione założenia. Są to silniki krokowe z grupy NEMA34 działające w pętli zamkniętej. Typ napędów podyktowany został również ich planowym przyszłym wykorzystaniem. Najmniejszy silnik charakteryzuje się następującymi parametrami [11]:

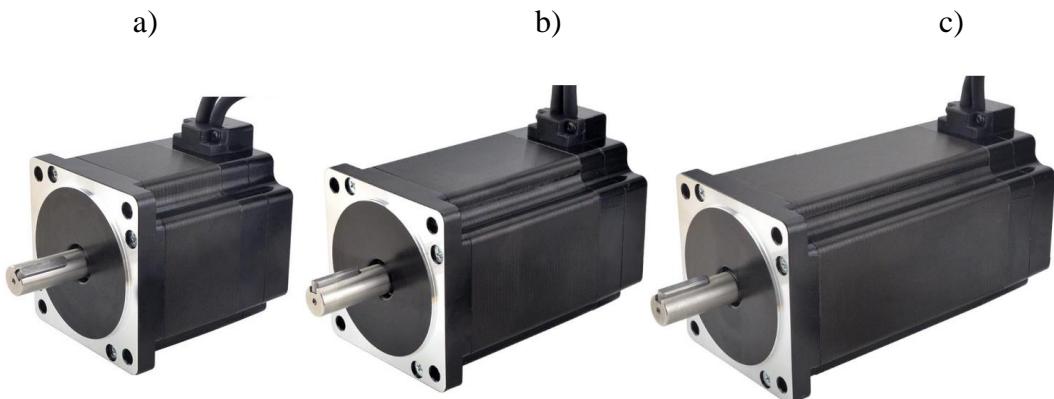
- Moment trzymający $M_t = 4.8\text{Nm}$
- Prąd znamionowy $I_n = 5\text{A}$
- Liczba kroków na obrót $n_k = 200$
- Ilość impulsów z enkodera na przypadająca na obrót $n_e = 1000$
- Waga silnika $m = 2.5 \text{ kg}$

Drugi typ silnika, który zostanie wykorzystany w dwóch miejscach, również spełniający zadane wymagania, jednak charakteryzujący się większym momentem trzymającym (ze względu na planowane zastosowanie do napędu sekcji ramienia generującej większy moment) jest opisany następującymi parametrami [12]:

- Moment trzymający $M_t = 9\text{Nm}$
- Prąd znamionowy $I_n = 6\text{A}$
- Liczba kroków na obrót $n_k = 200$
- Ilość impulsów z enkodera na przypadająca na obrót $n_e = 1000$
- Waga silnika $m = 4.2 \text{ kg}$

Trzecim typem silnika jest model o największym momencie trzymającym. Został on dobrany ze względu na planowe wykorzystanie go do napędu najbardziej obciążonego momentem przegubu. Parametry tego silnika prezentują się następująco [13]:

- Moment trzymający $M_t = 12\text{Nm}$
- Prąd znamionowy $I_n = 6\text{A}$
- Liczba kroków na obrót $n_k = 200$
- Ilość impulsów z enkodera na przypadająca na obrót $n_e = 1000$
- Waga silnika $m = 5.5 \text{ kg}$



Rys. 4.8. Zestawienie dobranych silników a) silnik 4.5Nm, b) silnik 9Nm, c) silnik 12Nm [14]

W celu odpowiedniego wysterowania napędów potrzebny jest specjalny układ elektroniczny noszący nazwę karty sterowniczej silnika krokowego. Ze względu na zapotrzebowanie na prąd wykorzystanych silników konieczne jest dobranie odpowiednich kart sterujących. Do realizacji układu wybrano sterownik firmy HLTNC o nazwie HBS860H pokazany na rysunku 4.9.



Rys. 4.9. Dobra karta sterująca silnikami [15]

Karta ta charakteryzuje się następującymi parametrami [15]:

- Maksymalny prąd fazowy: $I_{max} = 8A$,
- Napięcie zasilania karty $U_{DC} = 24-100 VDC$, $U_{AC} = 20-80VAC$,
- Maksymalna rozdzielczość 51200 kroków na obrót,
- Możliwość zdefiniowania typu pracy dla konkretnego modelu silnika,
- Obsługa enkodera inkrementalnego.

Ostatnim elementem układu napędowego jest układ zasilania. Ze względu na możliwość zasilania kart sterowniczych napięciem stałym oraz przemiennym możliwe jest wykorzystanie w tym celu transformatora obniżającego napięcie na odpowiednio dopasowane do potrzeb karty sterowniczej. Innym sposobem zasilenia układu jest wykorzystanie zasilaczy z napięciem wyjściowym na wymaganym poziomie i w realizowanej pracy ten sposób został wykorzystany.

Dobór mocy zasilacza należy rozpocząć od określenia napięcia, jakie będzie zasilało karty sterownicze. Wartość napięcia została przyjęta jako $U_{DC} = 60V$. Przyjmując 10% zapasu mocy dla każdego silnika oraz zakładając, że każdy napęd ma działać indywidualnie, można określić zależność na zapotrzebowanie na moc jako [16]:

$$P = U_{DC} \cdot I_n \cdot 1.1 \quad [6.0]$$

Gdzie:

- U_{DC} to napięcie wyjściowe zasilacza

- I_n to maksymalny prąd fazowy silnika (przyjęty został największy prąd pośród wybranych jednostek w celu ujednolicenia modułów)

$$P = 60V \cdot 6A \cdot 1.1 = 396W \quad [6.1]$$

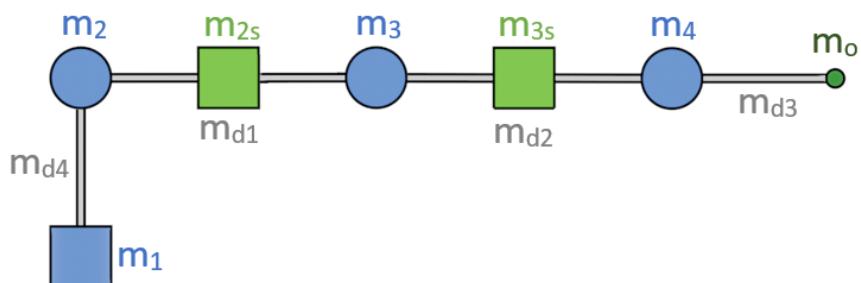
Jak wynika z powyższych obliczeń w celu zapewnienia poprawnego działania układu wraz z 10% marginesem bezpieczeństwa należy zastosować zasilacz o mocy wyjściowej około 400W. Wybranym modelem zasilacza jest zasilacz impulsowy firmy WEHO sygnowany kodem S-400-60. Jest to jednostka o napięciu wyjściowych 60VDC oraz mocy 400W, co spełnia powyższe założenia. Wybrany zasilacz pokazano na rysunku 4.10.



Rys. 4.10. Dobrany zasilacz impulsowy 400W, 60V [17]

4.4.1. Obliczenia obciążen dla napędów

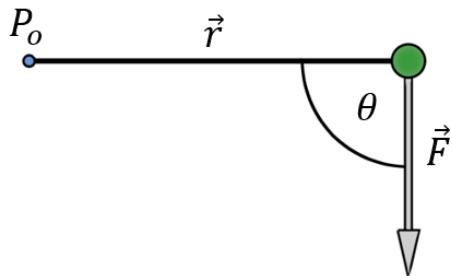
Dla dobranych silników konieczne jest również dobranie odpowiednich przekładni. W tym celu należy obliczyć momenty występujące na poszczególnych stopniach ramienia. Do tego konieczne jest wstępne rozmieszczenie elementów mechanicznych oraz silników w ramieniu. Dla uproszczenia obliczeń (przy wprowadzeniu niewielkiego błędu) można przyjąć, że środki ciężkości poszczególnych sekcji ramion leżą w połowie ich długości. Dodatkowo można przyjąć, że masy znajdujące się w poszczególnych przegubach (np. masy łożysk, śrub itp.) są masami skupionymi oraz znajdują się w odpowiadających ich położeniu osiach obrotu danej sekcji ramienia. Silniki napędzające odpowiadające im osie znajdują się w połowie długości poprzedniej sekcji ramienia. Zobrazowane zostało to poniżej na rysunku 4.11. Dodatkowe obciążenie, które znajduje się na chwytaku będzie rozpatrywane jako masa skupiona, znajdująca się w punkcie wodzącym ramienia.



Rys. 4.11. Schematyczne ułożenie poszczególnych elementów w ramieniu

Powyższy schemat ramienia pokazuje je w pozycji maksymalnie pochylonej w bok. W takim ułożeniu w ramionach występują największe możliwe obciążenia, inaczej mówiąc, w tej pozycji momenty obciążenia w poszczególnych stawach ramienia są największe.

Chcąc jednak rozpatrywać momenty obciążenia poszczególnych osi dla każdego możliwego ustawienia ramienia musimy взять pod uwagę nachylenie poszczególnych sekcji ramienia względem podłożu. Dla zrozumienia późniejszej analizy konieczne jest wprowadzenie opisu matematycznego momentu siły.



Rys. 4.12. Rysunek pomocniczy do opisu momentu siły

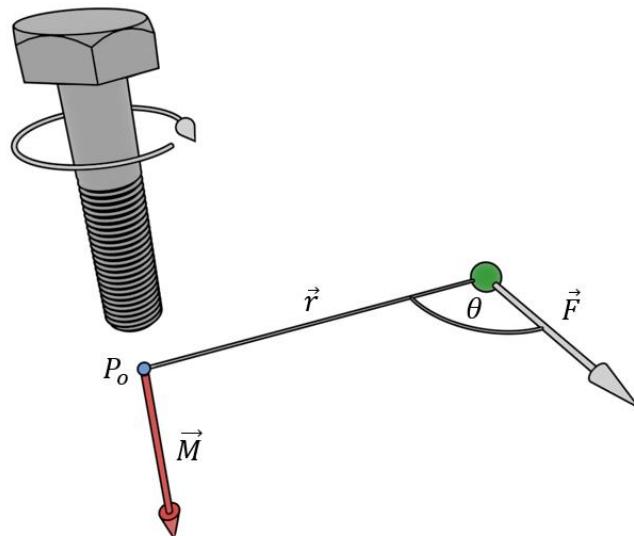
Moment siły jest to wynik iloczynu wektorowego promienia o punkcie zaczepienia P_o , kończącego się w punkcie przyłożenia siły, oraz siły F zgodnie z równaniem [18]:

$$\vec{M} = \vec{r} \times \vec{F} \quad [7.0]$$

Wartość wektora momentu można opisać równaniem:

$$M = r \cdot F \cdot \sin \theta \quad [7.1]$$

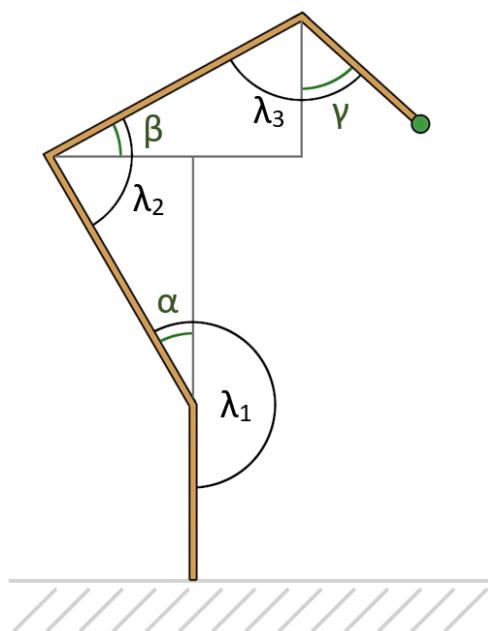
Jak wynika z powyższego wzoru, wartość momentu jest największa, kiedy wektor siły jest prostopadły do wektora wodzącego. Zwrot wektora momentu można wyznaczyć korzystając z reguły śruby prawoskrętnej. Reguła ta mówi, że jeśli w punkcie w którym wyznaczamy moment, przyłożymy śrubę (z klasycznym, prawoskrętnym gwintem), a kierunek obrotów śruby wyznaczamy korzystając ze zwrotu wektora siły \vec{F} , wówczas jeśli śruba będzie się wkręcała się w płaszczyznę wyznaczoną przez wektory \vec{r} oraz \vec{F} , wtedy zwrot wektora momentu przyjmuje się jako prostopadły do płaszczyzny oraz skierowany za płaszczyznę. W przeciwnym przypadku, kiedy śruba będzie wykręcała się z tej płaszczyzny, zwrot wektora przyjmuje się jako wychodzący z płaszczyzny. Pokazano to na rysunku 4.13.



Rys. 4.13. Zwrot wektora momentu na podstawie reguły śruby prawoskrętnej

Rozpatrywanie momentów dla poszczególnych przegubów ramienia zostanie podzielone na trzy główne sekcje. Pierwsza sekcja obejmuje obciążenia występujące w przegubie oznaczonym masą m_4 (rysunek 3.11). Obciążenia te będą przenoszone za pomocą przekładni (np. pasowej) na silnik oznaczony jako M_3 . Druga sekcja będzie obejmowała obciążenia w przegubie oznaczonym masą m_3 . Silnik który będzie odpowiedzialny za napędzanie tej części ramienia jest oznaczony jako M_2 . Ostatnia sekcja będzie obejmowała całą ruchomą masę ramienia, za jej ruch będzie odpowiedzialny silnik, który nie będzie znajdował się na samym ramieniu lecz na platformie poza nim.

Zanim jednak możliwe będzie przeprowadzenie obliczeń, przydatne będzie wyznaczenie kątów pomocniczych, które ułatwią dalsze obliczenia. Kąty te pokazano na rysunku 4.14.



Rys. 4.14. Kąty pomocnicze dla obliczeń momentów

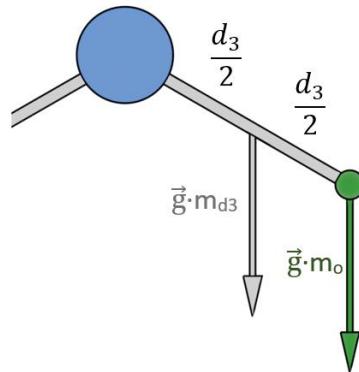
Wartości poszczególnych kątów można wyznaczyć znając wartości kątów λ_i oraz opierając się na poniższych zależnościach:

$$\alpha = \lambda_1 - \pi \quad [8.0]$$

$$\beta = \lambda_1 + \lambda_2 - \frac{3\pi}{2} \quad [8.1]$$

$$\gamma = \lambda_1 + \lambda_2 + \lambda_3 \quad [8.2]$$

I. Obliczenia dla ostatniej sekcji ramienia:

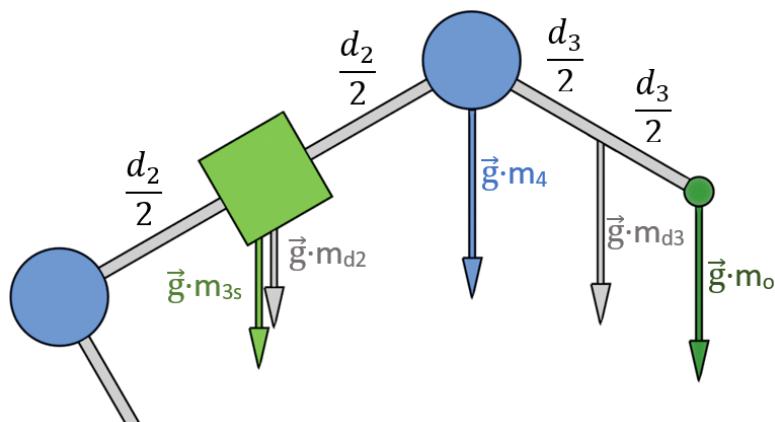


Rys. 4.15. Rysunek pomocniczy dla obliczeń związanych z pierwszą sekcją

Wartość momentu liczonego w punkcie obrotu sekcji d_3 wynosi:

$$M_3 = g \cdot \left[d_3 \cdot \sin(\gamma) \cdot m_o + \frac{d_3}{2} \cdot \sin(\gamma) \cdot m_{d3} \right] \quad [9.0]$$

II. Obliczenia dla środkowej sekcji ramienia:



Rys. 4.16. Rysunek pomocniczy dla obliczeń związanych z drugą sekcją

Wartość momentu w punkcie obrotu sekcji d_2 zostanie podzielona na cztery części, a następnie zsumowana w celu poprawienia przejrzystości obliczeń. Wynosi ona:

$$M_2 = M_{21} + M_{22} + M_{23} + M_{24} \quad [10.0]$$

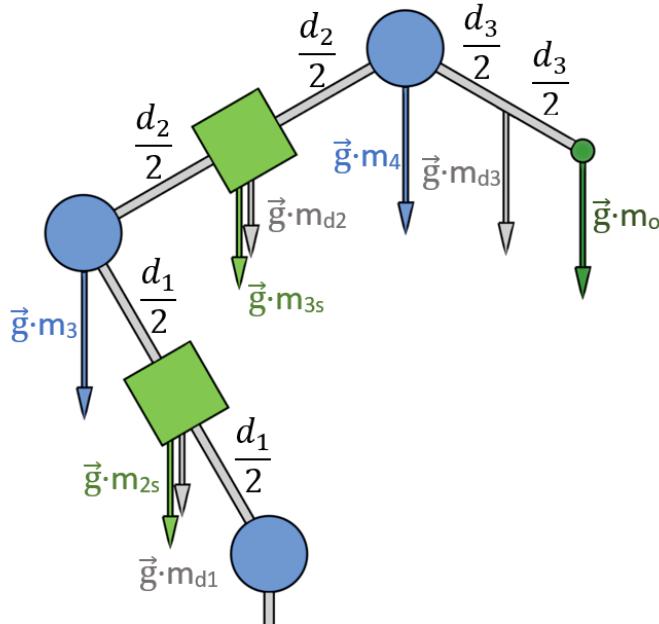
$$M_{21} = g \cdot m_o \cdot (d_3 \cdot \sin(\gamma) + d_2 \cdot \cos(\beta)) \quad [10.1]$$

$$M_{22} = g \cdot m_{d3} \cdot \left(\frac{d_3}{2} \cdot \sin(\gamma) + d_2 \cdot \cos(\beta) \right) \quad [10.2]$$

$$M_{23} = g \cdot m_4 \cdot d_2 \cdot \cos(\beta) \quad [10.3]$$

$$M_{24} = g \cdot (m_{d2} + m_{3s}) \cdot \frac{d_2}{2} \cdot \cos(\beta) \quad [10.4]$$

III. Obliczenia dla pierwszej sekcji ramienia:



Rys. 4.17. Rysunek pomocniczy dla obliczeń związanych z pierwszą sekcją

Wartość momentu w punkcie obrotu sekcji d_1 zostanie podzielona na 6 sekcji, które po zsumowaniu dadzą moment całkowity, a obliczenia będą bardziej przejrzyste. Moment ten wynosi:

$$M_3 = M_{31} + M_{32} + M_{33} + M_{34} + M_{35} + M_{36} \quad [11.0]$$

$$M_{31} = g \cdot m_o \cdot (d_3 \cdot \sin(\gamma) + d_2 \cdot \cos(\beta) - d_1 \cdot \sin(\alpha)) \quad [10.1]$$

$$M_{32} = g \cdot m_{d3} \cdot \left(\frac{d_3}{2} \cdot \sin(\gamma) + d_2 \cdot \cos(\beta) - d_1 \cdot \sin(\alpha) \right) \quad [10.2]$$

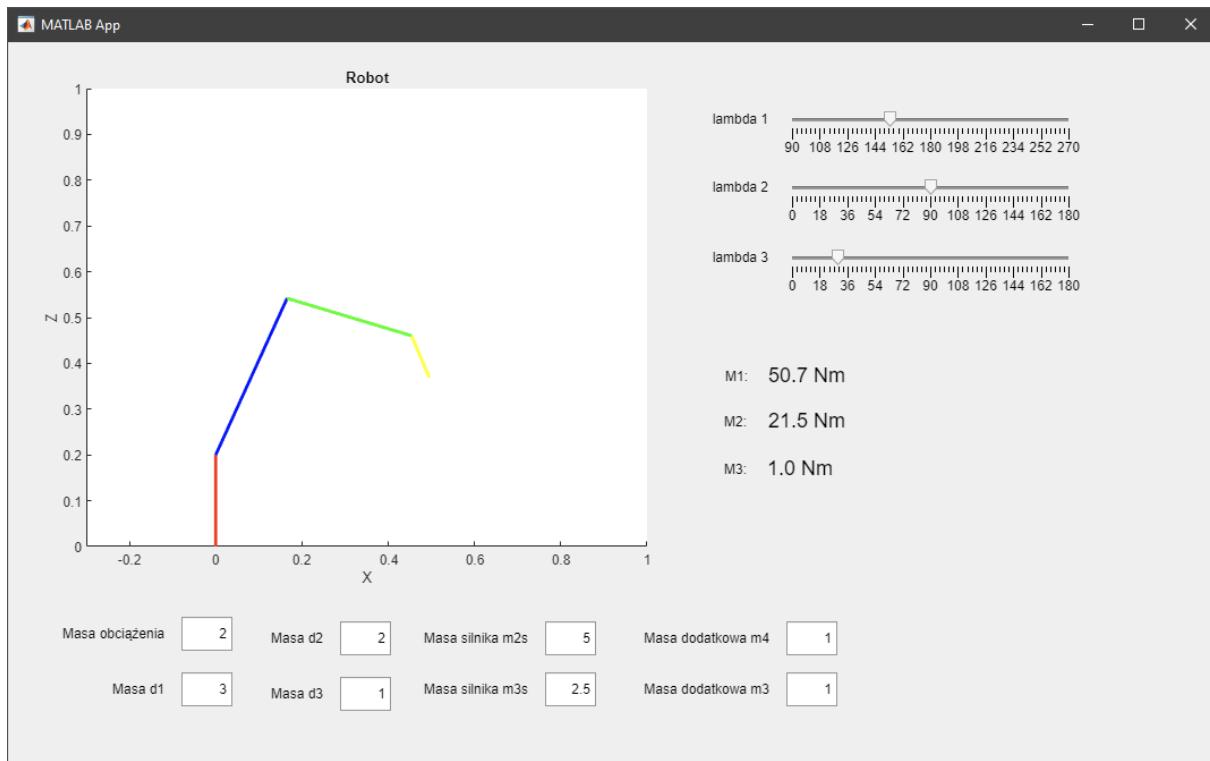
$$M_{33} = g \cdot m_4 \cdot (d_2 \cdot \cos(\beta) - d_1 \cdot \sin(\alpha)) \quad [10.3]$$

$$M_{35} = g \cdot (m_{d2} + m_{3s}) \cdot \left(\frac{d_2}{2} \cdot \cos(\beta) - d_1 \cdot \sin(\alpha) \right) \quad [10.4]$$

$$M_{34} = g \cdot m_3 \cdot (-d_1 \cdot \sin(\alpha)) \quad [10.5]$$

$$M_{35} = g \cdot (m_{d1} + m_{2s}) \cdot \frac{d_1}{2} \cdot \sin(\alpha) \quad [10.6]$$

Znając momenty sił dla każdego z możliwych ustawień ramienia, można dobrać odpowiedni system przeniesienia napędu. W tym celu konieczne jest wstępne oszacowanie masz poszczególnych elementów. W celu szybszego oraz prostszego uzyskania prawidłowych informacji o momentach sił stworzona została aplikacja w środowisku MATLAB.



Rys. 4.18. GUI aplikacji do obliczania obciążzeń

Jak pokazano na rysunku 4.18 na GUI aplikacji składają się 4 główne części. Pierwsza z nich to przestrzeń, w której rysowane jest uproszczone bieżące ułożenie ramion robota. Położenie to można zmieniać za pomocą suwaków które regulują wartości kątów λ_i zgodnie z nomenklaturą przyjętą na rysunku 3.3 a). Poniżej części rysującej ramię znajdują się pola numeryczne, w które można wprowadzać wartości konieczne do przeprowadzenia obliczeń momentów (wartości masy należy wprowadzać w kg). Wyniki tych obliczeń umieszczone są poniżej suwaków.

Tab. 3.1. Zestawienie długości potrzebnych do przeprowadzenia obliczeń momentów

d₁	d₂	d₃	d₄
400mm	300mm	100mm	200mm

Tab. 3.2. Przykładowe wyniki obliczeń obciążzeń dla różnych ustawień ramienia dla parametrów masy wynoszących $m_0 = 2\text{kg}$, $m_{d1} = 3\text{kg}$, $m_{d2} = 2\text{kg}$, $m_{d3} = 1\text{kg}$, $m_{2s} = 5\text{kg}$, $m_{3s} = 2.5\text{kg}$, $m_4 = 1\text{kg}$, $m_3 = 1\text{kg}$

Wartości kątów λ_i	M₁ [Nm]	M₂ [Nm]	M₃ [Nm]
$\lambda_1 = 90^\circ$ $\lambda_2 = 0^\circ$ $\lambda_3 = 0^\circ$	86,5	23,8	2,5
$\lambda_1 = 210^\circ$ $\lambda_2 = 90^\circ$ $\lambda_3 = 90^\circ$	5,4	19,7	1,2
$\lambda_1 = 130^\circ$ $\lambda_2 = 100^\circ$ $\lambda_3 = 130^\circ$	63,4	16,3	0,0

Chcąc wykorzystać taki model układu elementów ramienia, konieczne będzie dobranie odpowiednich przekładni. Przy ustawieniu, w którym momenty obciążeń będą największe (rysunek 3.11), wartość momentu M_3 wynosi niemal 90Nm. Silnik napędzający tę sekcję charakteryzuje się momentem maksymalnym o wartości 12Nm. Obliczając potrzebną przekładnię uzyskuje się:

$$n_1 = \frac{M_{wy}}{M_{we}} = \frac{90}{12} = 7.5 \quad [11.0]$$

Oznacza to, że minimalna przekładnia, jaka musi zostać wykorzystana powinna charakteryzować się przełożeniem 1:7.5 oraz maksymalnym wyjściowym momentem na poziomie 90Nm. Inną ważną cechą przekładni musi być brak samohamowności. Celem takiego podejścia jest umożliwienie ręcznej zmiany pozycji ramienia przy wyłączonym zasilaniu. Oznacza to, że np. przekładnie ślimakowe nie nadają się do tego zastosowania. Przełożenie drugim stopniu wiąże się z maksymalnym momentem obciążenia na poziomie około 30Nm oraz prezentuje się następująco:

$$n_2 = \frac{M_{wy}}{M_{we}} = \frac{25}{9} = 2.8 \quad [11.1]$$

Przełożenie dla napędu ostatniej sekcji może wynosić nawet 1:1, ponieważ moment generowany przez silnik wynosi 4.8Nm, a przy założonej maksymalnej masie obciążenia na poziomie 2kg, moment obciążenia wynosi 2.5Nm, co pokazuje, że nie jest potrzebne żadne przełożenie.

Przeglądając katalogi różnych firm produkujących przekładnie bez problemu można zaleźć odpowiedni model spełniający założenia dla przekładni n_1 . Jednym z nich jest przekładnia planetarna przystosowana do montażu z silnikami NEMA34, co jest dodatkowym atutem. Parametry tej przekładni to [19]:

- Przełożenie 15:1
- Maksymalny moment wyjściowy 160Nm
- Cena 790zł



Rys. 4.19. Przekładnia planetarna [19]

O ile parametry przekładni spełniają wszystkie założone kryteria, tak jej cena znaczaco wykraca poza przeznaczony budżet. Ze względu na konieczność zastosowania tak drogich części konieczne będzie przeprojektowanie rozkładu elementów w ramieniu tak, aby zmniejszyć wartości momentów obciążenia w poszczególnych sekcjach.

Największymi masami jakie znajdują się na ramieniu są masy silników m_{2s} oraz m_{3s} . Przenosząc te dwa elementy poza ruchomą część ramienia możliwe będzie znaczące odciążenie konstrukcji. W tym celu konieczne będzie jeszcze przeniesienie napędu w miejsce, w którym był w poprzednim modelu. Może to być zrealizowane poprzez przekładnie pasowe. Dodatkowo ograniczając udźwig maksymalny do 1.5kg (na maksymalnym wysięgu – rysunek 3.11) możliwe jest uzyskanie mniejszych momentów obciążających poszczególne osie.

Tab. 4.3. Porównanie momentów obciążen i przekładni przed i po wprowadzeniu zmian

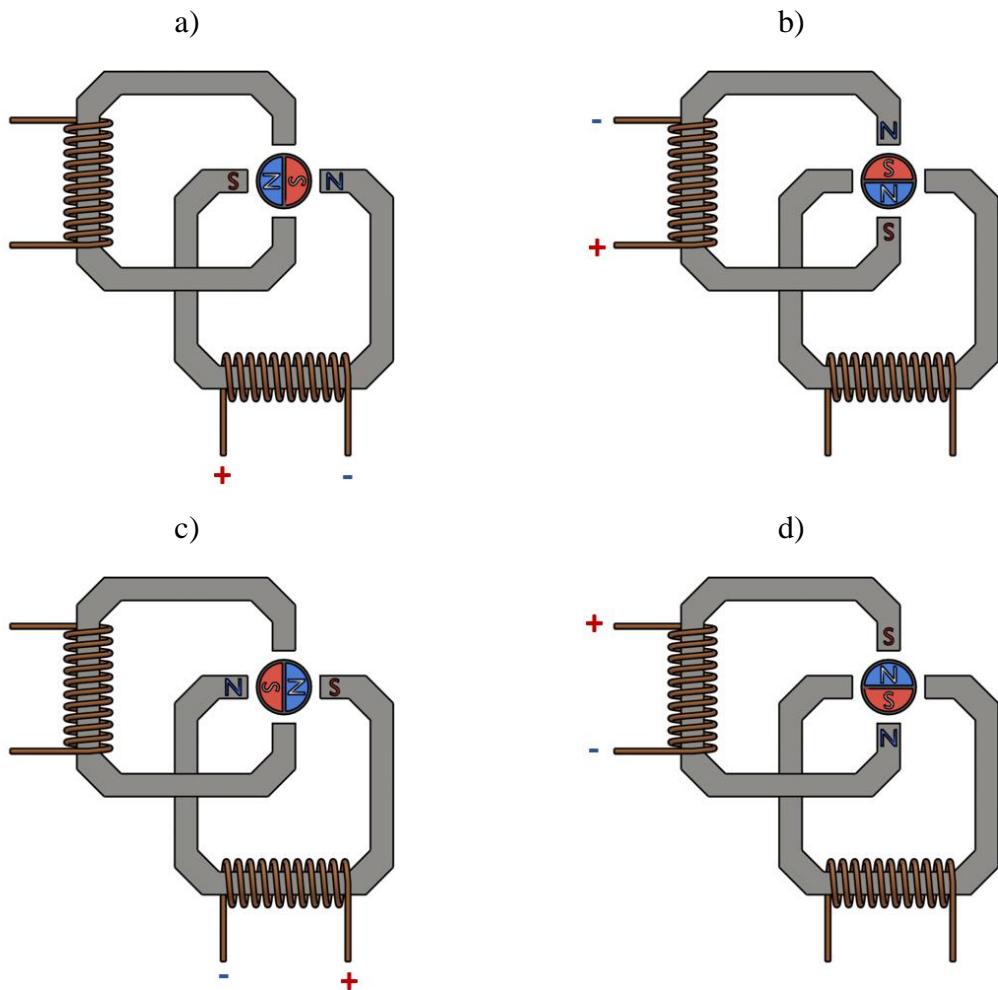
Wartości momentów	M₁ [Nm]	M₂ [Nm]	M₃ [Nm]
Układ przed zmianami	86,5	23,8	2,5
Układ po zmianach	39,8	13,7	2,0
Potrzebne przekładnie	n₁	n₂	n₃
Układ przed zmianami	7.5:1	2.8:1	1:1
Układ po zmianach	3.3:1	1.5:1	1:1

Ważną informacją jest fakt, że podane w powyższej tabeli przełożenia są minimalnymi jakie mogą być wykorzystane. Jeśli przełożenie docelowej przekładni będzie większe, wówczas nie wpłynie to na wytrzymałość mechaniczną ramienia, a tylko zwiększy precyzję pozycjonowania ramienia.

4.4.2. Opis teoretyczny wykorzystanych układów napędowych

Silniki krokowe są to silniki elektryczne, w których obrót wału zachodzi zawsze o pewien określony kąt. Cechą tego silnika jest jego sposób zasilania, który w przeciwieństwie do silników prądu przemiennego, w których prąd jest o charakterze przemiennym ale ciągłym, tak w silnikach krokowych ma on charakter impulsowy.

Podstawową zasadę działania silnika krokowego można przedstawić za pomocą układu dwóch faz oraz wirnika w postaci magnesu trwałego. Dla uproszczenia, w jednej chwili czasowej prąd może płynąć tylko przez jedno uzwojenie. Zasilając jedną fazę silnika tworzy się pole magnetyczne, które na rysunku 3.20 zostało symbolicznie oznaczone literami S oraz N. Ze względu na budowę wirnika, powstaje siła, która obróci wirnik tak, aby biegun S twornika znalazł się w pobliżu bieguna N wirnika, a biegun N twornika znalazł się w pobliżu bieguna S wirnika (rysunek 3.20. a). Jeśli następnie zasilimy drugą fazę w sposób pokazany na rysunku 3.20. b, wówczas wirnik obróci się o 90° w lewo. Zachowując kolejność łączenia faz wynikającą z kolejności przedstawionej na rysunku 3.20. uzyskamy ruch wirnika w lewo przy kroku wynoszącym 90° . Zmieniając kolejność łączenia poszczególnych faz (uwzględniając kierunek przepływu prądu) na układ a) → d) → c) → b) uzyskamy ruch wirnika w prawo. Zachowanie takie pokazuje, że jesteśmy w stanie kontrolować położenie wału silnika w pętli otwartej, co jest bardzo dużym atutem [20].



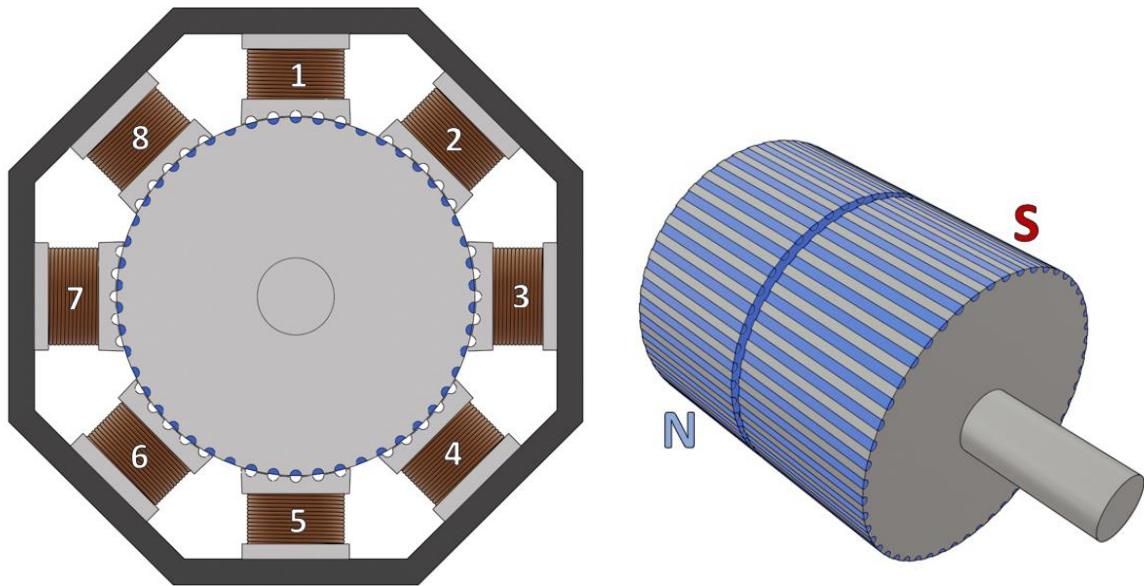
Rys. 4.20. Uproszczony schemat działania silnika krokowego

Ważnym zagadnieniem jest sterowanie silnikiem przy tzw. mikrokroku. W tym przypadku można jednocześnie zasilać obie fazy, co wymusi obrót silnika o kąt mniejszy niż podstawowy krok. Odpowiednio różnicując prądy w każdej fazie możliwe jest uzyskanie pracy przy mikrokroku równym nawet $1/256$ kroku podstawowego. Dzięki takiemu podejściu niwelowane są drgania silnika, a jego ruch jest płynniejszy

Pełny model działania najczęściej spotykanego typu silnika krokowego jest bardziej rozbudowany. W klasycznym przykładzie często spotykane są silniki z czterema wyprowadzeniami (silniki bipolarne), tak jak jest to widoczne na rysunku 3.20. Fazy te są dodatkowo rozmiieszczane na 8 cewkach. Model takiego silnika zaprezentowany jest na rysunku 3.21. Przyjmując, że fazy oznaczone są jako A i B, można przypisać fazie A cewki: 1, 3, 5 oraz 7, natomiast fazie B cewki: 2, 4, 6 oraz 8. Wirnik silnika krokowego jest tak skonstruowany, żeby korzystał zarówno z właściwości wirników reluktancyjnych oraz wirników z magnesami trwałymi. Takie połączenie tworzy układ nazywany silnikiem hybrydowym. Po jednej stronie wirnika znajdują się żlobki z biegunem S, natomiast po drugiej stronie żlobki z biegunem N.

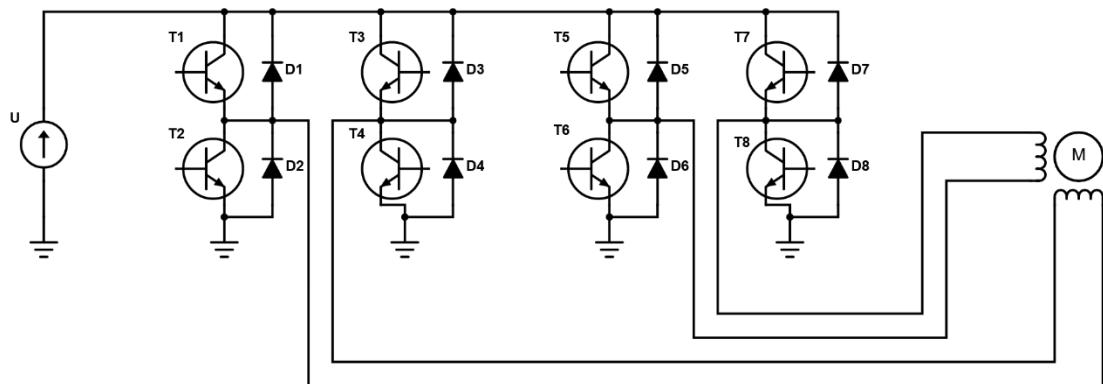
Zasilając fazę A wytwarza się północny N biegun magnetyczny w cewce 1 oraz 5. Cewki 7 oraz 3 wytworzą biegun południowy S. Dzięki żlobkowej budowie wirnika, żlobki z

biegunem S wirnika przyciągane są do żlobków stojana z biegunami N. Załączając fazę B przy analogicznym sterowaniu jak te przedstawione na rysunku 3.20. uzyskuje się zdyskretyzowany ruch w odpowiednim kierunku. Dla konstrukcji silnika zamieszczonej na rysunku 3.20. krok obrotu wynosi 1.8° , co przekłada się na 200 kroków na obrót [21].



Rys. 4.21. Pełny model silnika krokowego

Do sterowania silnikami krokowymi konieczne jest wykorzystanie specjalnych kart sterowniczych. W najprostszym podejściu karty te składają się z dwóch mostków H.



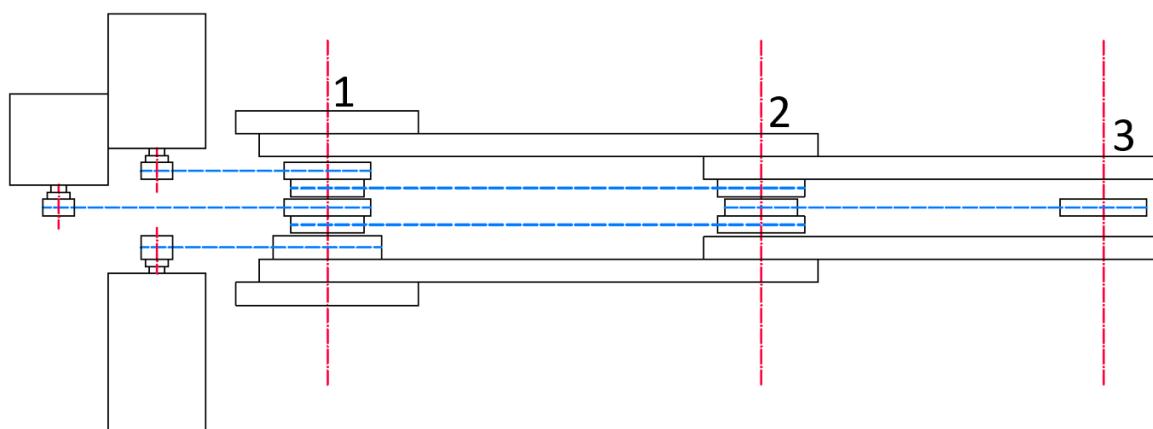
Rys. 4.22. Schemat podwójnego mostka H do sterowania bipolarnym silnikiem krokowym

Jak pokazano na rysunku 4.22, do tranzystorów sterujących prądamи równolegle dołączone zostały zaporowo spolaryzowane diody. Najczęściej stosuje się szybkie diody np. diody Schottky'ego. Ich celem jest ochrona tranzystorów przeciw napięciu pochodzącemu z uzupełnienia silnika pojawiającemu się podczas gwałtownych zmian prądu. Poprzez załączanie odpowiednich tranzystorów uzyskujemy możliwość kontroli kierunku przepływu prądów przez uzupełnienia.

4.5. Konstrukcja ramienia

Znając ogólny zamysł, który został zarysowany przez wymagania zebrane podczas projektowania kinematyki ramienia, wykonywania obliczeń oraz dobierania materiałów można wyróżnić podstawowe cechy które muszą zostać odzwierciedlone podczas tworzenia konstrukcji:

- odległości pomiędzy poszczególnymi osiami ramienia:
 - $d_1 = 400\text{mm}$,
 - $d_2 = 300\text{mm}$,
 - $d_3 \approx 100\text{mm}$,
 - $d_4 \approx 100\text{mm}$,
- wycofanie silników poza konstrukcję ramienia, oraz umieszczenie ich na platformie obrotowej ramienia,
- wykorzystanie aluminiowej blachy o grubości 20mm jako bazy do tworzenia poszczególnych elementów,
- wykorzystanie przekładni (pasowych) o wartościach przynajmniej równych tym, zamieszczonym w tabeli 4.3.



Rys. 4.23. Widok z góry wstępniego projektu ramienia z uwzględnieniem przeniesienia napędu

Jak pokazano na rysunku 4.23 umiejscowienie silników spełnia założenia projektowe. Napęd jest dostarczany w odpowiednie miejsca za pomocą przekładni pasowych, które jednocześnie są odpowiedzialne za zwiększenie przenoszonego momentu. W celu przeniesienia napędu do sekcji numer 2 i 3 konieczne jest obejście występującego w każdym stawie połączenia ruchomego. W tym celu wykorzystane zostaną dwa sprzężone ze sobą koła zębate. Silnik napędzał będzie większe z kół w takiej parze, następnie mniejsze z kół będzie napędzało większe koło w zestawie dwóch sprzężonych kół na kolejnym stopniu, a mniejsze koło w zestawie będzie przenosiło napęd do następnej sekcji. Opisana zasada została wykorzystana do przeniesienia napędu do ostatniej sekcji. W przypadku napędu osi numer 2 występuje taka sama analogia. Pierwsza oś będzie napędzana za pomocą jednego pasa zębnego. Sumaryczna ilość kół zębatych znajdująca się na samym ramieniu (nie wliczając kół zębatych znajdujących się na wałach silników oraz zestawu do napędu osi pionowej) wynosi 8.

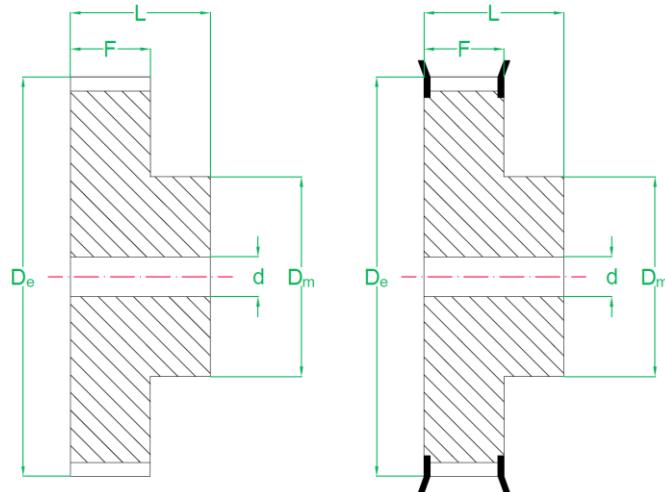
Powołując się na wyliczone w poprzednim rozdziale minimalne wartości przełożień, które prezentują się następująco:

- przekładnia na pierwszym stopniu: $n_i = 3.3:1$,
- przekładnia na drugim stopniu: $n_i = 1.5:1$,
- przekładnia na trzecim stopniu: $n_i = 1:1$,

dobrano koła zębate z grupy kół z oznaczeniem AT5. Ich głównymi cechami są:

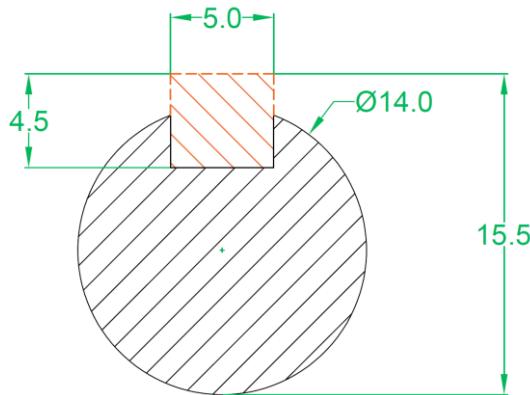
- przystępna cena, która ma duże znaczenie przy planowanej ilości potrzebnych kół,
- szeroka gama rozmiarowa kół od nawet 15 (z15) do 60 (z60) zębów w kole,
- materiał z którego są wykonane to aluminium co przy zachowaniu wystarczającej wytrzymałości zapewni odpowiednią lekkość konstrukcji,
- kompaktowe rozmiary kół – średnica największego koła (z60) nie przekracza 100mm
- możliwość doboru koła dla różnych szerokości pasów zębatych (10mm, 16mm, 25mm),
- Koło zębate wraz z odpowiednim pasem zębatym zapewnia bardzo małe luzy.

W celu dokładnego zrozumienia dalszego opisu należy przyjąć odpowiednią nomenklaturę związaną z parametrami kół zębatych. Poniższy rysunek prezentuje najważniejsze informacje [22].



Rys. 4.24. Charakterystyczne wymiary kół zębatych (po lewej wykonanie „6”, a po prawej wykonanie „6F”)

Wybór kół zębatych, które zostaną zamontowane na wałach silników podykutowany jest ich minimalną możliwą średnicą, którą narzuca średnica wałów silników. Średnica ta wynosi 14mm, przy czym należy uwzględnić również rozmiar elementu blokującego koło zębate na wale. Pokazano to na rysunku 4.25.



Rys. 4.25. Wymiary wału silnika

Wiedząc, że otwór w kole zębatym maksymalnie będzie szeroki na 15,5mm, można przyjąć, że minimalna wartość średnicy D_m musi wynosić około 20mm. Przeglądając katalogi kół zębatych AT5 można znaleźć koło o zakładanych parametrach. Jest to koło o 18 zębach, pokazane na rysunku 4.26.



Rys. 4.26. Przykładowe koło zębate AT5 [23]

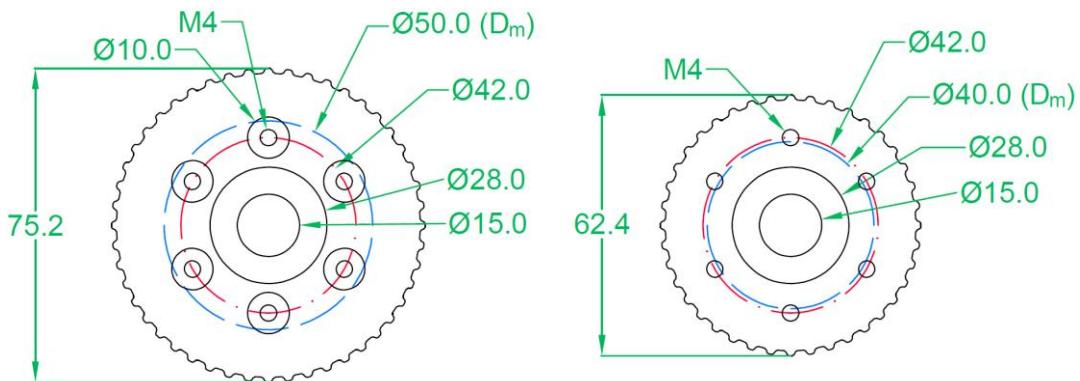
Dobierając przekładnię dla napędu stopniem pierwszym, konieczne jest zastosowanie przełożenia o wartości 3.3:1. Wiedząc, że w przypadku kół zębatych przekładnia to stosunek ilości zębów w jednym kole do ilości zębów w drugim, można łatwo wyznaczyć ilość zębów jakie musi mieć większe koło.

$$z_2 = z_1 \cdot n_1 = 18 \cdot 3.3 = 59.4 \approx 60 \quad [12.0]$$

Przeglądając katalog kół zębatych największe koło z grupy AT5 posiada 60 zębów (z60). Oznacza to, że idealnie nadaje się ono do tej aplikacji.

Dobór kół zębatych do napędu osi numer 2, powinien skutkować wypadkowym przełożeniem większym bądź równym 1.5:1. Sposób przeniesienia napędu (zgodny z rysunkiem 3.24.) zakłada wykorzystanie dwóch pętli pasa zębnego, co jest równoznaczne z czterema kołami zębnymi. Taki układ przewiduje wykorzystanie jednej pary kół sprzężonych ze sobą. W celu odpowiedniego dobrania kół zębatych należy obrać minimalną wartość parametru D_m . Średnica osi, które zostaną wykorzystane została wstępnie przyjęta na 15mm. Aby koła zębate mogły się swobodnie wokół niej obracać, konieczne jest wykorzystanie łożysk. W tym przypadku wybór padł na łożyska kulkowe o średnicy wewnętrznej 15mm, średnicy zewnętrznej 28mm oraz szerokości 7mm. Znając średnicę zewnętrzną łożyska, możliwe jest

określenie minimalnej wartości parametru D_m dla kół znajdujących się na osi 1. Ze względu na konieczność sprzężenia ze sobą dwóch kół w celu stworzenia wcześniej wspominanej pary kół zębatych należy pamiętać o uwzględnieniu miejsca na połączenie ze sobą kół. W tym celu konieczne będzie uwzględnienie otworów pod śruby. Zakładając wykorzystanie śrub M4, konieczne będzie dopasowanie wartości parametru D_m tak aby zachować wystarczającą wytrzymałość. Najmniejsze koło zębate które spełnia powyższe założenia posiada 40 zębów (z40). Drugim kołem jakie zostało wstępnie dobrane w celu skompletowania pary kół zębatych będzie koło o ilości zębów wynoszącej 48 (z48). Sposób połączenia obu kół został przedstawiony na rysunku 4.27.



Rys. 4.27. Najważniejsze wymiary pary kół zębatych

Jak pokazano na powyższym rysunku, aby wykorzystanych śrub zostaną zagłębione wewnętrzko koła tak, aby licowały się z ścianą boczną koła zębnego. Średnica D_m dla mniejszego z kół zębatych przecina tor otworów pod śruby łączeniowe, jednak materiał, który pozostaje między krawędzią zębów oraz brzegiem otworu śrubowego jest wystarczający na tyle, że koło zachowuje wymaganą wytrzymałość. W celu zapewnienia możliwości swobodnego obracania się zespołu kół po osi, konieczne jest zamontowanie łożysk. Do każdej pary kół zębatych dołożone zostaną dwa łożyska, które zostaną zlicowane z zewnętrznymi ścianami kół zębatych. Ważną informacją jest fakt, że koła w opisywanej parze są przystosowane do pracy z pasami o szerokości 10mm. Efektywna szerokość koła zębnego (wartość parametru L) wynosi 15mm, co oznacza to, że szerokość pary kół będzie wynosiła 30mm.



Rys. 4.28. Wykonana para kół zębatych

Znając parametry kół zębatych jakie zostaną wykorzystane do przeniesienia napędu na drugą oraz trzecią os, można obliczyć wypadkowe przełożenia. Przełożenie dla drugiej osi n_2 oraz dla 3 osi n_3 można obliczyć jako:

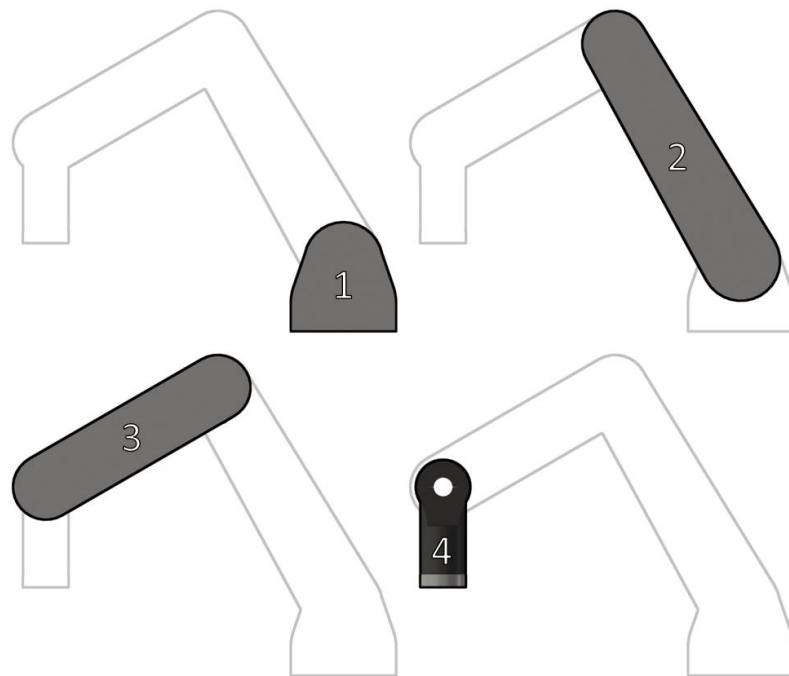
$$n_2 = \frac{z_{12}}{z_{11}} \cdot \frac{z_{22}}{z_{21}} = \frac{48}{18} \cdot \frac{48}{40} = 3.2 \quad [13.0]$$

$$n_2 = \frac{z_{12}}{z_{11}} \cdot \frac{z_{22}}{z_{21}} \cdot \frac{z_{32}}{z_{31}} = \frac{48}{18} \cdot \frac{48}{40} \cdot \frac{40}{40} = 3.2 \quad [13.1]$$

gdzie:

- z_{11} – ilość zębów w kole zębatym zamontowanym na wale silnika,
- z_{12} – ilość zębów w kole zębatym połączonym paskiem z kołem opisanym jako z_{11} ,
- z_{21} – ilość zębów w kole zębatym sprężonym z kołem z_{12} ,
- z_{22} – ilość zębów w kole zębatym połączonym paskiem z kołem opisanym jako z_{21} ,
- z_{31} – ilość zębów w kole zębatym sprężonym z kołem z_{12} ,
- z_{32} – ilość zębów w kole zębatym połączonym paskiem z kołem opisanym jako z_{31} .

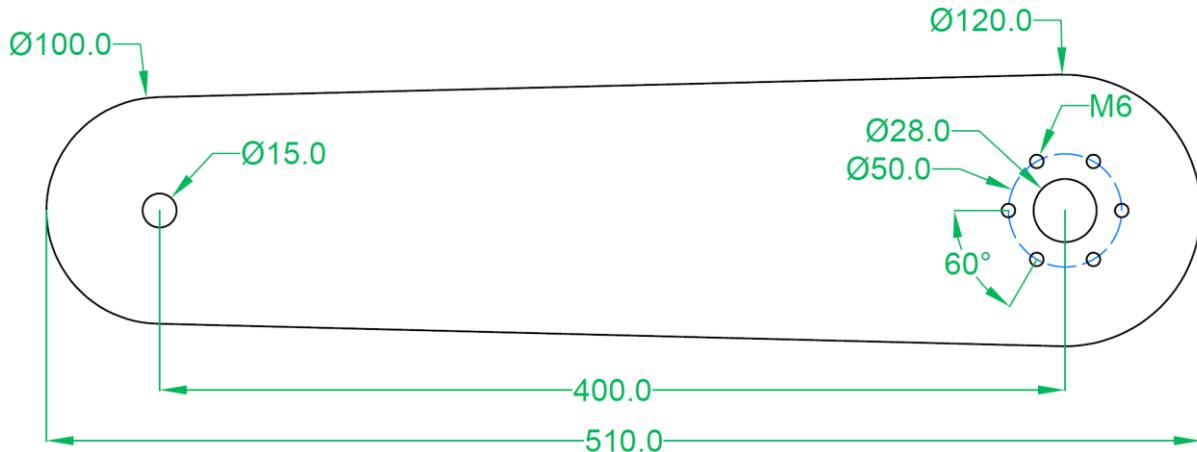
Wiedząc, w jaki sposób zostanie przeniesiony napęd do 2 i 3 osi możliwe jest zaprojektowanie części konstrukcyjnych ramienia. Konstrukcja ta zostanie podzielona na 4 główne sekcje. Podział sekcji zaprezentowany został na rysunku 4.29.



Rys. 4.29. Główne sekcje ramienia

Jak pokazano na powyższej grafice, wyróżnione zostały 4 główne sekcje. Przyjęte wcześniej założenia, mówią, że odległość między osiami w sekcji 2 ma wynosić 400mm. Kształt tej sekcji zostanie oparty o dwa połączone ze sobą okręgi. Większy z okręgów będzie musiał charakteryzować się średnicą większą od średnicy D_e koła zębatego napędzającego tą sekcję. Oznacza to, że średnica musi być większa od 100mm. Docelowa średnica została ustalona na 120mm. Drugi z okręgów został określony w jednakowy sposób, a jego średnica

wynosi 100mm. W celu swobodnego poruszania się konstrukcji wokół osi konieczne jest wykorzystanie łożysk oraz stworzenie otworów montażowych do przykręcenia koła zębatego. Końcowy model drugiej sekcji ramienia przedstawiony jest na rysunku 4.30.



Rys. 4.30. Widok z przodu drugiej sekcji ramienia

Sekcja ta docelowo będzie składała się z dwóch symetrycznych części ustawionych równolegle względem siebie w celu zapewnienia większej sztywności ramienia. Jedyną różnicą pomiędzy częściami tej sekcji będzie brak otworów montażowych pod koło zębate w jednej z nich. Jak pokazano na rysunku 4.30, w dolnej części koła (charakteryzującej się większą średnicą okręgu) jest miejsce pod łożysko kulkowe. Wykorzystane w tym celu zostaną łożyska o średnicy wewnętrznej 15mm, średnicy zewnętrznej 28mm oraz szerokości 7mm. Z drugiej strony sekcji widoczny jest otwór, przez który będzie przechodziła oś trwale przymocowana do tej sekcji. Grubość materiału (aluminium) z jakiego zostanie wykonana ta sekcja wynosi 20mm. Ze względu na masę, do jakiej należy się zbliżyć w celu nie przekroczenia momentów obciążen wziętych pod uwagę w części obliczeniowej pracy (rozdział 3.2.1.), konieczna jest redukcja masy z początkowej całkowitej masy obu części wynoszącej niemal 3kg do około 1.5kg na część. Końcowa waga pojedynczego ramienia sekcji drugiej wynosi 1.75kg.



Rys. 4.31. Wykonana druga sekcja ramienia wraz z zamocowanym kołem zębatym

Tworzenie modelu oraz konstrukcji trzeciej sekcji bazuje na podobnym podejściu. Wstępnie zdefiniowana odległość pomiędzy osiami sekcji trzeciej wynosi 300mm. W tym przypadku wybór padł na oparcie kształtu o dwa okręgi. Średnica większego z nich została

zdefiniowana poprzez średnicę mniejszego z okręgów w sekcji drugiej oraz wynosi 100mm. W przypadku mniejszego z okręgów średnica została ustalona na 80mm. Podobnie jak w poprzedniej sekcji konieczne jest utworzenie otworów pod łożyska. Wykorzystane zostaną również łożyska kulkowe o parametrach: średnica wewnętrzna 15mm, średnica zewnętrzna 28mm oraz szerokość 7mm. Na drugim końcu tej sekcji również znajduje się otwór o średnicy 15mm przeznaczony do osadzenia w nim osi, która będzie trwale skręcona z konstrukcją sekcji trzeciej. Materiał wykorzystany do stworzenia tej sekcji to również aluminium, a jego grubość przed obróbką wynosi 20mm.



Rys. 4.32. Widok z przodu trzeciej sekcji ramienia

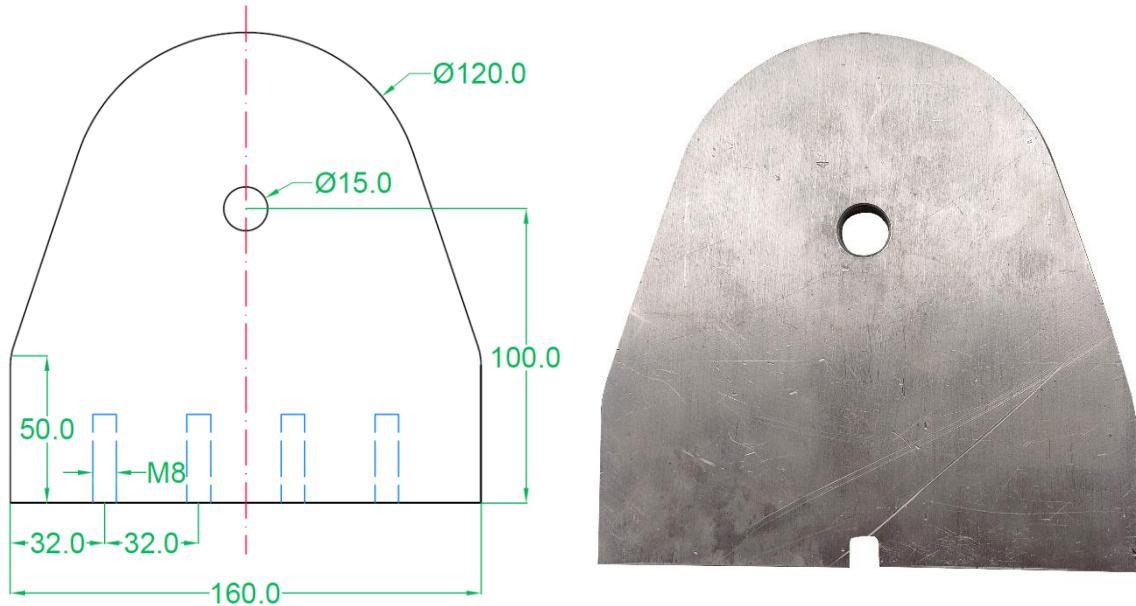


Rys. 4.33. Wykonana trzecia sekcja ramienia wraz z zamocowanym kołem zębatam

Sumaryczna masa dwóch wykonanych elementów zgodnych z kształtem z rysunku 4.32 wynosi 4.2kg, jednak konieczne jest jej obniżenie do 2kg aby dopasować ją do wcześniejszych przyjętych założeń. W tym celu została wycięta kratownica, a ostateczny efekt pokazano się na rysunku 4.33. Po zabiegu wycięcia kratownicy łączna masa sekcji wynosi 1.9kg.

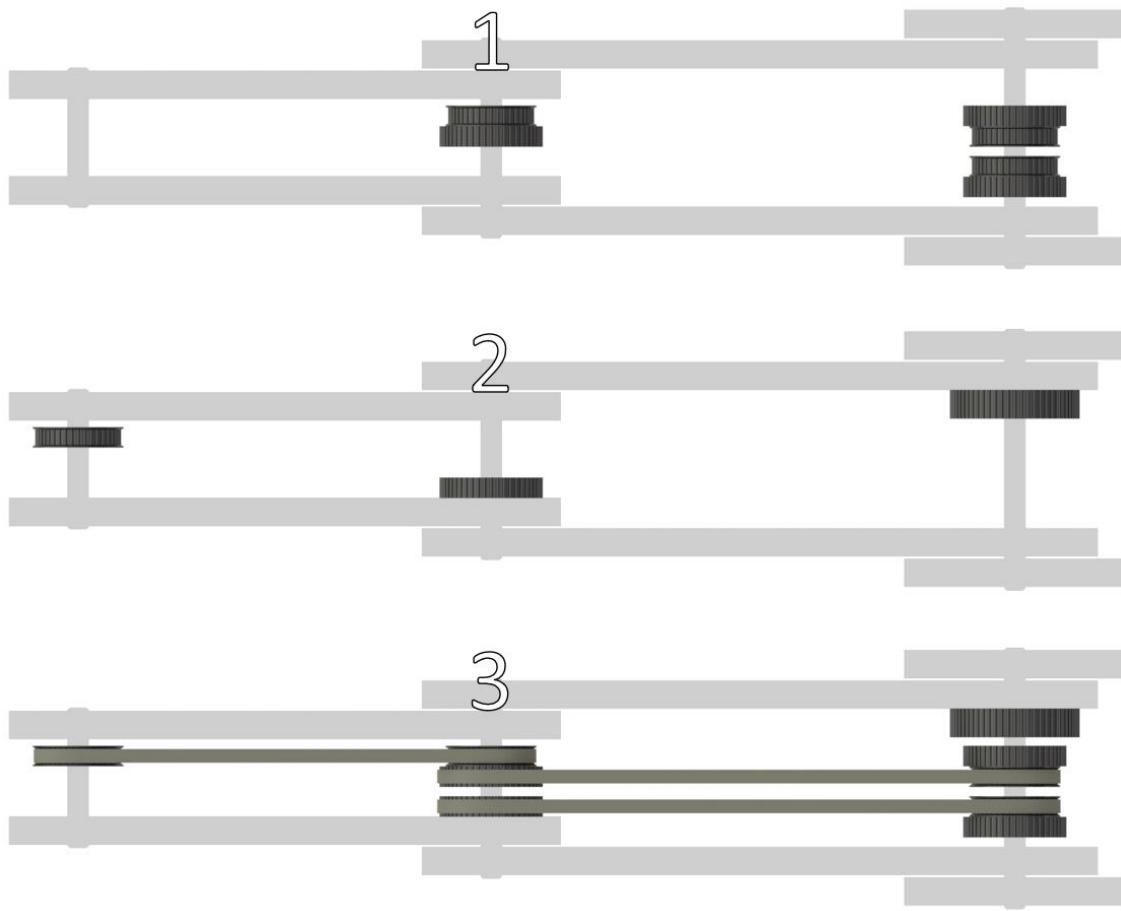
Projekt sekcji pierwszej zostanie podyktowany kształtem sekcji drugiej. Głównym parametrem jest średnica okręgu w dolnej części sekcji drugiej, czyli 120mm. Wartość ta będzie średnicą okręgu wykorzystanego do konstrukcji głównego elementu. Element ten będzie zawierał otwór przeznaczony do osadzenia w nim osi. Oś ta będzie wykonana z wałka stalowego o średnicy 15mm. Wałek ten będzie unieruchomiony w konstrukcji sekcji pierwszej.

Bardzo ważnym elementem będzie mocowanie sekcji pierwszej do podstawy obrotowej całego ramienia. W tym celu zostaną wykorzystane cztery śruby M8 które przymocują każdą z dwóch części do płyty stanowiącej podłożę. Odległość od osi obrotowej do podłoża została przyjęta jako 100mm. Wartość ta została dobrana ze względu na fakt, że dla mniejszych wartości tego parametru, kiedy ramię było wychylone w stronę platformy na której mają znaleźć się silniki mogło dochodzić do kolizji konstrukcji ramienia oraz obudowy silników. Całkowita szerokość sekcji pierwszej została przyjęta jako 160mm dzięki czemu możliwe będzie uzyskanie odpowiedniej sztywności konstrukcji ramienia.



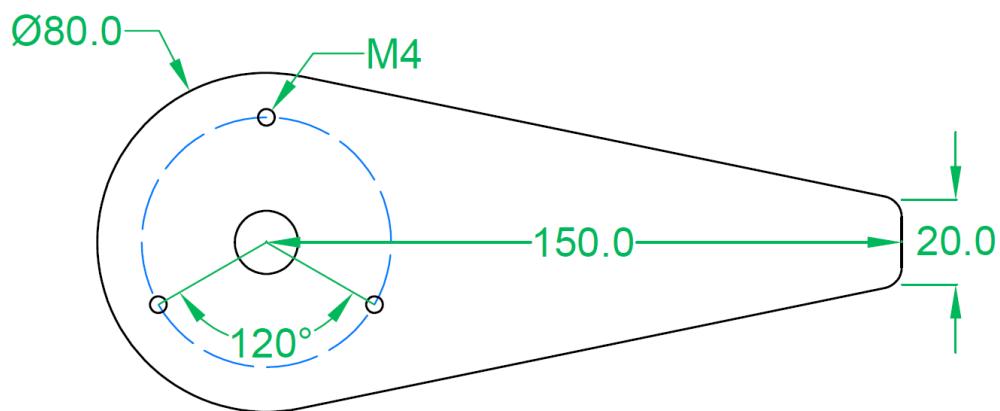
Rys. 4.34. Widok z przodu pierwszej sekcji ramienia oraz widok gotowego elementu

Zanim możliwe będzie stworzenie projektu sekcji 4, konieczne będzie odpowiednie rozmieszczenie kół zębatych, pokazane na rysunku 4.35. W części oznaczonej numerem 1 widoczny jest sposób rozmieszczenia zaprojektowanych wcześniej par kół zębatych. Jak widać dwie pary znajdują się na pierwszej osi, oraz jedna para znajduje się na drugiej osi. W części rysunku oznaczonej numerem 2, widoczne są koła zębate odpowiedzialne za napędzanie poszczególnych sekcji ramienia. Idąc od lewej strony: pierwsze koło napędza sekcję 4, drugie koło napędza sekcję 3, a ostatnie koło napędza sekcję pierwszą. W części rysunku oznaczonej numerem 3 widoczne są już połączenia pasowe poszczególnych kół. Trzy koła znajdujące się na pierwszej osi nie są połączone pasami z żadnymi innymi kołami, jednak docelowo będą one napędzane przez koła zębate znajdujące się na wałach silników.



Rys. 4.35. Rozmieszczenie kół zębatych

Wiedząc jaki kształt przyjmie całe ramię, możliwe będzie stworzenie sekcji czwartej. Podobnie jak poprzednie elementy, również sekcja czwarta zostanie wykonana z aluminium. Element ten ma być również na tyle uniwersalny, żeby do jego końca można było przymocować dowolne narzędzie. Ważnym aspektem kształtu tej sekcji jest konieczność maksymalnego ograniczenia jej masy, ponieważ znajdująca się najdalej od środka obrotu sekcji drugiej jest częścią bardzo wpływającą na moment obciążający sekcję drugą.



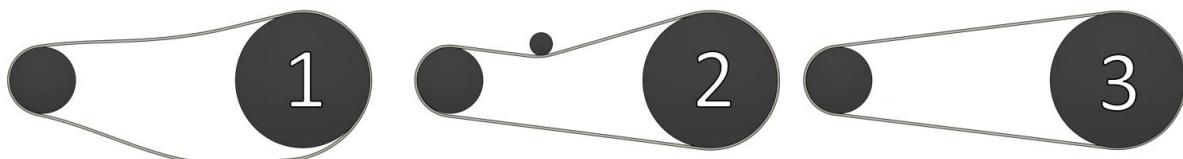
Rys. 4.36. Projekt sekcji czwartej

Element ten został wykonany również z blachy o grubości 20mm, jednak jego końcowa grubość została zredukowana do 15mm. Podobnie jak w przypadku poprzednich sekcji zostało wykonane odcięcie elementu poprzez wyfrezowanie znacznej części materiału, dzięki czemu ustalono masę elementu na 2 kg. W projekcie części znajdują się również otwory montażowe, przez które sekcja ta ma zostać przymocowana do koła zębatego. Dodatkowo, w części w której znajduje się docelowy punkt wodzący ramienia umieszczony został otwór, do którego można zamontować dowolny element będący efektorem robota. Na rysunku 4.37 pokazano wykonaną sekcję 4 ramienia robota.



Rys. 4.37. Gotowa sekcja czwarta wraz z zamocowanym kołem zębataym

Część ruchoma ramienia w celu poprawnego funkcjonowania musi zostać osadzona na platformie, na której znajdują się również silniki napędzające poszczególne osie. Platforma ta będzie dodatkowo pełniła rolę pionowej osi obrotu ramienia. W tym celu do platformy zostanie przymocowany element konstrukcyjny w kształcie długiego walca zakończonego tarczą z otworami. Tarcza ta zostanie przykręcona do spodu platformy, a na wale zostaną osadzone łożyska w celu umożliwienia swobodnego obracania się ramienia wokół osi z (rysunek 3.4.). Wymiary platformy są podytowane ustawieniem na niej silników. Oznacza to, że im dalej zostaną odsunięte silniki od krawędzi sekcji pierwszej (rysunek 3.23.) tym dłuższa musi być platforma. W celu zachowania odpowiedniego naprężenia pasków zębatach, które ma zapobiegać ślizganiu się pasa po kole zębataym (rysunek 3.38. część 1) co stwarza duże zagrożenie, konieczne jest zastosowanie systemu naciągania pasków. Można to zrealizować poprzez dołożenie dodatkowego koła (rysunek 3.38. część 2), którego celem będzie naprężenie pasa bez konieczności regulacji położenia osi kół zębatach które są połączone pasem. Innym sposobem jest wykorzystanie faktu, że położenie silników może być regulowane co daje możliwość odsunięcia łożyska silnika w taki sposób aby naprężyc pas (rysunek 3.38. część 3).



Rys. 4.38. Przedstawienie sposobów naprężenia pasa zębatego

Ze względu na oszczędność miejsca wybrana została metoda napinania z ruchomym łożem silnika. Nie będzie wówczas konieczne montowanie dodatkowych uchwytów pod rolki

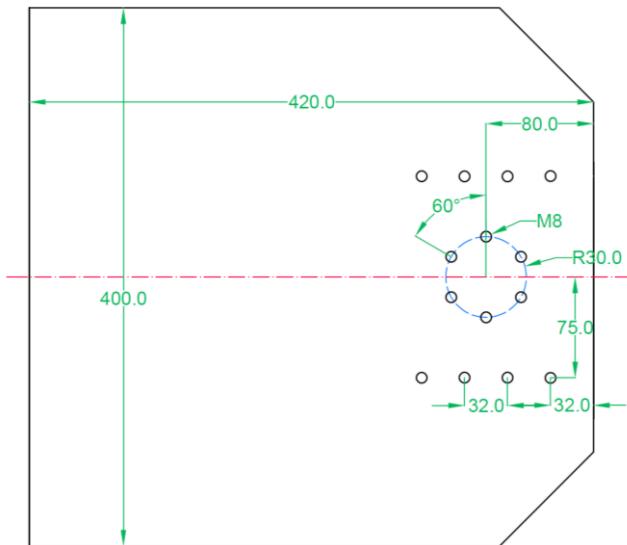
napinające. Ze względu na fakt, że na platformie znajdą się 3 silniki, konieczny był by montaż 3 rolek napinających. W przypadku wykorzystania ruchomego łożyska silnika wykorzystane zostaną do tego celu łożyska, które i tak musiałyby pojawić się w wersji z rolkami.



Rys. 4.39. Ruchome łożysko silnika

Rysunek 4.39 przedstawia wykonane łożyska umożliwiające napinanie pasków w silniku. Podobne łożyska zostały wykonane dla pozostałych silników znajdujących się na płycie obrotowej. W przypadku silnika realizującego obrót wokół osi z, konieczne jest zastosowanie innego podejścia, które jest wymuszone koniecznością umieszczenia silnika wewnątrz konstrukcji ramy stanowiącej podstawę robota. Sposób napinania pasków pozostaje jednak taki sam – regulacja położenia łożyska silnika.

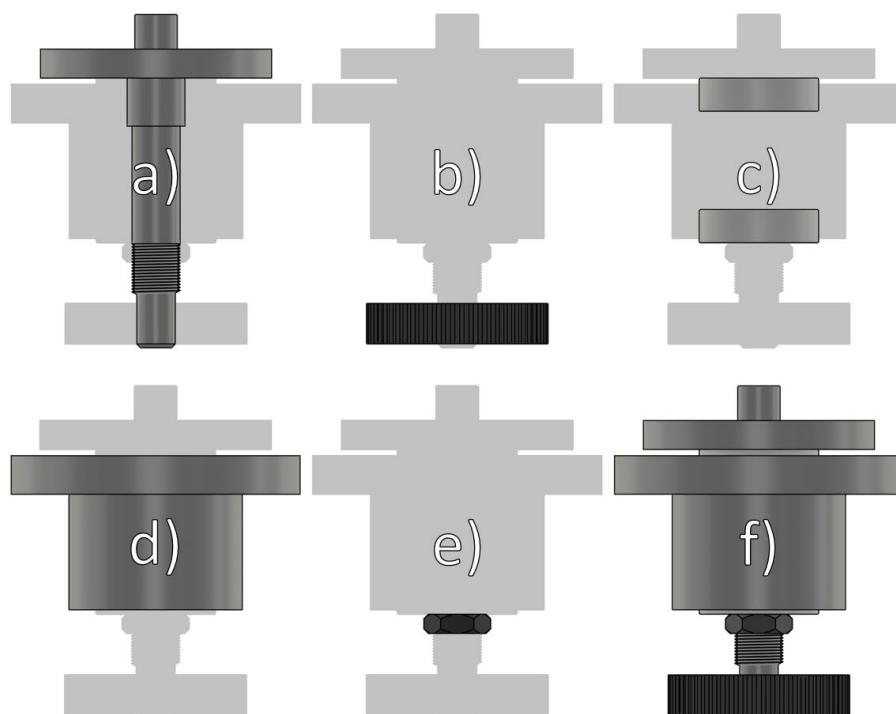
Ze względu na ilość elementów, które muszą znaleźć się na płycie obrotowej, konieczne jest wykonanie jej w odpowiednio dużym formacie. Docelowa płyta została wykonana z arkusza blachy aluminiowej o grubości 20mm. Wymiary główne płyty wynoszą 420mm długości, 400mm szerokości oraz 18mm grubości. Taka grubość pozwoli uniknąć efektu uginania się płyty pod ciężarem silników. Płyta zawiera otwory montażowe pod elementy takie jak sekcję pierwszą ramienia (która składa się z dwóch symetrycznych elementów), część układu obrotnicy oraz otwory do montażu silników.



Rys. 4.40. Wymiary główne wraz z niektórymi otworami montażowymi płyty obrotowej

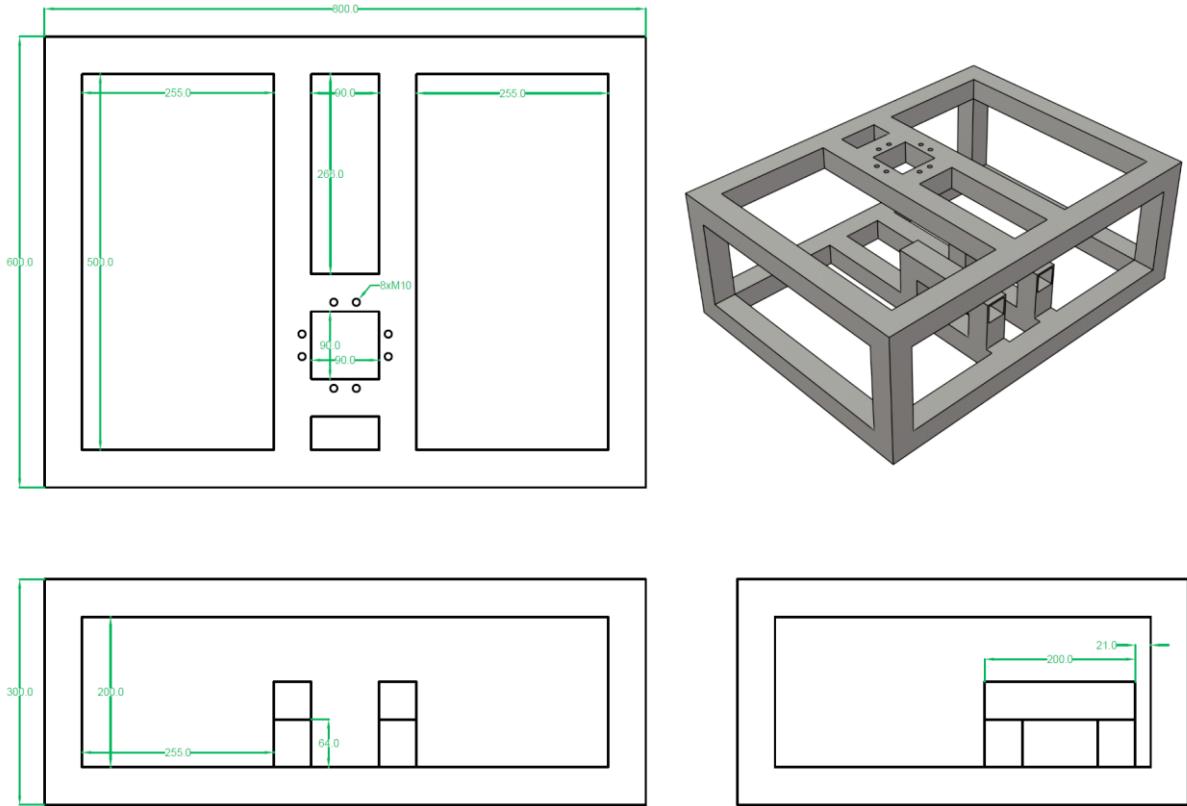
Ze względu na obciążenia statyczne oraz dynamiczne jakimi obarczona będzie konstrukcja płyty obrotowej wraz z mechanizmem obrotu, konieczne jest dopasowanie materiałów oraz wykorzystanych elementów, tak aby zapewnić odpowiednią sztywność konstrukcji. Mechanizm obrotowy będzie składał się z pięciu głównych części:

- element ruchomy, przymocowany do płyty obrotowej – rys. 4.41. a),
- element stacjonarny, przymocowany do podstawy robota – rys. 4.41. d),
- łożyska oporowe – rys. 4.41. c),
- nakrętka trzymająca łożysko – rys. 4.41. e),
- koło zębate – rys. 4.41. b).



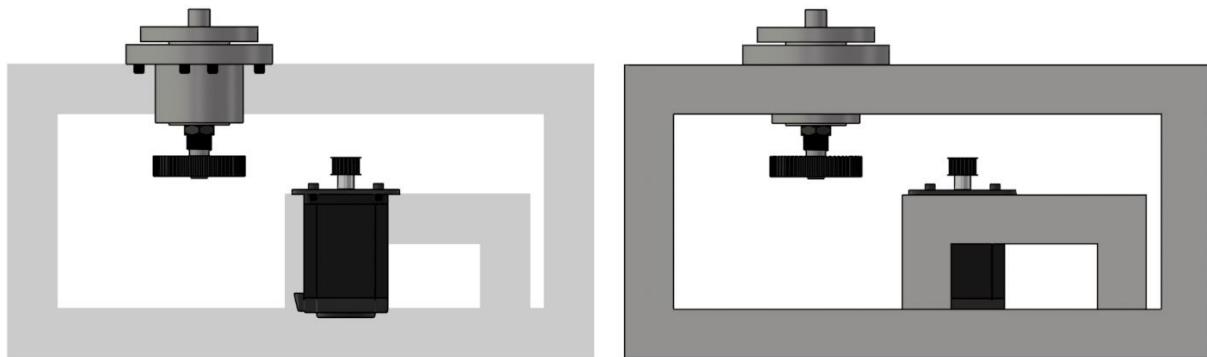
Rys. 4.41. Elementy konstrukcji sekcji obrotowej

Zespół elementów znajdujących się na rysunku 4.41. f) zostanie przymocowany od góry do płyty obrotowej, a od dołu do podstawy robota. Ze względu na znaczne gabaryty robota konstrukcja podstawy musi zapewnić odpowiednią stabilność oraz konieczne jest uwzględnienie w niej systemu mocowania silnika odpowiedzialnego za ruch względem osi z.



Rys. 4.42. Konstrukcja podstawy dla robota

Kształt podstawy został oparty o prostopadłościan wykonany z profili kwadratowych o wymiarach 50mm na 50mm oraz grubości ścianki równej 3 mm. Podstawa ta zawiera elementy niezbędne do montażu sekcji obrotowej oraz dodatkowego silnika zgodnie ze schematem umiejscowienia elementów przedstawionym na rysunku 4.43.

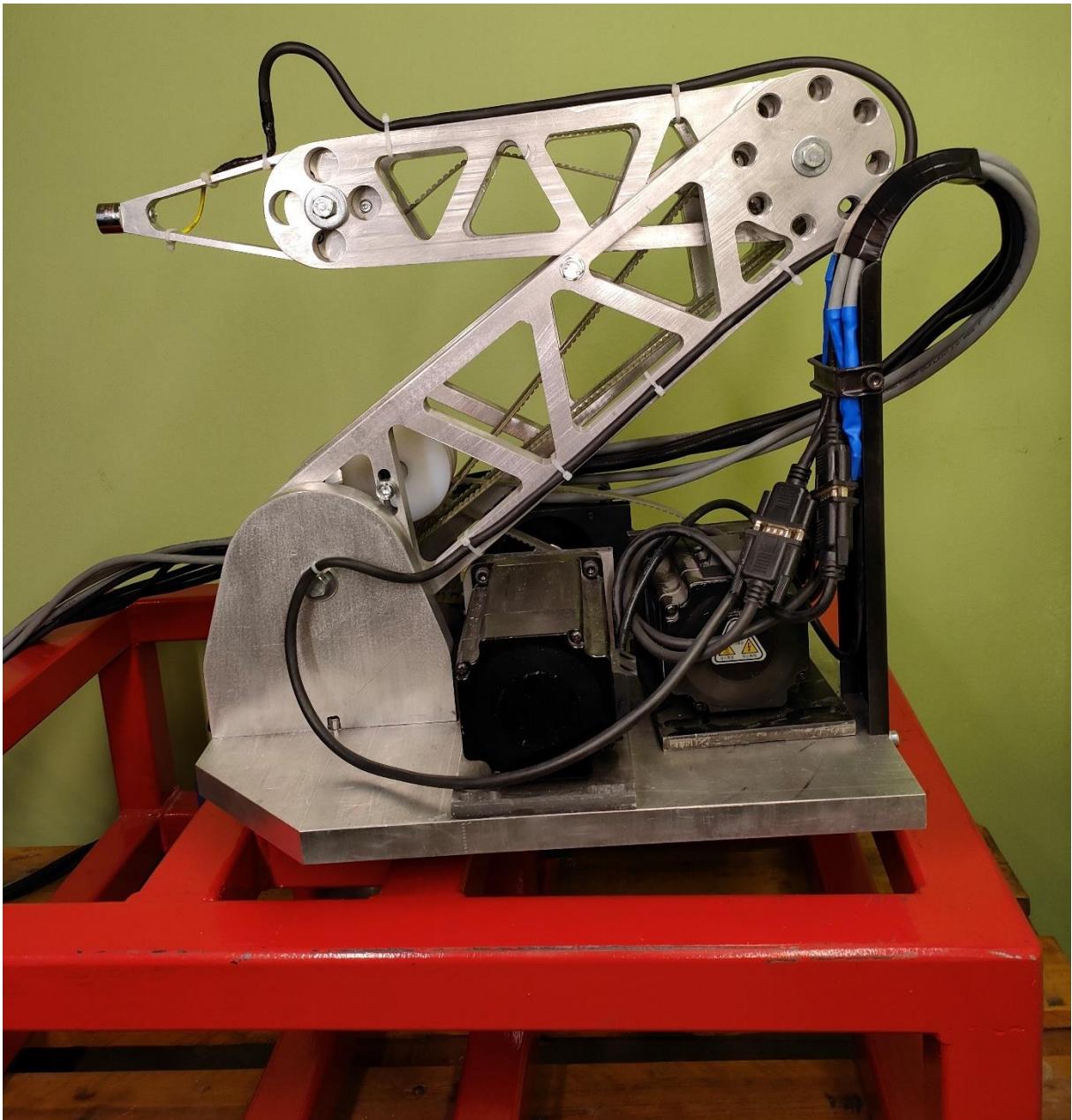


Rys. 4.43. Umiejscowienie silnika wraz z sekcją obrotową w podstawie robota

Następnym etapem prac konstrukcyjnych jest połączenie wykonanych sekcji ramion wraz z podstawą robota. Ważnym elementem jest zachowanie odpowiedniej kolejności montażu, która jest narzucona przez wykorzystanie gotowych pętli pasków zębatych.

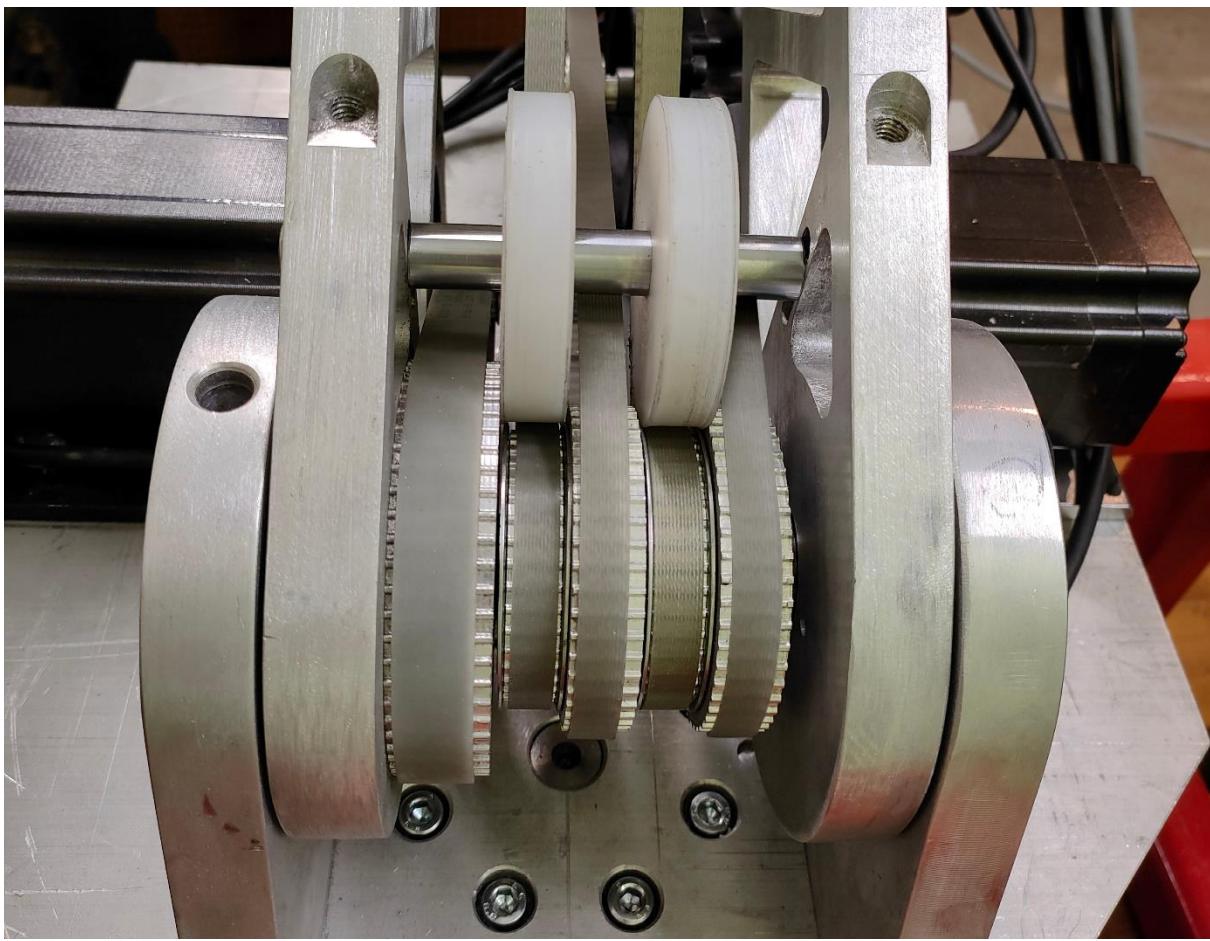


Rys. 4.44. Ukończone ramie robota

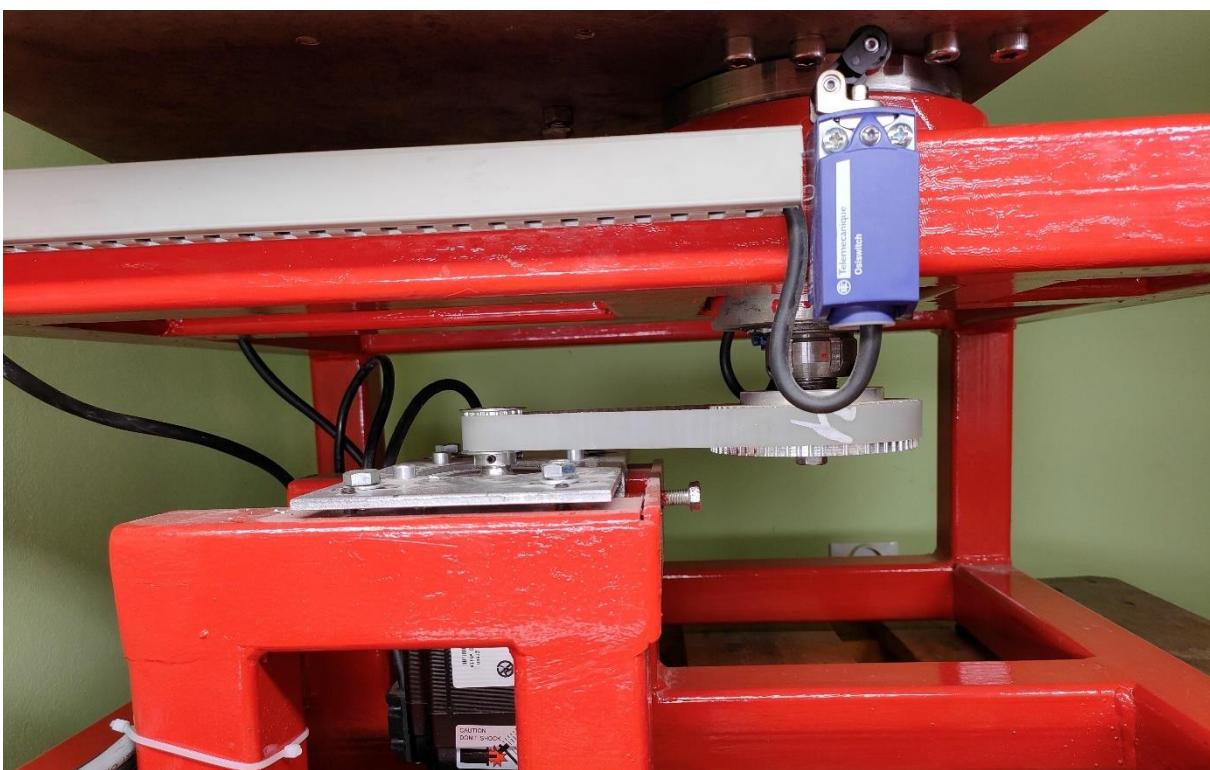


Rys. 4.44. Widok z boku robota

Jak pokazano na powyższych zdjęciach, budowa robota zakończona została powodzeniem. Istotnym elementem konstrukcji są mechanizmy napinające paski zębate łączące ze sobą poszczególne sekcje robota. Jak przedstawiono na rysunku 4.45 zostały one wykonane w postaci plastikowych walców, które skutecznie realizują zadanie naprężania pasków. W sekcji trzeciej napięcie paska zostało osiągnięte poprzez umieszczenie w odpowiednim miejscu belki ślizgowej, która zarówno usztywnia konstrukcję spajając ze sobą obie części ramienia jak i realizuje napinanie paska.



Rys. 4.45. Widok układu napinania pasków drugiej sekcji robota



Rys. 4.46. Widok napędu osi z wraz z wyłącznikami krańcowymi

Robot wyposażony jest w zestaw wyłączników krańcowych, które realizują dwa główne zadania. Pierwsze z nich służy do określenia pozycji robota po pierwotnym włączeniu układu oraz uruchomieniu procedury odnajdywania położenia. Drugim zadaniem jest ochrona przeciw kolizjom robota – w przypadku aktywacji wyłącznika krańcowego blokowany jest dalszy ruch ramienia w konkretnym kierunku co zapobiega uszkodzeniom konstrukcji. Dodatkowym wzmacnieniem są mechaniczne elementy blokujące ruch robota tak aby nie doszło do kolizji w przypadku gdyby wyłącznik krańcowy nie zadziałał.

Na platformie na której znajdują się silniki znajduje się dodatkowy element (rys. 4.44) odpowiedzialny za prowadzenie przewodów niezbędnych do działania robota. Element ten w przyszłości powinien zostać zastąpiony łańcuchowym układaczem przewodów, który pozwoli na ich lepszą organizację. W zastosowanym rozwiążaniu mogą one zaczepiać się o ewentualne przedmioty znajdujące się w bliskim otoczeniu robota.



Rys. 4.47. Widok przewodów koniecznych do sterowania robotem

Cała konstrukcja robota wraz z podstawą waży około 50kg. Na wartość tę w dużej mierze ma wpływ masa podstawy, która stanowi prawie 75% całkowitej masy robota. Ze względu na wysoką masę niemożliwe staje się swobodne manewrowanie robotem, jednak celem budowy była realizacja urządzenia stacjonarnego, a w tym przypadku duża masa zapewnia wysoką stabilność podczas każdego ruchu.

4.6. Budowa układu sterowania robotem

W celu wprawienia konstrukcji ramienia w ruch konieczne jest doprowadzenie odpowiedniego zasilania do poszczególnych silników. W tym celu konieczne jest odpowiednie połączenie oraz wysterowanie elementów dobranych w rozdziale 3.3.1. pracy. Układ sterowania zostanie umieszczony w zamkniętej obudowie. Elementami jakie będą musiały się znaleźć w skrzyni sterującej są:

- karty sterujące silnikami krokowymi (jedna karta dla jednego silnika),
- zasilacze impulsowe (jeden zasilacz dla jednej karty sterującej),
- moduł dedykowanego układu sterującego ramieniem.

Dodatkowo, na obudowie zostaną umieszczone elementy takie jak:

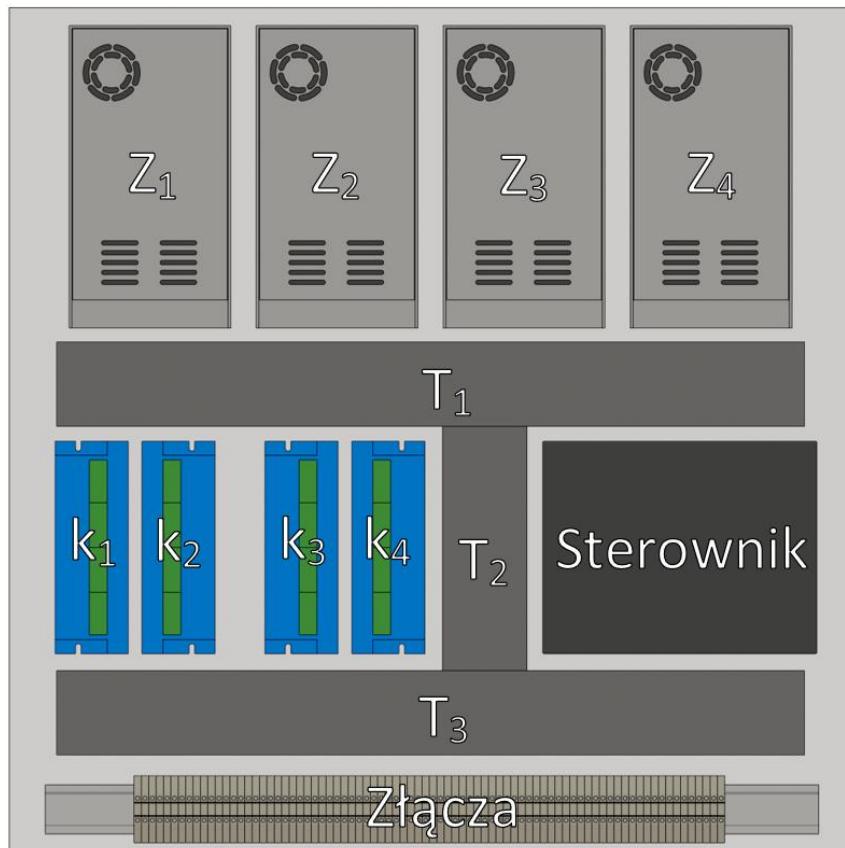
- przycisk bezpieczeństwa (wyłącznik awaryjny),
- przełącznik krzywkowy (służący do włączania i wyłączania zasilania),
- lampki kontrolne,
- gniazda do podłączania:
 - silników krokowych,
 - enkoderów,
 - dodatkowego zasilania oraz sterowania efektorem,
 - wyłączników krańcowych,
 - gniazdo USB do komunikacji układu sterującego z komputerem,
 - gniazdo do podłączenia zasilania sieciowego.

Na podstawie powyższych założeń dobrano skrzynię o konstrukcji metalowej o wymiarach 600mm x 600mm x 250mm. Skrzynia o takich rozmiarach pozwoliła na swobodne rozmieszczenie sterownika, zasilaczy oraz kart sterujących wraz z dodatkowymi kanałami kablowymi umożliwiając przy tym swobodną cyrkulację powietrza co przyczyni się do uzyskania niższych temperatur poszczególnych układów.



Rys. 4.48. Wykorzystana skrzynia sterownicza [24]

Sposób rozmieszczenia poszczególnych elementów wewnętrz obudowy zaprezentowany został na rysunku 4.49.



Rys. 4.49. Schemat ułożenia poszczególnych elementów w skrzyni sterowniczej

Legenda oznaczeń:

- Z_{1-4} – zasilacze impulsowe
- k_{1-4} – karty sterujące silnikami krokowymi
- T_{1-3} – grzebieniowe koryta kablowe
- Złącza – złączki szynowe
- Sterownik – dedykowany układ sterujący robotem

Każda z kart sterowniczych musi zostać zasilona napięciem stałym z odpowiadającego danej karcie zasilacza. Dodatkowo do każdej z kart musi zostać doprowadzony sygnał z enkodera. Każda karta musi zostać połączona ze sterowanym silnikiem. W celu kontroli działania silników konieczne jest wsterowanie każdej z kart za pomocą sterownika. W celu kalibracji ramienia oraz poprawieniu bezpieczeństwa, sterownik musi reagować na sygnały z wyłączników krańcowych (po 2 wyłączniki dla każdej sekcji ramienia). W celu umożliwienia sterowania za pomocą komputera konieczne jest wyposażenie sterownika w odpowiedni port komunikacji oraz zapewnienie stabilnego połączenia. Poza GUI aplikacji sterującej na komputerze, sterownik powinien sygnalizować stany awaryjne za pomocą lamp kontrolnych. Ostatnią kwestią jest doprowadzenie zasilania sieciowego do każdego z zasilaczy oraz do sterownika. W tym celu konieczne jest uwzględnienie przełącznika krzywkowego oraz wyłącznika bezpieczeństwa.

W celu zapewnienia pewnego rodzaju modułowości całej konstrukcji, ramię robota oraz skrzynia sterująca powinny stanowić dwie osobne części, które można łatwo połączyć za pomocą odpowiednio dobranych wtyczek oraz gniazd. Konieczne jest więc zamontowanie na obudowie szeregu gniazd w następujących ilościach:

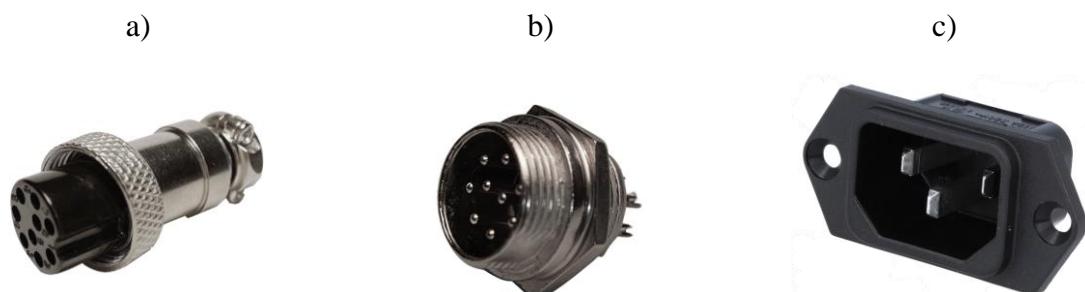
- gniazda 4 pinowe (dla każdego z silników) – 4 sztuki,
- gniazda 6 pinowe (dla enkoderów) – 4 sztuki,
- gniazda 2 pinowe (dla wyłączników krańcowych) – 8 sztuk,
- gniazdo np. 7 pinowe (do efektora) – 1 sztuka,
- gniazdo zasilające (230VAC) – 1 sztuka,
- gniazdo USB (do komunikacji z komputerem) – 1 sztuka.

Ze względu na wysokie znamionowe prądy każdego z silników wynoszące nawet 6A, konieczne jest zastosowanie gniazd o prądzie znamionowym większym od wyżej wymienionego. Do tego zastosowania dobrane zostały gniazda oraz wtyczki z dużym zapasem prądowym ($I_n = 30A$). Są to złącza firmy WEIPU o numerach SP21XX. Z tej samej serii pochodzą gniazda oraz wtyczki do enkoderów, jednak charakteryzują się one obecnością siedmiu pinów [25].



Rys. 4.50. Zestawienie gniazd oraz wtyczek wykorzystywanych do podłączenia silników (a, b) wraz z enkoderami (c, d) [25]

Do podłączenia wyłączników krańcowych zostaną wykorzystane gniazda oraz wtyki, które w przeszłości były wykorzystane przy budowie innych projektów. Są to gniazda cztero- oraz sześciu-pinowe, jednak w tym przypadku wykorzystane zostaną tylko dwa piny z każdego złącza



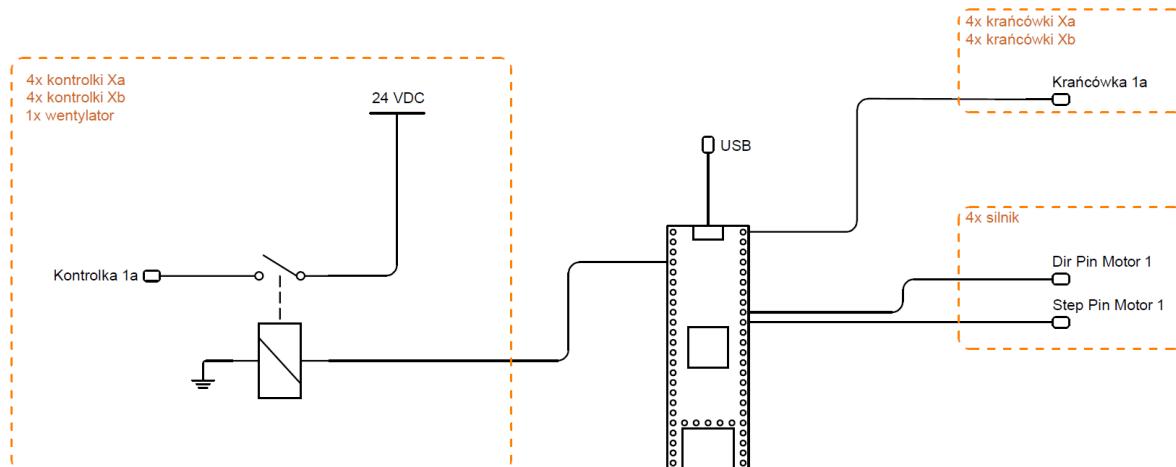
Rys. 4.51. Zestawienie gniazd a) oraz wtyczek b) wykorzystywanych do podłączenia wyłączników krańcowych wraz z gniazdem zasilającym 230VAC [26]

Skrzynia sterująca będzie zasilana jednofazowo napięciem 230V. Do tego celu wykorzystane zostanie gniazdo zgodne ze standardem IEC 60320 w klasie C14 (rysunek 4.51. c). Ze względu na konieczność przylutowania poszczególnych przewodów do gniazd zamontowanych na obudowie, dodatkowo zostały wykorzystane złączki na szynę DIN 35, które łączą przewody wychodzące z gniazd na obudowie z przewodami przyłączonymi do poszczególnych elementów w skrzynii.

Ważnym aspektem układu sterującego jest sam sterownik, który będzie układem dedykowanym do tej aplikacji. Układ ten będzie opierał się na 3 głównych układach, którymi są:

- mikrokontroler,
- zespół przekaźników,
- zasilacz.

W celu interpretacji sygnałów wysyłanych przez program sterujący (program komputerowy) oraz sterowania silnikami, konieczne jest zastosowanie odpowiedniego układu logicznego. Wybór padł na mikrokontroler o architekturze ARM Teensy 4.1. Ze względu na wcześniej wspomnianą dodatkową sygnalizację ewentualnych błędów przez skrzynię sterowniczą wymagane jest odpowiednie wysterowanie lampek kontrolnych. Ze względu na fakt, że pracują one na napięciu 24VDC, konieczne będzie podniesienie wartości napięcia z 3.3VDC (napięcie z jakim pracuje mikrokontroler) do wymaganych 24VDC. Innym sposobem jest zastosowanie przekaźników oraz wykorzystanie osobnego źródła zasilania. Ze względu na większą prostotę oraz dostępność odpowiedniego zasilacza wykorzystany został właśnie ten sposób. W jednakowy sposób zostaną również wysterowane wentylatory zamontowane na obudowie skrzyni.



Rys. 4.52. Uproszczony schemat połączeń wewnętrz sterownika, pełny schemat połączeń zamieszczono w załączniku 1

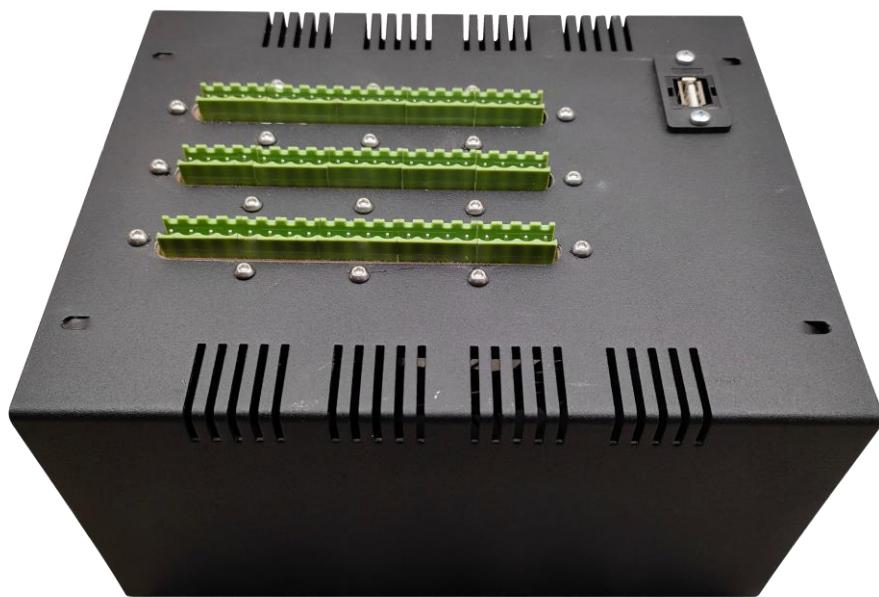
Jak pokazano na powyższym schemacie, ze sterownika będzie wychodziło wiele przewodów sterujących, co przekłada się na mnogość złącz jakie będą musiały zostać dopasowane do jego konstrukcji. W tym celu została dobrana odpowiednia obudowa, która pomieści poszczególne elementy sterownika.

Jako zasilacz został wybrany moduł marki SIEMENS opisany jako SITOP PSU100C. Jest to zasilacz stabilizowany o napięciu wyjściowym 24V oraz maksymalnym prądzie wyjściowym równym 3.7A. Jednostka ta bez problemu zapewni duży zapas mocy dla zaprojektowanego robota.



Rys. 4.53. Zasilacz stabilizowany SIEMENS SITOP PSU100C

Wykonany sterownik po zrealizowaniu niezbędnych połączeń oraz złożeniu wszystkich części pokazano na rysunku 4.54:



Rys. 4.54. Gotowy moduł sterownika

Kolejnym krokiem po skompletowaniu wszystkich niezbędnych elementów jest złożenie skrzyni sterowniczej. Efektem tych działań jest skrzynia zaprezentowana na rysunkach 4.55-4.58.



Rys. 4.55. Wnętrze skrzyni sterującej robotem.



Rys. 4.56. Front skrzyni sterującej



Rys. 4.57. Widok gniazd znajdujących się na skrzyni



Rys. 4.58. Wyłącznik główny układu sterującego robotem

Ważną kwestią jest bezpieczeństwo osób operujących robotem. Ze względu na gabaryty oraz duże masy ruchome konieczne było przystosowanie robota do wymogów bezpieczeństwa. Istnieją liczne dokumenty prawne warunkujące sposób zabezpieczenia maszyny, jaką jest robot przemysłowy. Mimo faktu, że wykonane ramię nie zostanie wykorzystane w żadnym procesie przemysłowym a jego budowa ma charakter eksperymentalny zastosowano się jednak do wytycznych zamieszczonych w dokumentach takich jak:

- dyrektywa maszynowa
- bezpieczeństwo elektryczne maszyn - norma EN60204

Analiza tych dokumentów pod kątem możliwości zastosowania opisanych tam rozwiązań w pracy określiła konieczność wykorzystania dwóch elementów bezpieczeństwa, którymi są wyłącznik główny widoczny na rys. 4.58 oraz wyłącznik awaryjny znajdujący się na froncie skrzyni elektrycznej – rys. 4.56. Dzięki tym elementom możliwe jest wyłączenie robota w sytuacji awaryjnej, co podczas testowania robota okazało się istotnym elementem.

5. Program sterujący robotem

Każdy robot przemysłowy musi być obiektem programowalnym, inaczej mówiąc, człowiek przez odpowiednią ingerencję w program robota może dowolnie wpływać na jego działanie. W tym celu roboty przemysłowe posiadają dedykowane środowiska programowe. Obsługa robota może być realizowana za pomocą panelu operatorskiego lub programu komputerowego. Panel operatorski różni się od klasycznego wykorzystywanego w przemyśle HMI (ang. *Human-Machine Interface*). Ze względu na fakt, że robot jest maszyną precyzyjną oraz jego efektor może być poruszany po całym obszarze roboczym robota, operator lub programista często musi znajdować się w pobliżu ramienia w celu sprawdzenia czy zachowanie układu (np. pozycjonowanie czy stan chwytaka) jest poprawne. Aby ułatwić użytkownikom interakcję z maszyną został wykonany mobilny panel operatorski, który łączy się ze sterownikiem urządzenia.

Przykładem typowego panelu operatorskiego jest Teach Pendant firmy ABB pokazany na rysunku 5.1. Pozwala on na interakcję z robotem bez konieczności przebywania w jednym miejscu wyznaczonym przez np. skrzynię sterowniczą robota. Teach Pendant składa się z kilku głównych elementów [27]:

- a. wyświetlacza – zawiera on wszystkie potrzebne użytkownikowi funkcje do programowania oraz wyświetla odpowiednie informacje (np. kąty poszczególnych stawów lub położenie efektora),
- b. awaryjnego wyłącznika – pozwala to operatorowi na wyłączenie robota bez konieczności przemieszczenia się w celu naciśnięcia przycisku na skrzyni sterującej,
- c. panelu z przyciskami i drążkami – dzięki niemu użytkownik może odpowiednio posługiwać się środowiskiem robota, często zamiast osobnego panelu stosuje się ekran dotykowy
- d. trzystanowego przycisku (ang. *Dead man's switch*) – jest to przycisk, który należy trzymać w ścisłe określonym położeniu. W przypadku kiedy zostanie on puszczony – interpretowane jest to jako brak operatora w pobliżu co blokuje ruchy robota, jeśli natomiast przycisk zostanie zbyt mocno wcisnięty również blokowany jest rucha stan ten jest interpretowany jako skurcz mięśni operatora wywołany np. porażeniem prądem. W celu poprawnego działania, przycisk ten należy trzymać w pozycji środkowej,
- e. portu USB – w ten sposób możliwe jest łatwe wykonanie kopii zapasowej oraz umieszczenie jej na osobnym nośniku pamięci.



Rys. 5.1. Teach Pendant firmy ABB [8]

Ze względu na mnogość rozwiązań, jakie są zastosowane w Teach Pandantach odwzorowanie działania takiego układu jest dużym wyzwaniem. Projekt budowanego ramienia zakłada stworzenie środowiska do jego obsługi. Stworzenie Panelu podobnego do umieszczonego na rysunku 5.1. jest skomplikowanym zadaniem, przez co konieczne będzie wdrożenie podobnej funkcjonalności, jednak przy wykorzystaniu innych środków. Dla wielu osób najbardziej intuicyjną formą sterowania jakimkolwiek układami jest kontroler do gier (gamepad). Jest to urządzenie stworzone z myślą o intuicyjnym użytkowaniu. Dzięki niemu możliwe jest spełnienie trzech z pięciu założeń funkcjonalności Teach Pendant, w szczególności:

- a) gamepad posiada 2 drążki analogowe. Ze względu na fakt, że każdy drążek analogowy posiada dwa kierunki ruchu, można je wykorzystać do sterowania poszczególnymi osiami ramienia lub do sterowania każdą współrzędną efektora z osobna. Pozostałe przyciski znajdujące się na kontrolerze mogą służyć do wyzwalania poszczególnych funkcji,
- b) jeden z przycisków może posłużyć za przycisk awaryjnego wyłączenia,
- c) dodatkowo, gamepady posiadają dwa osobne przyciski analogowe, które mogą posłużyć jako zamiennik dead man's switcha.



Rys. 5.2. Wykorzystany gamepad [28]

Aby dodać pozostałą funkcjonalność tj. wyświetlacz (port USB do zapisu kopii zapasowej nie jest niezbędny w tym projekcie), można wykorzystać w tym celu osobne urządzenie jak np. laptop, tablet czy smartfon. Ze względu na większą uniwersalność, w tym celu został wykorzystany komputer. Dodatkowo, na komputerze będzie działał program będący głównym elementem sterującym całym robotem. Dzięki wykorzystaniu komputera jako bazy dla programu nie będzie również konieczne łączenie gamepada z mikrokontrolerem, co wymagałoby napisania własnego sterownika. Przyjęte wcześniej podejście zakładające komunikację poprzez port USB sterownika (mikrokontrolera) z komputerem będzie mogło zostać zrealizowane jako komunikacja szeregowa poprzez port szeregowy komputera, co bardzo ułatwi połączenie. System operacyjny który znajduje się na komputerze to Windows 10. Ze względu na przyjęte podejście w kodzie, aplikacja ta nie będzie prawidłowo funkcjonowała na komputerach z innymi systemami.

5.1. Omówienie aplikacji komputerowej do sterowania robotem

Do napisania programu sterującego ramieniem wykorzystany został wysokopoziomowy język C#. Wybór padł na ten język programowania ze względu na bogate w funkcje biblioteki do obsługi stringów – zmiennych typu tekstowego. Ma to duże znaczenie ze względu na konieczność odpowiedniej konstrukcji oraz ciągłego przetwarzania wielu komunikatów wymienianych między sobą przez program komputerowy oraz mikrokontroler. Ze względu na dobór języka wybrane zostało również odpowiednie środowisko programistyczne którym jest *Visual Studio*.

Przyjęte zostały 3 główne założenia jakie ma spełniać program sterujący robotem.

- a. umożliwienie użytkownikowi wykonania połączenia z mikrokontrolerem,
- b. możliwość sterowania ramieniem za pomocą wcześniej przyjętego podejścia – wykorzystania gamepada w celu kontroli ruchów ramienia,
- c. zapewnienie pewnego rodzaju automatyzacji ruchów ramienia. Zostanie to zrealizowane poprzez stworzenie prostego środowiska programistycznego wewnętrz opisywanej aplikacji, które pozwoli zaprogramować ramię w taki sposób aby poruszało się prostym ruchem od punktu A do punktu B, które to punkty zostaną wcześniej zadeklarowane przez użytkownika.

Implementacja komunikacji szeregowej

Realizacja pierwszego postulatu opiera się o wykorzystanie klasy *SerialPort*, która jest dostępna w *Visual Studio*. Klasa ta zawiera metody konieczne do nawiązania połączenia szeregowego z urządzeniem zewnętrznym. Dodatkowo, dostępne są odpowiednie metody do odbierania oraz nadawania komunikatów. Komunikaty wysłane z programu komputerowego będą zawierały odpowiednie dane określające bieżące pozycje w jakich mają ustawić się silniki. Przekłada się to na konieczność wysłania czterech różnych informacji o docelowym kącie w jakim powinien się znaleźć silnik. Dodatkowo, potrzebne jest przesłanie ewentualnych komend do mikrokontrolera, które będą stanowiły wyjątek poza przesyłanymi standardowo komendami pozycjonującymi ramię robota. W celu uproszczenia komunikacji zastosowane zostało podejście, w którym poszczególne koordynaty dla każdego z silników zostaną umieszczone w jednym ciągu tekstowym. Ciąg ten dodatkowo zostanie poszerzony o potencjalną informację, stanowiącą wyjątek w ciągłej komunikacji. Ciąg tekstowy, jaki zostanie wysłany do mikrokontrolera będzie w podanej niżej formie:

„a**POS0**b**POS1**c**POS2**d**POS3**e**INSTRUCTION**x”

Gdzie:

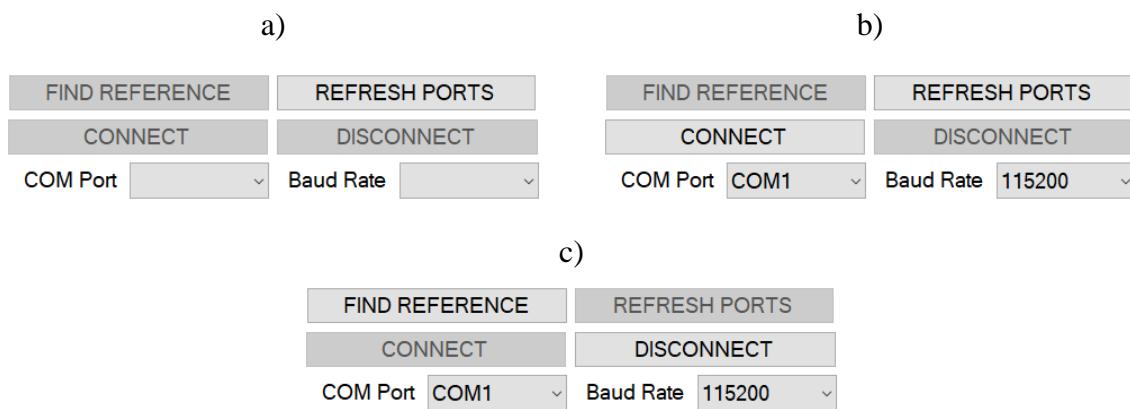
- a. POS0, POS1, POS2, POS3 – docelowe pozycje dla każdego silnika
- b. INSTRUCTION – dodatkowa instrukcja dla mikrokontrolera

W celu rozróżnienia poszczególnych liczb zastosowane zostało podejście, w którym każda liczba oddzielona jest osobnym znakiem od „a” do „e”. Po znaku e występuje ewentualna instrukcja, a cały ciąg tekstowy kończy się znakiem „x”. Takie podejście pozwala na

wykorzystanie metody *ReadUntil()* która odczytuje ciąg znaków z portu szeregowego do momentu odczytania określonego znaku, który w tym przypadku został znak „x”.

Aby nawiązać połączenie poprzez port szeregowy konieczne jest dobranie jednakowych parametrów komunikacji zarówno po stronie programu komputerowego oraz po stronie mikrokontrolera. Ze względu na brak możliwości prostej zmiany parametrów komunikacji po stronie mikrokontrolera konieczne jest zapewnienie wyboru wszystkich odpowiednich ustawień z poziomu programu komputerowego. Najważniejszym parametrem jest prędkość transmisji (ang. *BaudRate*) – w przypadku połączenia poprzez port szeregowy, które opierają się o komunikację asynchroniczną oba urządzenia muszą znać prędkość z jaką są wysyłane poszczególne bity, w przeciwnym przypadku transmisja nie będzie przebiegała poprawnie. Innym bardzo ważnym aspektem jest poprawny wybór portu do jakiego został podłączony mikrokontroler. Ze względu na fakt, że mikrokontroler został podłączony z komputerem fizycznie poprzez kabel USB, po stronie mikrokontrolera numer portu jest z góry narzucony ze względu na dostępność wyłącznie jednego portu poprzez złącze USB. Z perspektywy programu komputerowego, konieczne jest wybranie odpowiedniego portu spośród dostępnych. Proces identyfikacji portu, przez który przebiegała będzie komunikacja szeregowa będzie musiał zostać przeprowadzony samodzielnie przez użytkownika. Jest to możliwe do odczytania za pomocą „Menadżera urządzeń”, który jest dostępny poprzez ustawienia systemu Windows. Do odczytu dostępnych portów zostanie wykorzystana metoda *GetPortNames()*. Odczytane porty zostaną wyświetlane w postaci listy, spośród której użytkownik wybierze odpowiedni port.

W celu zapewnienia odpowiedniej prędkości transmisji, wartość *BaudRate* w mikrokontrolerze została ustawiona na 115200 bps. Jednak w celu umożliwienia dobrania innej prędkości transmisji użytkownik będzie mógł wybrać spośród trzech najczęściej wykorzystywanych którymi są 9600, 57000 oraz 115200. Ze względu na zapewnienie stabilnej pracy urządzeń konieczne jest umożliwienie wykonania połączenia oraz jego przerwania pomiędzy każdym z urządzeń co zapewnione jest dzięki odpowiednim przyciskom, które otwierają oraz kończą transmisję szeregową.



Rys. 5.3. Część aplikacji odpowiedzialna za ustanowienie komunikacji szeregowej

W celu zapobiegnięcia przypadkowym oraz niepożądany połączeniom, przycisk odpowiedzialny za nawiązanie połączenia wraz z przyciskiem kończącym komunikację będą niedostępne do czasu wybrania odpowiednich parametrów komunikacji Rys 4.3. a). Dodatkowo, przycisk kończący komunikację będzie dostępny dla użytkownika dopiero po

nawiązaniu komunikacji Rys 4.3. c). Takie podejście zapewni większą stabilność programu chroniąc użytkownika przed przypadkowymi wciśnięciami przycisków. W celu możliwości podłączenia się do mikrokontrolera połączonego z komputerem kablem USB po uruchomieniu programu została dodana możliwość odświeżenia listy dostępnych portów szeregowych. Wraz z odświeżeniem zachodzi konieczność ponownego wybrania portu. Po nawiązaniu połączenia funkcjonalność ta nie będzie dostępna.

W chwili, kiedy zostanie nawiązane stabilne połączenie rozpoczęta zostanie transmisja. Sposób, w jaki zostanie stworzony ciąg znaków zawierający odpowiednie informacje zostanie utworzony na drodze konkatenacji poszczególnych zmiennych przechowujących informację o docelowym (w danej chwili czasowej) kącie obrotu poszczególnych silników zamienionych z typu zmiennoprzecinkowego na typ tekstowy. Dzięki wysokiemu poziomowi języka C# operacja ta sprowadza się do użycia metody *ToString()*, która przyjmie argument określający precyzję rzutowania z typu *double* na typ *string*. Ze względu na ograniczoną rozdzielcość ruchu wykorzystanych napędów, w tym celu wystarczająca będzie precyzja na poziomie dwóch liczb po przecinku. Budowa stosownego komunikatu wewnętrz programu została zrealizowana w następujący sposób:

```
string komunikat = "a" + Kin.kat0.ToString("0.00") +
                    "b" + Kin.kat1.ToString("0.00") +
                    "c" + Kin.kat2.ToString("0.00") +
                    "d" + Kin.kat3.ToString("0.00") +
                    "e" + instruction_for_uc + "x";
```

Wyjaśnienie pochodzenia poszczególnych sekcji tworzących całościowy komunikat należy rozpocząć od faktu, że *Kin* jest to obiekt stworzonej klasy *kinematyka*, która jest odpowiedzialna za przechowywanie stosownych zmiennych oraz wykonywanie obliczeń potrzebnych do obsługi algorytmu kinematyki odwrotnej. Fragment *kat0* (jak i również *kat1*, *kat2* oraz *kat3*) odnosi się do zmiennej klasy *kinematyka*. Część *ToString("0.00")* jest metodą, która zamienia zmienne typu *double* na zmienne typu *string* z dokładnością do dwóch miejsc po przecinku. Część *instruction_for uc* jest to nazwa zmiennej typu *string*, która przechowuje aktualną instrukcję dla mikrokontrolera.

Stworzony powyżej komunikat zostanie wysłany do mikrokontrolera. W odpowiedzi mikrokontroler przekaże do programu komputerowego osobny komunikat. Do jego odczytu również zostało przyjęte podejście, w którym każdy ciąg znaków kończy się znakiem specjalnym (również jest to „x”).

Implementacja sterowania za pomocą gamepada

Powyższy opis spełnia pierwsze założenie dotyczące kształtu aplikacji. Drugą, równie ważną cechą programu ma być możliwość sterowania ramieniem za pomocą gamepada. W tym celu konieczne jest wykorzystanie przestrzeni nazw *Windows.gaming.input* która pozwala na wykorzystanie odpowiednich metod oraz zmiennych zawartych w klasie *gamepad*. Klasa ta pozwala na obsługę kontrolerów zgodnych z gamepadami Xbox.

Odczyt poszczególnych wartości z każdego z drążków analogowych oraz odczyt aktualnie wciśniętych przycisków zrealizowany został w oparciu o metodę *GetCurrentReading* pochodząącą z klasy *gamepad*. Wychylenia drążków są reprezentowane za pomocą liczb zmiennoprzecinkowej z zakresu od -1 do 1. W sytuacji kiedy użytkownik nie porusza drążkami poszczególne odczyty nie zawsze wskazują wartości zera. Jest to związane z wysoką

rozdzielczością przetworników ADC wykorzystanych do odczytu wartości napięcia z potencjometru, który został wykorzystany jako element zamieniający wychylenie drążka na zmienne napięcie. Potencjometry te nie zawsze w pozycji stabilnej (przy braku wychylenia) znajdują się w miejscu, w którym odczytane napięcie będzie oznaczało zerowe wychylenie. Jest to związane z fizycznymi parametrami potencjometrów takimi jak ich dokładność wykonania oraz czynnikami zewnętrznymi jak np. temperatura pracy. W celu uniknięcia tego zachowania, wykorzystane zostało podejście w którym wartość odczytana z gamepada zostaje zamieniona za pomocą funkcji *Recalculate_Readings()*, która przyjmuje jako argument odpowiednią liczbę z zakresu od -1 do 1, mnoży tą wartość razy 10, a następnie rzutuje ją na typ całkowity (int). Dzięki takim zabiegom, w zależności od wychylenia mamy do czynienia z liczbą z zakresu od -10 do 10. Wpływ wcześniej opisanej niezerowej wartości w stanie stabilnym jest na tyle mały, że przy przejściu na liczby całkowite nie ma on znaczenia.

Ze względu na konieczność obsługi czterech współrzędnych określających położenie punktu wodzącego ramię, konieczne jest wykorzystanie czterech osobnych kanałów. Możliwe jest do dzięki temu, że każdy z dwóch dostępnych drążków obsługuje dwa kanały. Przyjęta metoda sterowania za pomocą tych drążków opiera się na zwiększeniu lub zmniejszeniu wartości zmiennej przechowującą aktualną koordynatę o odpowiednią wartość zależącą od wychylenia drążka.

```
x_move = Convert.ToDouble(Recalculate_Readings(Reading.RightThumbstickX)) / 20.0;
y_move = Convert.ToDouble(Recalculate_Readings(Reading.RightThumbstickY)) / 20.0;
z_move = Convert.ToDouble(Recalculate_Readings(Reading.LeftThumbstickY)) / 20.0;
epsilon_move = Convert.ToDouble(Recalculate_Readings(Reading.LeftThumbstickX)) / 20.0;

int gamepad_condition = Recalculate_Readings(Reading.LeftTrigger);
if (gamepad_condition > 0 && gamepad_condition < 10)
{
    if (Kin.point_in_range_check(Kin.X + x_move, Kin.Y + y_move, Kin.Z +
        z_move, Kin.epsilon + epsilon_move))
    {
        Kin.X += x_move;
        Kin.Y += y_move;
        Kin.Z += z_move;
        Kin.epsilon += epsilon_move;
    }
}
```

Powyższy fragment kodu realizuje opisane wyżej zadanie. Pierwsza część kodu realizuje odczyt wartości wychyleń z drążków analogowych. Odpowiednie dane odczytane ze zmiennej *Reading* są następnie przeliczane z zakresu od -1 do 1 w formie zmienoprzecinkowej na liczby całkowite z zakresu od -10 do 10 przy pomocy funkcji *Recalculate_Readings*. Wartości te są następnie z powrotem konwertowane na typ zmienoprzecinkowy za pomocą metody *ToDouble* klasy *Convert*. Jednak ze względu na fakt, że współrzędne zapisywane są w cm, należy ograniczyć prędkość ich zmian. Można to uzyskać poprzez ograniczenie wartości o jaką zmienione zostaną poszczególne współrzędne przy pojedynczym cyklu timera. Timer ten jest odpowiedzialny za cykliczne wywołania odczytów pochodzących z poszczególnych przycisków oraz drążków analogowych. Częstotliwość pracy timera została ustawiona na 20Hz. Maksymalna zmiana wartości pojedynczej współrzędnej została ustawiona na 0.5cm. Zostało to osiągnięte poprzez podzielenie wartości z zakresu od -10 do 10 przez 20.

Aby było możliwe wykonanie ruchu za pomocą drążków analogowych, konieczne jest wcisnięcie dodatkowego przycisku analogowego, którego zadaniem będzie działania w formie dead man's switcha. Możliwość sterowania ramieniem zostanie przyznana wyłącznie, jeśli przycisk ten znajdzie się w pozycji środkowej – nie może zostać maksymalnie wcisnięty ani

pozostać zwolnionym. W celu dodatkowego zezwolenia na sterowaniem położeniem punktu wodzącego za pomocą gamepada, użytkownik musi samodzielnie zaznaczyć check-boxa który aktywuje opisaną funkcjonalność. Ze względu na ograniczony zasięg roboczy ramienia, konieczne jest ograniczenie możliwości sterowania tak, aby nie wykroczyć poza dopuszczalny zakres. W tym celu wykorzystana została metoda `point_in_range_check` klasy `Kin`, która realizuje algorytm sprawdzający czy nowy punkt, w którym użytkownik chce umieścić ramię znajduje się w jego zakresie ruchowym. Realizowane jest to za pomocą sprawdzenia, czy dane podawane jako argumenty dla funkcji \cos^{-1} (równanie 5.5 oraz 5.6) należą do dziedziny tej funkcji. W przypadku spełnienia tego warunku następuje zmiana wartości współrzędnych punktu wodzącego ramieniem na nowe.

```
switch (Reading.Buttons)
{
    case GamepadButtons.A:
        if (instruction_for_uC == "0" && is_gamepad_enable)
        {
            instruction_for_uC = "TON";
        }
        break;
    case GamepadButtons.B:
        if (instruction_for_uC != "find_position" && is_gamepad_enable)
        {
            instruction_for_uC = "TOFF";
        }
        break;
}
```

Powyższy fragment kodu realizuje obsługę przycisków znajdujących się na gamepadzie. W tym przypadku pokazana jest obsługa przycisku *A* oraz *B* ze względu na ich pełnione role. W chwili przyciśnięcia przycisku *A* wysyłana jest do mikrokontrolera informacja, o tym, że ma zostać włączone dodatkowe narzędzie (realizowane jest to za pomocą podania stanu wysokiego na odpowiedni pin narzędzia). Jest to jednak możliwe wyłącznie w sytuacji, kiedy użytkownik zezwolił na sterowanie ramieniem za pomocą gamepada, oraz kiedy nie jest wysyłana, żadna inna instrukcja. W chwili wcisnięcia przycisku *B*, wysyłana jest informacja o wyłączeniu narzędzia. Instrukcja ta jest nadzorowana względem instrukcji włączenia oraz jeśli użytkownik jednocześnie wcisnie przyciski *A* i *B*, wówczas sterownik otrzyma informację o wyłączeniu narzędzia. Wykorzystany gamepad posiada większą liczbę przycisków, jednak nie zostały one przypisane do realizowania żadnego zadania. Tworzy to możliwość przyszłego rozwijania oprogramowania robota o kolejne funkcjonalności.

Implementacja dedykowanego środowiska do programowania robota

Aby użytkownik mógł swobodnie programować ruchy ramienia konieczne jest ujednolicenie systemu wprowadzania poszczególnych komend sterujących do pamięci układu. W tym celu zostało przyjęte podejście w którym użytkownik będzie wprowadzał poszczególne komendy w formie tekstowej. Do tego celu został specjalnie zaprojektowany język programowania, który zawiera funkcje potrzebne do podstawowego poruszania się robotem. Podstawowym rodzajem ruchu jaki zostanie zaimplementowany jest ruch prostoliniowy. Dodatkowo ważną funkcją będzie możliwość wykonania przerwy pomiędzy ruchami. Zaimplementowana zostanie również możliwość regulacji prędkości oraz możliwość jednobitowego sterowania dodatkowym narzędziem

Ważnym aspektem podczas wykonywania programu jest kolejność wykonywanych funkcji. Muszą być one realizowane jedna po drugiej w kolejności z jaką zostały zapisane w edytorze tekstowym.

Tu użytkownik wprowadzi swój kod

Rys. 5.4. Wygląd edytora tekstowego do programowania robota

Pierwszą ważną funkcją jest funkcja ruchu. Sposób kodowania został zainspirowany profesjonalnym językiem służącym do programowania robotów przemysłowych MELFA-BASIC V. Język ten został wprowadzony przez firmę MITSUBISHI ELECTRIC. W składni tego języka można znaleźć komendę odpowiedzialną za realizację ruchu prostoliniowego wywoływaną za pomocą instrukcji MVS. Przykład wykorzystania tej funkcji:

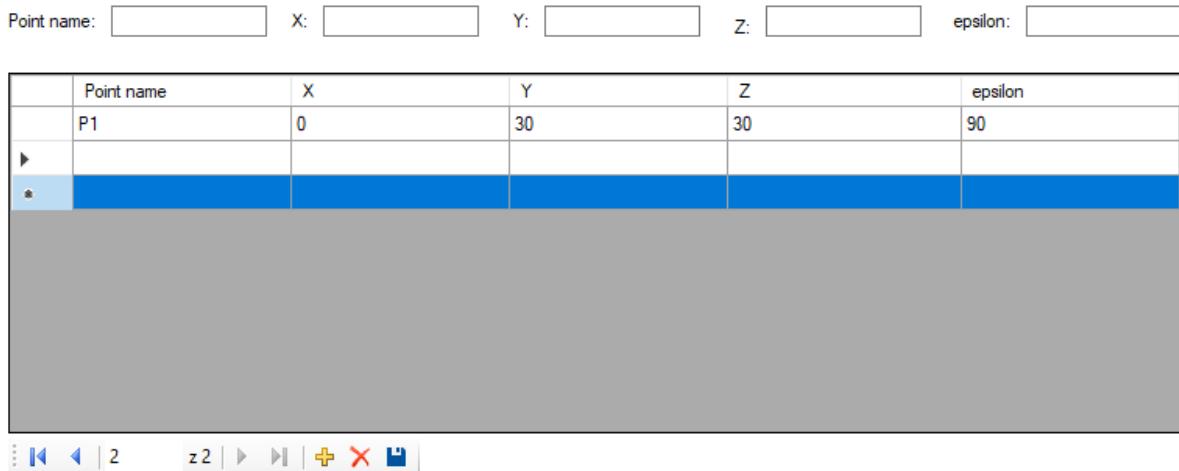
MVS P1

W przykładzie tym część zaznaczona kolorem pomarańczowym wywołuje funkcję ruchu prostoliniowego, natomiast część zaznaczona kolorem niebieskim jest to argument funkcji w postaci nazwy punktu, pod którą kryją się docelowe współrzędne tego punktu [9]. Robot wykonujący funkcję MVS porusza się ruchem prostoliniowym od punktu w którym aktualnie się znajduje do punktu który został podany jako argument – w tym przypadku jest to P1.

Podejście wykorzystane w programie opiera się o podobny model funkcji. Również ruch prostoliniowy wywoływany będzie funkcją MVS, jednak sposób wprowadzania argumentu będzie się różnił. W celu wprowadzenia argumentu musimy wprowadzić go pomiędzy nawiasami, natomiast całą linię zakończyć średnikiem. Przykład wykorzystania tej funkcji:

MVS(P1);

Ze względu na sposób podawania argumentów w funkcji MVS w postaci ich samej nazwy, konieczne jest stworzenie miejsca, które będzie przechowywało współrzędne punktów powiązane z ich nazwą. W tym celu została stworzona prosta baza danych stanowiąca bazę punktów, które użytkownik może wprowadzać oraz dowolnie edytować. Obliczenia kinematyki ramienia wymagają czterech współrzędnych, które pozwolą jednoznacznie określić kąty w jakich mają ustawić się silniki. Tak więc baza punktów musi zawierać 5 pól w które użytkownik będzie mógł wprowadzić nazwę punktu, który deklaruje oraz jego cztery współrzędne.



Rys. 5.5. Baza punktów

W celu wprowadzenia nowego punktu do bazy, użytkownik musi nacisnąć żółty przycisk „**+**” znajdujący się na dole rysunku 4.5.. Następnie możliwe jest wprowadzenie danych w każde z pięciu pól znajdujących się ponad tabelą. W celu zapiania wprowadzonych danych, konieczne jest wcisnięcie przycisku dyskietki. Chcąc usunąć punkt z bazy wystarczy zaznaczyć go oraz wcisnąć czerwony przycisk „**X**”. Takie podejście sprawia, że kod wprowadzany przez użytkownika staje się czytelny, co przekłada się na większą łatwość programowania.

Dodatkowo została wprowadzona możliwość dodania punktu bazując na aktualnym położeniu ramienia. Jest to szczególnie przydatne, kiedy użytkownik chce nakierować robota w odpowiednie miejsce za pomocą gamepada, po czym przechwycić daną pozycję. W ten sposób możliwe jest stworzenie całej trajektorii ruchu bazując na ręcznym pozycjonowaniu, a następnie stworzenie odpowiedniego programu bazując na uprzednio zapisanych punktach.

Drugą ważną funkcją jest prosta funkcja typu delay. Stosując tą funkcję układ podczas jej wykonywania powinien być w spoczynku. Funkcja ta jako argument przyjmuje wartość określającą czas lub liczbę cykli przez których okres trwania układ będzie nieruchomy. W tym przypadku również inspiracją jest język MELFA BASIC V, w którym tą funkcję można wykorzystać w następujący sposób:

Dly 0.5

Powyższy zapis oznacza, że czas trwania funkcji wyniesie 0.5 sekundy. Funkcja delay zaimplementowana w programie będzie różniła się pod względem składni oraz pod względem sposobu podawania argumentu. Składnia będzie dopasowana do wcześniej opisywanej funkcji MVS, co oznacza, że konieczne będzie wprowadzenie argumentu poprzez umieszczenie go w nawiasach. Okres trwania funkcji będzie podawany jednak w milisekundach. Dzięki takiemu podejściu możliwe będzie precyzyjniejsze określanie czasu trwania funkcji. Zapis będący jednoznaczny z *Dly 0.5* przyjmie postać:

DLY(500);

Kolejną funkcją jaka musi zostać zaimplementowana jest funkcja umożliwiająca zmianę prędkości z jaką porusza się punkt wodzący ramienia. W języku MELFA BASIC V regulacja prędkości jest możliwa za pomocą funkcji *Ovrd*. Prędkość ustalamy poprzez podanie odpowiedniej wartości w procentach, gdzie 100 oznacza prędkość maksymalną. Przykładowe wykorzystanie tej komendy:

Ovrd 65

W tym przypadku również konieczna jest zmiana składni, na dopasowaną do dwóch wcześniej opisywanych funkcji. Wartość prędkości również będzie zmieniana z zakresu od 1 do 100%, natomiast funkcja przyjmie bardziej intuicyjną nazwę – SPEED. Przykład wykorzystania odpowiadający zapisowi *Ovrd 65*:

SPEED(65);

Ostatnią ważną funkcją jest komenda umożliwiająca sterowanie narzędziem przymocowanym do efektora. Sterowanie to zostało zrealizowane jako jednobitowe, oznacza to, że użytkownik ma możliwość sterowania wyłącznie dwoma stanami narzędzia (włączone – ON oraz wyłączone – OFF). Komenda odpowiedzialna za realizację włączenia prezentuje się następująco:

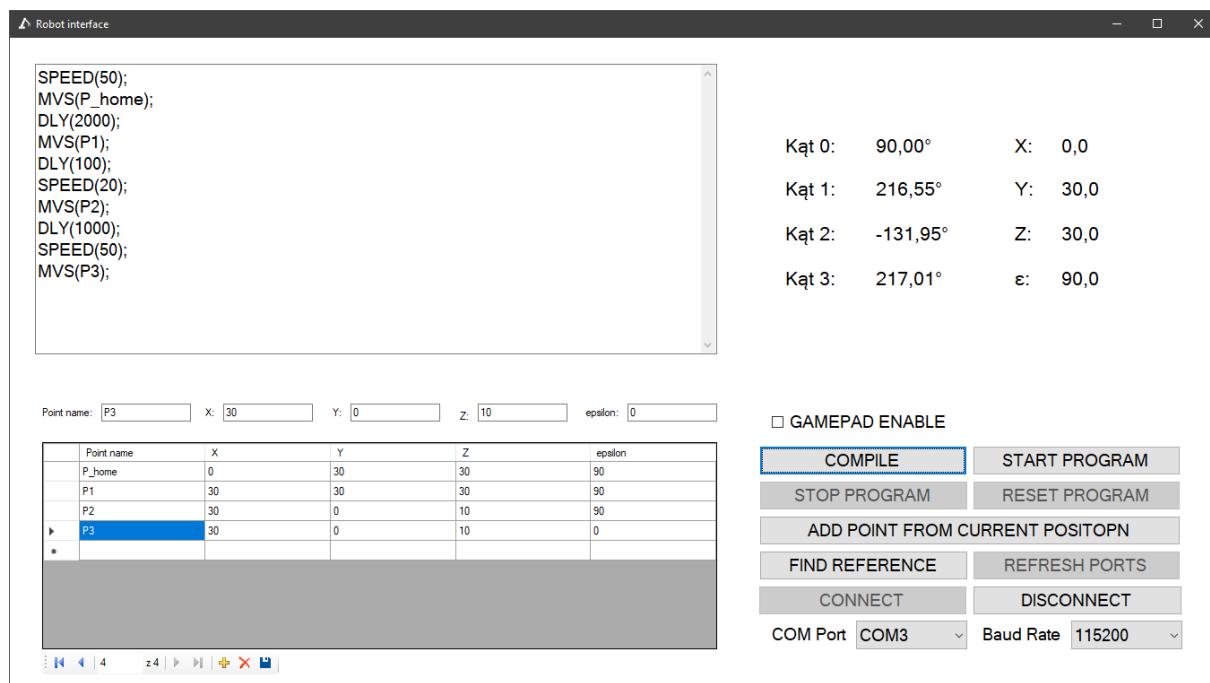
TOOL(ON);

Chcąc wyłączyć narzędzie, wystarczy zmienić argument na OFF. Takie podejście wyklucza zastosowanie układów opartych o bardziej skomplikowaną logikę, jednak umożliwia wykorzystanie różnego rodzaju chwytaków, gdzie stan załączenia może oznaczać zamknięcie uchwytu, a stan wyłącznie jego otworzenie. Podobna zasada może zostać wprowadzona poprzez wykorzystanie elektromagnesu jako narzędzia. Wówczas załączając narzędzie, włączony zostanie elektromagnes oraz nastąpi analogiczne działanie dla wyłączenia narzędzia.

W przypadku języka MELFA BASIC V realizacja zamknięcia lub otwarcia chwytaka może zostać zrealizowana za pomocą komendy:

Hopen 1

Zamknięcie chwytaka realizowane jest za pomocą podobnej komendy, jednak człon *Hopen* należy zamielić na człon *Hclose*. Liczba obok komendy oznacza numer chwytaka.



Rys. 5.6. GUI programu sterującego robotem

Jak widać na powyższym rysunku, wcześniej opisywane pojedyncze sekcje programu pozwalają na stworzenie podstawowej aplikacji pozwalającej na sterowanie robotem oraz

jego proste programowanie. Poza opisanymi wcześniej elementami widoczna jest część która zawiera informacje na temat aktualnego położenia punktu wodzącego ramię oraz każdy z kątów o jaki obrócony jest wirnik każdego z silników. W sekcji aplikacji w której znajdują się przyciski można wyróżnić wcześniej opisane przyciski do nawiązywania połączenia z mikrokontrolerem. Dodatkowo znajdują się tam przyciski odpowiedzialne za obsługę programu sterującego robotem.

W celu sprawdzenia poprawności działania napisanego programu należy wcisnąć przycisk *COMPILE*, który wywoła odpowiednie funkcje sprawdzające czy kod został napisany w poprawny sposób. Realizowane są wówczas następujące zadania:

- a) odczytanie ilości punktów znajdujących się w bazie punktów,
- b) odczytanie kodu zamieszczonego w polu tekstowym,
- c) sprawdzenie czy każda linijka kończy się średnikiem,
- d) sprawdzenie czy każda funkcja została zapisana w poprawny sposób,
- e) wyodrębnienie wartości argumentu funkcji,
- f) sprawdzenie czy każdy wykorzystany punkt znajduje się w bazie punktów
- g) sprawdzenie czy każdy punkt znajduje się w zasięgu ramienia
- h) przypisanie aktualnej pozycji robota jako pozycji startowej dla funkcji MVS.

Realizacja podpunktu a) została przeprowadzona poprzez odczytanie wartości przechowywanej w zmiennej *Rows.Count* w obiekcie jakim jest baza danych.

```
int points_number = pointsDataGridView.Rows.Count - 1;
```

Ze względu na fakt, że ilość wierszy znajdujących się w wyświetlonej tabeli jest liczona w sposób gdzie pierwszy wiersz (zawierający informacje co znajduje się w danej kolumnie) ma numer 0, a wiersze które są ważne dla działania aplikacji zaczynają się od numeru 1, potrzebna była modyfikacja wykonana poprzez pomniejszenie odczytanej wartości o 1.

Podpunkt b) został zrealizowany za pomocą poprzez odczytanie zawartości ze zmiennej *Text* obiektu którym jest pole tekstowe.

```
string code_Input = codeInputTextBox.Text;
```

Aby odczytać poszczególne linijki kodu konieczna jest zamiana pojedynczego ciągu tekstopowego zawierające poszczególne instrukcje na pojedyncze ciągi znaków, gdzie każdy ciąg to osobna instrukcja. Zostało to zrealizowane za pomocą poniższego fragmentu kodu:

```
string[] separators = new string[] { "\r\n" };
string[] single_Line_Code = code_Input.Split(separators, StringSplitOptions.None);
```

Za pomocą metody Split możliwe jest podzielenie pojedynczego ciągu znaków. Metoda ta za argument przyjmuje separatory, za które uznany został znak nowej linii, którym potocznie mówiąc jest *ENTER*. Windows jako znak nowej linii uznaje串 znaków w postaci \r\n.

Podpunkt c) został zrealizowany za pomocą pętli for, która przeszukuje każdą z odczytanych linii, oraz sprawdza czy znajduje się w niej średnik. W przeciwnym wypadku wyświetlany jest *MessageBox*, który informuje użytkownika w której linii brakuje średnika.

```

for (int i = 0; i < single_Line_Code.Length; ++i)
{
    if (single_Line_Code[i].IndexOf(";") == -1)
    {
        MessageBox.Show("ERROR at line: " + (i + 1).ToString() + "\nMissing ; ");
        start_program_button.Enabled = false;
    }
}

```

Podpunkt d) oraz e) zostały połączone oraz wykonywane są za pomocą instrukcji *try catch*, która w przypadku pojawienia się wyjątku informuje użytkownika, że w kodzie występują błędy. Każda linijka kodu odpowiadająca odpowiedniej komendzie oraz argumentowi zostaje podzielona na dwie części. Pierwsza część jest zapisywana do osobnej listy, w której przechowywane są wszystkie instrukcje wprowadzone przez użytkownika. Druga część zawiera same argumenty dla tych funkcji. Kolejność występowania funkcji oraz argumentów musi się zgadzać w obu listach. Dodatkowo z obu sekcji muszą zostać usunięte wszystkie dodatkowe znaki stanowiące o składani kodu – są to znaki otwarcia oraz zamknięcia nawiasu wraz ze średnikiem. Kolejnym etapem jest przeszukanie nowo utworzonych list w celu sprawdzenia czy wprowadzone przez użytkownika instrukcje są poprawne. W przeciwnym wypadku wywołany zostaje *MessageBox* informujący o błędnie wprowadzonej funkcji. Fragment kodu realizujący powyższy opis został przytoczony poniżej.

```

try
{
    for (int i = 0; i < single_Line_Code.Length; ++i)
    {
        if (single_Line_Code[i].IndexOf("(") < 0)
        {
            throw new Exception("Syntax ERROR at: " + (i+1).ToString() + " line");
        }
        only_instructions[i] = single_Line_Code[i].Remove(single_Line_Code[i].
            IndexOf("("));
        only_parameters[i] = single_Line_Code[i].Substring(single_Line_Code[i].
            IndexOf("(")).Trim(chars_to_trim);
    }
    bool is_instruction_correct;

    for (int i = 0; i < single_Line_Code.Length; ++i)
    {
        is_instruction_correct = false;
        if (only_instructions[i] == "MVS") is_instruction_correct = true;
        if (only_instructions[i] == "DLY") is_instruction_correct = true;
        if (only_instructions[i] == "SPEED") is_instruction_correct = true;
        if (only_instructions[i] == "TOOL") is_instruction_correct = true;

        if (!is_instruction_correct)
        {
            MessageBox.Show("Incorrect instruction")
            start_program_button.Enabled = false;
        }
    }
}

catch (Exception wyjatek)
{
    MessageBox.Show(wyjatek.Message);
    start_program_button.Enabled = false;
}

```

Podpunkt f) stanowi o sprawdzeniu zgodności nazw punktów wykorzystanych w kodzie z punktami zapisanymi w bazie punktów. Zostało to zrealizowane za pomocą dwóch pętli for. Zewnętrzna pętla przechodzi przez każdą z instrukcji wprowadzonych przez użytkownika w poszukiwaniu instrukcji *MVS*, natomiast wewnętrzna pętla w chwili kiedy pierwotna pętla natrafi na wspomnianą funkcję przeszukuje bazę punktów w poszukiwaniu punktu, którego nazwa pokrywa się z nazwą punktu wykorzystanego w kodzie. Jeśli dla każdego punktu zostanie znaleziony odpowiednik w bazie, wówczas zostaje przyznana możliwość przyciśnięcia przycisku *START BUTTON*.

```

bool point_check = false;
for (int j = 0; j < single_Line_Code.Length; ++j)
{
    if (only_instructions[j] == "MVS")
    {
        point_check = false;
        for (int m = 0; m < (pointsDataGridView.RowCount - 1); ++m)
        {
            if (pointsDataGridView[point_name_column, m].Value.ToString() ==
                only_parameters[j])
            {
                point_check = true;
            }
        }
        if (point_check == false)
        {
            MessageBox.Show("Missing point in base!!! \n" + only_parameters[j].
                           ToString() + " not found");
            start_program_button.Enabled = false;
        }
    }
}
if (point_check == true)
{
    start_program_button.Enabled = true;
}

```

Kolejną ważną kwestią jest sprawdzenie czy punkty dodane do bazy zawierają się w przestrzeni roboczej ramienia. Jest to realizowane w analogiczny sposób jak w przypadku ograniczenia podczas sterowania za pomocą gamepada. Kod odpowiedzialny za realizację tego zadania jest następujący:

```

for (int m = 0; m < (pointsDataGridView.RowCount - 1); ++m)
{
    curr_point_x = Convert.ToDouble(pointsDataGridView[X_column, m].Value.ToString());
    curr_point_y = Convert.ToDouble(pointsDataGridView[Y_column, m].Value.ToString());
    curr_point_z = Convert.ToDouble(pointsDataGridView[Z_column, m].Value.ToString());
    curr_point_epsilon = Convert.ToDouble(pointsDataGridView[epsilon_column, m]
                                         .Value.ToString());
    curr_point_name = pointsDataGridView[point_name_column, m].Value.ToString();
    if (Kin.point_in_range_check(curr_point_x, curr_point_y, curr_point_z,
                                 curr_point_epsilon))
    {
        point_check = true;
    }
    else

```

```

{
    point_check = false;
    MessageBox.Show("Compilation ERROR\nCoordinates of: " + current_point_name +
                    " exceed robot range\n");
}
}

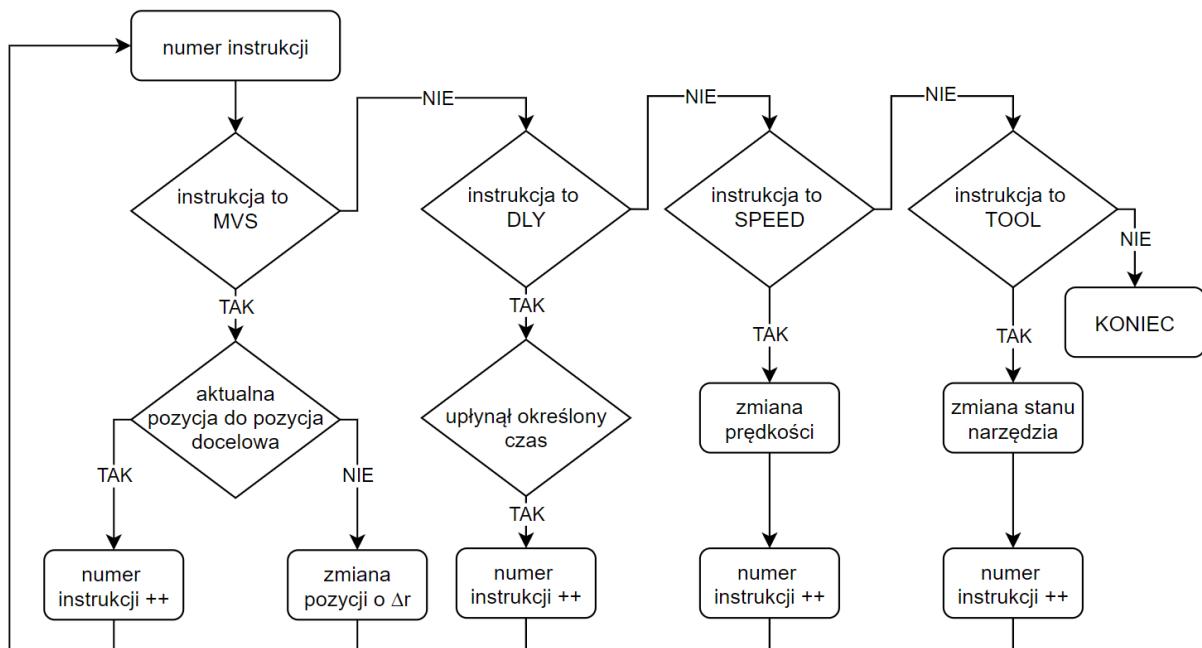
```

Powyższy kod opiera się o wykonanie sprawdzenia poprawności współrzędnych dla każdego punktu zamieszczonego w bazie poprzez odczytanie współrzędnych, przekazanie ich do metody `point_in_range_check` oraz w przypadku uzyskania błędного wyniku wyświetlany jest komunikat błędu komplikacji informujący, który punkt zawiera błędne współrzędne.

Kiedy wyżej opisane algorytmy nie wykryją błędów, do punktu będącego początkowym dla ruchu z wykorzystaniem instrukcji MVS zostaje przypisana aktualna pozycja ramienia, tak, aby nastąpił płynny ruch z punktu w którym aktualnie znajduje się ramię do punktu zadeklarowanego jako docelowy. Zadanie to realizuje ostatni punkt oraz kończy algorytm wykonywany podczas komplikacji kodu.

Przycisk *START PROGRAM* jest odpowiedzialny za uruchomienie timera realizującego wykonywanie uprzednio skompilowanego kodu. Przycisk *STOP PROGRAM* zatrzymuje wywoływanego timera co przekłada się na to, że program jest zatrzymywany w chwili wciśnięcia tego przycisku z czym idzie przerwanie ruchu ramienia. Ostatnim przyciskiem odpowiedzialnym za obsługę programu jest przycisk *RESET PROGRAM*. Wciskając ten przycisk następuje działanie jednakowe do tego wywoływanego przyciskiem *STOP PROGRAM*, jednak dodatkowo wszystkie informacje o postępie wykonywania programu zostają zresetowane. Jeśli użytkownik ponownie wciśnie przycisk *START PROGRAM*, wówczas kod będzie realizowany od początku.

Najważniejszą częścią kodu jest fragment odpowiedzialny za realizację kodu wprowadzonego przez użytkownika. Istotne jest aby każda instrukcja była wykonywana w kolejności w jakiej została zaprogramowana. W tym celu został wykorzystany mechanizm maszyny stanów.



Rys. 5.7. Uproszczony schemat maszyny stanów

Struktura ta realizowana jest przy każdorazowym wywołaniu timera odpowiedzialnego za wykonanie kodu. Podczas komplikacji została odczytana liczba instrukcji wprowadzona przez użytkownika. W każdym wywołaniu timera, sprawdzana jest aktualna instrukcja. Realizowane jest to za pomocą struktury *switch case*. Pierwszą instrukcją jest zawsze komenda wpisana jako pierwsza w kodzie. Dostępne są trzy podstawowe instrukcje. Opis każdej z poszczególnych instrukcji zaprezentowany został poniżej.

- MVS

Jest to instrukcja ruchu prostoliniowego. Aby móc wykonać tego typu ruch konieczna jest znajomość wektora kierunkowego, według którego powinien przebiegać ruch. W pierwszym wywołaniu timera, jako docelowe współrzędne są zapisywane wartości odczytane z bazy punktów odpowiadające punktowi będącemu argumentem funkcji MVS.

```
current_point = find_current_point();

if (czy_to_pierwsze_powtorzenie)
{
    goal_x = Convert.ToDouble(pointsDataGridView[X_column, current_point].Value.ToString());
    goal_y = Convert.ToDouble(pointsDataGridView[Y_column, current_point].Value.ToString());
    goal_z = Convert.ToDouble(pointsDataGridView[Z_column, current_point].Value.ToString());
    goal_epsilon = Convert.ToDouble(pointsDataGridView[epsilon_column,
                                                    current_point].Value.ToString());
    czy_to_pierwsze_powtorzenie = false;
}
```

Jak widać powyższy fragment kodu na początku znajduje aktualny punkt w bazie punktów a następnie pobierane są odpowiednie współrzędne, dodatkowo działanie to jest realizowane wyłącznie jednokrotnie, co jest zapewnione dzięki strukturze warunkowej *if* oraz odpowiedniemu ustaleniu zmiennej logicznej *czy_to_pierwsze_powtorzenie*.

```
double delta_x = goal_x - start_X;
double delta_y = goal_y - start_Y;
double delta_z = goal_z - start_Z;
double delta_epsilon = goal_epsilon - start_epsilon;
vector_length = Math.Sqrt(Math.Pow(delta_x, 2) + Math.Pow(delta_y, 2) + Math.Pow(delta_z, 2));
```

Aby obliczyć odległość brakującą do uzyskania punktu docelowego obliczana jest brakująca odległość dla każdej ze współrzędnych, a następnie obliczana długość tak określonego wektora:

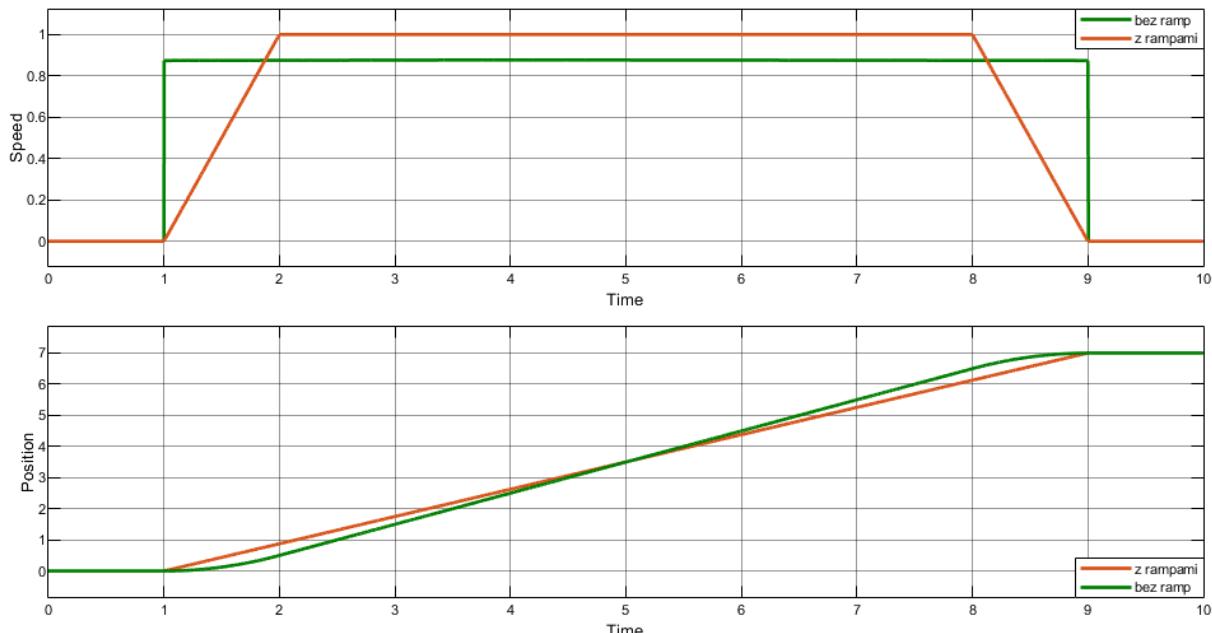
```
double missing_distance_x = Math.Abs(goal_x - Kin.X);
double missing_distance_y = Math.Abs(goal_y - Kin.Y);
double missing_distance_z = Math.Abs(goal_z - Kin.Z);
double missing_distance_epsilon = Math.Abs(goal_epsilon - Kin.epsilon);
double missing_global_distance = Math.Sqrt(Math.Pow(missing_distance_x, 2) +
                                             Math.Pow(missing_distance_y, 2) + Math.Pow(missing_distance_z, 2));
```

W celu sprawdzenia, czy punkt wodzący ramienia znajduje się w docelowej pozycji następuje sprawdzenie każdej ze współrzędnych

```
if (missing_distance_x <= delta_s && missing_distance_y <= delta_s &&
    missing_distance_z <= delta_s && missing_distance_epsilon <= delta_s)
{
    state_of_code = 1;
    start_X = Kin.X;
    start_Y = Kin.Y;
    start_Z = Kin.Z;
    start_epsilon = Kin.epsilon;
}
```

Jeśli brakujący dystans do osiągnięcia pozycji docelowej jest mniejszy bądź równy odległości, o którą w każdym wywołaniu timera zostaje zmieniona pozycja odpowiedniej współrzędnej, wówczas zmienna logiczna *state_of_code* zostaje zmieniona na wartość *true*, co wiąże się z przejściem do następnej instrukcji. Aktualne współrzędne punktu wodzącego zostają zapisane jako nowe współrzędne początkowe dla następnej instrukcji ruchu.

Ze względu na fakt, że rzeczywiste napędy nie są w stanie uzyskać nieskończonych przyspieszeń, konieczne jest zastosowanie ramp, które pozwolą się im rozpoczędzić do określonej prędkości, przy zachowaniu przyspieszenia na odpowiednim dla danego zastosowania poziomie.



Rys. 5.8. Porównanie przebiegów prędkości i położenia dla sterowania z rampami i bez nich

Jak widać na powyższym rysunku przebieg prędkości po zastosowaniu rampy charakteryzuje się większą ustaloną prędkością. Jest to celowy zabieg, którego efektem jest przebycie jednakowej odległości przez napędy o obu typach sterowania, co jest widoczne na dolnym przebiegu. Ze względu na zastosowanie ramp, zmniejszeniu ulega średnia prędkość na danym odcinku. Przekłada się to na krótszy przebyty dystans. Aby uniknąć wpływu tego efektu należy podnieść prędkość docelową. Dzięki temu uzyskamy płynne zmiany prędkości silnika, a jego działanie nie będzie tak gwałtowne.

Aby programowo wprowadzić rampy do sterowania, konieczna jest znajomość kilku parametrów. Najważniejszym parametrem jest prędkość docelowa. Wartość ta będzie definiowana przez użytkownika za pomocą instrukcji *SPEED*. Dodatkowo konieczna jest znajomość jednego z trzech parametrów:

- wartość przyspieszenia,
- czas przyspieszania,
- droga na której ma zostać uzyskana docelowa prędkość.

Równanie ruchu prostoliniowego jest w postaci:

$$s = s_0 + v_0 \cdot t + \frac{a \cdot t^2}{2} \quad [14.0]$$

gdzie:

s – aktualne położenie układu

s_0 – to położenie początkowe przed rozpoczęciem docelowego ruchu

v_0 – to prędkość początkowa przed rozpoczęciem docelowego ruchu

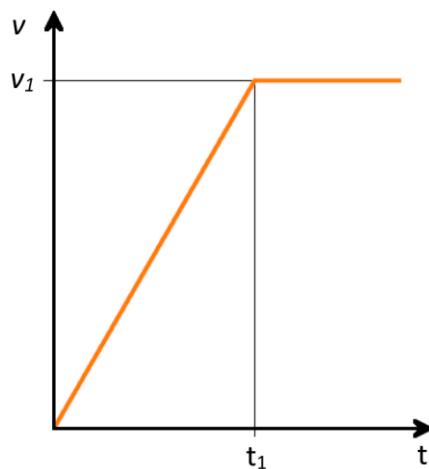
t – czas liczony od początku trwania ruchu

a – przyspieszenie

W rozpatrywanym przypadku można przyjąć, że ruch rozpoczynany jest zawsze w położeniu początkowym $s_0 = 0$, a prędkość początkowa wynosi $v_0 = 0$. Takie założenia sprowadzają powyższe równanie do zależności pomiędzy trzema zmiennymi jakimi są aktualny czas i położenie oraz przyspieszenie.

$$s = \frac{a \cdot t^2}{2} \quad [14.1]$$

Przyjmując nomenklaturę opartą na oznaczeniach z poniższego rysunku, gdzie v_1 oznacza docelową wartość prędkości, a t_1 oznacza czas konieczny do jej uzyskania, możliwe jest wyznaczenie wartości przyspieszenia na tej podstawie.



Rys. 5.9. Wartości prędkości oraz czasu określające charakter przyspieszenia

$$a = \frac{v_1}{t_1} \quad [14.2]$$

Na podstawie wzorów 14.1 oraz 14.2 możliwe jest uzależnienie przebytej odległości od prędkości docelowej v_1 , czasu rozpoczęcia t_1 oraz aktualnego czasu t .

$$s = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot t^2 \quad [14.3]$$

Chcąc przełożyć powyższe równanie na rzeczywiste działanie w układzie mikroprocesorowym, konieczne jest przejście z dziedziny ciągłej na dyskretną. Zostało to zrealizowane poprzez określenie interwału czasowego z jakim realizowane są poszczególne wywołania timera. Czas ten wynosi 10ms. Wówczas można przyjąć dodatkową zmienną

określającą numer aktualnego wywołania timera. Wzór określający położenie przyjmuje wówczas następującą postać:

$$s = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot (0.01 \cdot n)^2 \quad [14.4]$$

Zmienna t_1 przechowującą informację o czasie została zamieniona przez iloczyn zmiennej przechowującej informację o numerze aktualnego wywołania n oraz okresu trwania tego wywołania.

Dzięki takiemu podejściu możliwe jest określenie zmiany położenia jaka powinna nastąpić pomiędzy dwoma wywołaniami timera wraz z możliwością zmiany wartości prędkości docelowej oraz czasu przyspieszania.

$$s_1 = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot (0.01 \cdot (n - 1))^2 \quad [15.0]$$

$$s_2 = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot (0.01 \cdot n)^2 \quad [15.1]$$

$$\Delta s = s_2 - s_1 = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot 0.0001 \cdot [n^2 - (n^2 - 2n + 1)] \quad [15.2]$$

$$\Delta s = \frac{1}{2} \cdot \frac{v_1}{t_1} \cdot 0.0001 \cdot (2n - 1) \quad [15.3]$$

Równanie 15.3 określa o jaką wartość należy zwiększyć wartość aktualnej pozycji w każdym z wywołań timera, tak aby tego efektem był ruch jednostajnie przyspieszony po odpowiedniej rampie.

W celu implementacji tego podejścia w programie, konieczne jest określenie w jakim zakresie czasu ma następować realizacja przyspieszenia. W tym celu została obliczona ilość wywołań timera, która jest potrzebna do realizacji całego przyspieszenia, a dodatkowo stworzona została zmienna logiczna *czy_przyspieszasz* niosąca informację o tym czy w danej chwili realizowany jest ruch z przyspieszeniem.

```
acceleration_periods = Convert.ToInt32(acceleration_time * freq);

if ((current_period <= acceleration_periods) && czy_przyspieszasz)
{
    delta_s = 0.5 * speed / acceleration_time * 0.0001 * (2 * current_period - 1);
    current_period += 1;
}
else
{
    delta_s = speed / freq;
    czy_przyspieszasz = false;
}
```

W przypadku kiedy aktualny numer wywołania timera przekroczy ilość wywołań przeznaczoną na przyspieszanie, wówczas wartość zmiana położenia zostaje ustawiona jako stosunek prędkości do częstotliwości wywołań timera.

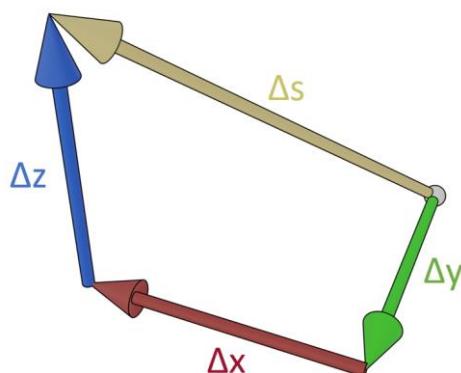
Powyższy fragment kodu realizuje kontrolę prędkości dla rampy narastającej. Konieczne jest zastosowanie analogicznego podejścia dla rampy hamującej. W tym celu należy obliczyć odległość jaką zajmie wykonanie hamowania. Możliwe jest to poprzez obliczenie pola powierzchni ograniczonego przez przebieg prędkości na zakresie czasu od 0 do t_1 (rysunek

4.9.). Mimo faktu, że jest to rampa realizująca przyspieszanie, przyjęte zostało podejście gdzie obie rampy są symetryczne (charakteryzują się takimi samymi czasami trwania).

Podobnie jak podczas przyspieszania, algorytm zwalniania realizowany jest wyłącznie jeśli pozostała odległość jaką ma przebyć punkt wodzący ramienia jest mniejsza bądź równa odległości jaką zajmie wyhamowanie.

```
double deceleration_road = 0.5 * speed * acceleration_time;
if ((missing_global_distance <= deceleration_road) && !czy_przyspieszasz)
{
    delta_s = 0.5 * speed / acceleration_time * 0.0001 * (2 * current_perioid - 1);
    current_perioid -= 1;
}
```

Kolejnym krokiem jest zamiana wartości zmiany położenia obliczonej dla wektora Δs , wyznaczającego najkrótszą odległość pomiędzy punktami między którymi ma odbyć się ruch na zmiany wartości dla każdej ze współrzędnych.



Rys. 5.10. Rozkład wektora głównej zmiany położenia na poszczególne współrzędne

Aby obliczyć wartości zmian dla poszczególnych współrzędnych należy zrzutować wektor Δs na każdą z osi. Można to zrealizować znając właściwości wektora wyznaczonego pomiędzy początkowym oraz końcowym punktem ruchu. Właściwości takie jak długość tego wektora oraz poszczególne jego składowe zostały obliczone wcześniej w kodzie. Kod realizujący rzutowanie wektora na poszczególne osie prezentuje się następująco:

```
double step_x = delta_s * delta_x / vector_length;
double step_y = delta_s * delta_y / vector_length;
double step_z = delta_s * delta_z / vector_length;
double step_epsilon = delta_epsilon * delta_s / vector_length;
```

Mimo tego, że wartość epsilon nie bierze czynnego udziału w tworzeniu wektora ruchu, tak jednak zmiana tej wartości została dopasowana do zmian pozostałych współrzędnych w taki sposób aby uzyskanie przez epsilon wartości docelowej współgrało z uzyskaniem tej pozycji poprzez pozostałe współrzędne.

Ostatnią ważną częścią tego algorytmu realizującego funkcję MVS jest dodanie wyżej obliczonych wartości zmiany dla każdej ze współrzędnych do aktualnego położenia punktu wodzącego ramienia. Oraz wykonanie obliczeń kinematyki odwrotnej w celu zamiany położenia we współrzędnych X, Y Z, epsilon na odpowiednie wartości kątów. Jest to realizowane za pomocą poniższego fragmentu kodu:

```

Kin.X += step_x;
Kin.Y += step_y;
Kin.Z += step_z;
Kin.epsilon += step_epsilon;
Kin.oblicz();

```

- **DLY**

Jest to instrukcja wymuszająca przerwę o określonym czasie trwania. Realizacja tej funkcji nie jest złożona oraz opiera się wyłącznie na zliczaniu ile wywołań timera upłynęło od pierwszego wywołania timera z tą funkcją, oraz sprawdzanie czy upłynął zadany czas. Wiedząc, że częstotliwość wywołań timera wynosi 100Hz, co przekłada się na okres wynoszący 10ms, wystarczy odczytać argument funkcji który został wprowadzony w milisekundach, odpowiednio dopasować go do zmiennej zliczającej wywołania timera (z funkcją DLY) oraz kontrolować czy przekroczony został docelowy czas. Kod realizujący to zadanie wygląda następująco:

```

int duration = Convert.ToInt32(current_parameter) / 10;
if (duration <= 2)
{
    duration = 2;
}
time_counter += 1;
if (time_counter >= duration)
{
    state_of_code = 1;
    time_counter = 0;
}

```

W chwili kiedy wartość zmiennej `time_counter` jest większa od wartości odpowiednio dopasowanej długości trwania przerwy `duration` następuje przejście do następnej fazy wykonywania kodu.

- **SPEED**

Jest to instrukcja odpowiedzialna za zmianę prędkości z jaką powinien poruszać się punkt wodzący ramienia. Funkcja ta jako argument przyjmuje wartości z zakresu od 1 do 100. Jest to reprezentacja procentowa wartości maksymalnej prędkości jaką może osiągnąć punkt wodzący. W celu określenia prędkości na podstawie jej wartości maksymalnej konieczna jest znajomość tej prędkości. Ze względu na bezpieczeństwo wartość ta została ustalona na 200 mm/s. Dodatkowo została przyjęta minimalna prędkość ustalona jako 10 mm/s. Algorytm realizujący funkcję speed w przeciwieństwie do pozostałych funkcji zawiera się w pojedynczym wywołaniu timera. Kolejność działań wykonywanych przez algorytm jest następująca:

- odczytany argument jest sprawdzany pod kątem zawierania się w domyślnym przedziale oraz wprowadzane są ewentualne poprawki,
- następuje mapowanie wartości odczytanej z zakresu od 1 do 100 [%] na zakres od 1 do 20 [cm].

Kod odpowiedzialny za realizację funkcji SPEED prezentuje się następująco:

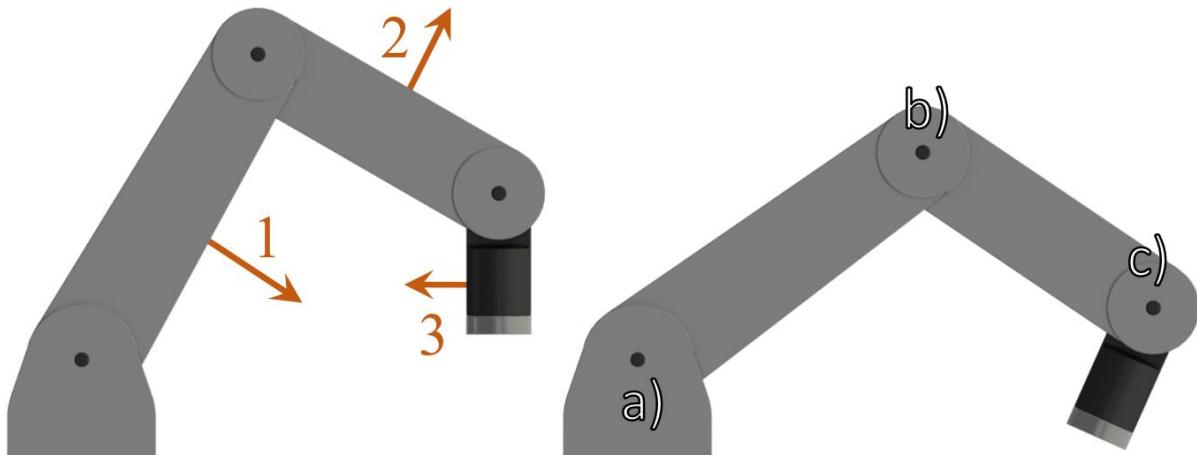
```
int odczyt = Convert.ToInt32(current_parameter);
if (odczyt < 1) odczyt = 1;
if (odczyt > 100) odczyt = 100;
speed = Convert.ToSingle(odczyt);
speed = (speed - 1) * (20 - 1) / (100 - 1) + 1;
state_of_code = 1;
```

- **TOOL**

Jest to instrukcja realizująca zmianę wartości bitu odpowiedzialnego za stan narzędzia. Jej działanie opiera się o zapisanie do zmiennej tekstowej instrukcji jaka ma zostać przesłana do mikrokontrolera. Realizowane jest to za pomocą poniższego fragmentu kodu.

```
string odczytane = current_parameter;
instruction_for_uC = "T" + odczytane;
state_of_code = 1;
```

Zasada działania obliczeń kinematyki jest taka sama jak przedstawione zostało to w rozdziale 4.3. Jednak przyjęty model układu napędowego ramienia wpływa na jego ruchy tak, że podczas wykonywania ruchu oznaczonego strzałką 1 na poniższym rysunku jednocześnie są wykonywane ruchy oznaczone strzałkami 2 oraz 3. Jest to spowodowane przeniesieniem napędu za pomocą kół i pasów zębatach. Nie jest więc możliwe wykonanie ruchu sekcją drugą tak, aby kąty λ_2 oraz λ_3 nie uległy zmianie. W przypadku ruchu wyłącznie sekcją trzecią (ruch według strzałki 2) następuje również ruch sekcji czwartej. Oznacza to, że każdy następny stopień jest sprzężony mechanicznie ze stopniem poprzednim.



Rys. 5.11. Graficzne przedstawienie problematyki sprzężenia ze sobą sekcji ramienia

Do poszczególnych sekcji napęd jest przenoszony za pomocą przekładni pasowo zębatach. Stosunek przełożenia dla napędu napędzającego sekcję 4 pomiędzy punktami a) oraz b) wynosi 6:5, natomiast przełożenie pomiędzy punktami b) oraz c) wynosi 1:1. Dla napędu sekcji trzeciej występuje analogiczne przełożenie. W tym przypadku pomiędzy punktami a) i b) wynosi ono również 6:5. Oznacza to, że jeśli wykonamy ruch sekcją drugą w kierunku wskazywanym przez

strzałkę pierwszą na rysunku 5.11. o kąt 24° , wówczas nastąpi odchylenie się sekcji trzeciej o kąt 20° w kierunku określonym przez strzałkę nr. 2, a sekcja czwarta wychyli się również o 20° w kierunku zgodnym ze strzałką nr. 3. Wykonując ruch sekcją trzecią w kierunku określonym przez strzałkę nr. 2 o kąt wynoszący 10° , sekcja czwarta odchyli się zgodnie z kierunkiem strzałki 3. o kąt 10° . Poruszając sekcją czwartą nie wpływamy na poprzednie sekcje, analogia ta przenosi się również na poruszanie sekcją trzecią.

Analizując wyżej opisane zachowanie układu przeniesienia napędu, można wykonać opis matematyczny pozwalający na określenie odpowiedniej pozycji kątowej każdego z silników tak aby podczas ruchu sekcją drugą nie wpływać na kąty λ_2 oraz λ_3 oraz uniknąć analogicznego zachowania dla ruchu sekcją trzecią. Docelowe wartości kątów pomiędzy sekcją drugą i trzecią to λ_2 , a pomiędzy sekcją trzecią i czwartą to λ_3 . Kąty w jakich mają się znaleźć poszczególne silniki wraz z uwzględnieniem opisanych mechanizmów sprzężeń opisane zostały jako:

$$\lambda_{silnik_1} = \lambda_1 \quad [16.1]$$

$$\lambda_{silnik_2} = \lambda_2 - \frac{5}{6} \cdot \lambda_{silnik_1} = \lambda_2 - \frac{5}{6} \cdot \lambda_1 \quad [16.2]$$

$$\lambda_{silnik_3} = \lambda_3 - \lambda_{silnik_2} = \lambda_3 - \lambda_2 + \frac{5}{6} \cdot \lambda_1 \quad [16.3]$$

5.2. Omówienie programu wykonawczego mikrokontrolera

Układ mikroprocesorowy ze względu na swoje zastosowanie musi spełniać wymagania, które pozwolą na bezproblemową realizację zadań którymi są:

- sterowania układem napędowym,
- obsługa komunikacji szeregowej,
- sterowanie układem sygnalizującym stany awaryjne.

Ważnym aspektem jest więc prędkość z jaką będą wykonywane poszczególne zadania. Dodatkowo układ ten powinien zapewniać odpowiednie moce obliczeniowe ze względu na konieczność ciągłej analizy danych wejściowych – interpretacja komunikatów wysyłanych przez aplikację komputerową. Ze względu na sporą liczbę układów do obsłużenia, konieczne jest aby układ zawierał odpowiednią liczbę portów WE/WY. Zgodnie z opisem w rozdziale 4.5 wykorzystanym układem jest płytka rozwojowa Teensy 4.1. Najważniejszymi cechami tej płytki są [29]:

- jest to układ oparty na architekturze ARM, a układem wykonawczym jest 32 bitowa jednostka ARM Cortex-M7,
- obecność 42 pinów WE/WY,
- obsługa do 7 portów szeregowych wraz z 4B FIFO,
- wbudowane złącze USB,
- taktowanie procesora wynoszące 600MHz, z możliwością podkręcenia do 1.2GHz,
- zgodność ze środowiskiem programistycznym Arduino IDE.

Ze względu na możliwość programowania płytka wykorzystując środowisko Arduino IDE, podczas tworzenia programu wykorzystane zostało środowisko Visual Micro, które zapewnia jednakowe możliwości co Arduino IDE, jednak oferuje bogatszy interfejs, co przekłada się na większą wygodę podczas tworzenia kodu.



Rys. 5.12. Płytki rozwojowa Teensy 4.1 [29]

Program realizowany przez mikrokontroler składa się z pięciu głównych części. Pierwszą częścią jest kod realizowany zawsze w pierwszym cyklu pracy mikrokontrolera. W tej części ustalana jest komunikacja z aplikacją komputerową, oraz realizowane są funkcje zerowania wszystkich ważnych zmiennych oraz ustawienia dotyczące zachowania wszystkich pinów. Kod odpowiedzialny za tą część jest następujący:

```
Serial.begin(115200);
setPinMode();
setAllAnglesToZero();
```

Kolejną bardzo ważną częścią kodu jest pętla główna realizowana przez cały czas działania mikrokontrolera. W pętli tej znajduje się ciągły odczyt danych odbieranych poprzez port szeregowy oraz wysyłany jest odpowiedni komunikat zwrotny. Kod odpowiedzialny za realizację tego zadania znajduje się poniżej:

```
if(Serial.available() > 0)
{
    incoming_data = Serial.readStringUntil('x');
    incoming_data.remove(0,1);
    if (!find_position)
    {
        getData(incoming_data);
    }
    /*
    send_data = w tym miejscu znajduje się odpowiedni komunikat
    Serial.println(send_data);
    */
}
```

Jak widać realizacja całego fragmentu kodu jest możliwa dopiero wówczas, kiedy aplikacja komputerowa nawiąże połączenie z mikrokontrolerem. Dane odczytywane są do czasu kiedy odczytany zostanie znak kończący串 znaków – „x”. Dodatkowo jeśli odczytany zostanie komunikat stanowiący, że użytkownik chce przeprowadzić pozycjonowanie napędów (znalezienie punktów zerowych) wówczas odebrane dane nie zostają odpowiednio interpretowane za pomocą funkcji `getData` oraz realizowany jest odpowiedni algorytm. W zaprezentowanym powyżej kodzie dane wysyłane do komputera przechowywane są

w zmiennej `send_data` jednak jest to wyłącznie przygotowanie kodu pod przyszłe modyfikacje, dzięki którym możliwe będzie rozwinięcie aplikacji komputerowej o wprowadzenie elementów takich jak funkcja `if(input_state)`, która będzie stanowiła o realizacji wybranego fragmentu kodu na podstawie stanu odpowiedniego wejścia. Pozwoli to na możliwość prostej komunikacji pomiędzy robotem a procesem w którym bierze on udział. Inną funkcjonalnością jaką może wprowadzić komunikacja dwukierunkowa jest możliwość implementacji różnych czujników, które będą komunikować się z mikrokontrolerem, który zaś będzie przekazywał informację do aplikacji komputerowej.

Kolejnym istotnym zagadnieniem jest interpretacja danych wysyłanych przez aplikację komputerową. Sposób transmisji danych został opisany w rozdziale 5.1. Wysyłany komunikat składa się z docelowych wartości które mają zostać odczytane rozgraniczonych kolejnymi literami alfabetu od a do e oraz dodatkowego komunikatu po którym występuje znak „x”. Kod realizujący odczyt wartości z ciągu znaków znajduje się wewnątrz funkcji `getData()` oraz został opisany poniżej:

```
int pos_a = inputData.indexOf("a");
int pos_b = inputData.indexOf("b");
int pos_c = inputData.indexOf("c");
int pos_d = inputData.indexOf("d");
int pos_e = inputData.indexOf("e");
int pos_x = inputData.indexOf("x");

lambda_0 = inputData.substring(pos_a+1, pos_b).replace(", ", ".").toFloat();
lambda_1 = inputData.substring(pos_b+1, pos_c).replace(", ", ".").toFloat();
lambda_2 = inputData.substring(pos_c+1, pos_d).replace(", ", ".").toFloat();
lambda_3 = inputData.substring(pos_d+1, pos_e).replace(", ", ".").toFloat();
instruction = inputData.substring(pos_e+1, inputData.length());

updateAngles();

if (instruction == "find_position")
{
    find_position = true;
    positioning_stage = 0;
    setAllAnglesToZero();
    myTimer.begin(timer_tick, timer_interval);
}

if (instruction == "TON")
{
    tool_state = true;
}

if (instruction == "TOFF")
{
    tool_state = false;
}
```

W pierwszej części sprawdzana jest pozycja poszczególnych znaków w komunikacie wejściowym, tak by na jej podstawie określić poszczególne liczby stanowiące docelowe kąty dla każdego z silników. Aby to zrealizować konieczna jest jeszcze konwersja liczby w której część dziesiętna odgraniczona jest przecinkiem na liczbę gdzie znakiem odgraniczającym jest kropka. Po takiej zmianie możliwa jest zamiana ciągu znaków na liczbę zmiennoprzecinkową pojedynczej precyzji. Dodatkowo odczytywana jest ewentualna instrukcja, która przesyłana jest w formie słownej, tak więc nie jest potrzebna jakakolwiek konwersja. W momencie, kiedy

wszystkie dane zostaną odczytane wykonana zostaje funkcja `updateAngles()`, dzięki której zmienne przechowujące informacje o kodzie zostają zaktualizowane o nowo odczytane wartości. Jeśli użytkownik wcisnie przycisk *FIND POSITION* w aplikacji komputerowej, wówczas po odczytaniu odpowiedniego komunikatu, następuje rozpoczęcie procedury ustalania punktów odniesienia dla ruchu. Jeśli odczytana instrukcja będzie informowała o załączniu dodatkowego narzędzia (TON) lub jego wyłączeniu (TOFF), wówczas do zmiennej odpowiedzialnej za przechowywanie stanu bitu sterującego narzędziem zapisywany jest odpowiedni stan.

Podejście zastosowane podczas realizacji algorytmu sterowania nadążnego (mikrokontroler podąża za pozycją jaka została mu przekazana poprzez aplikację) opiera się o realizację każdego ruchu przy pomocy timera. Każde wywołanie timera realizuje jeden z 3 kroków potrzebnych do sterowania. Aby wykonać ruch silnikiem konieczne jest podanie impulsu sterującego na odpowiedni pin kontrolera. Równolegle powinien pojawić się odpowiedni stan na pinie odpowiedzialnym za ustalenie kierunku obrotu wirnika. Aby wykonać impuls sterujący konieczne jest początkowe ustalenie stanu niskiego na danym pinie, następnie należy podać stan wysoki oraz zakończyć całą sekwencję ponownie stanem niskim. Realizacja tego algorytmu została oparta na wykorzystaniu struktury *switch case*, gdzie podczas każdego wywołania timera realizowany jest jeden z kroków. Kod odpowiedzialny za to zadanie jest następujący:

```
switch (timer_execution_count)
{
    case 0:
    {
        digitalWrite(motor_0_step_pin, 0);
    }
    case 1:
    {
        if (motor_0_angle > (motor_0_current_angle + 0.225))
        {
            digitalWrite(motor_0_dir_pin, counterclockwise);
            digitalWrite(motor_0_step_pin, 1);
            motor_0_current_angle += 0.225;
        }
        if (motor_0_angle < (motor_0_current_angle - 0.225))
        {
            digitalWrite(motor_0_dir_pin, clockwise);
            digitalWrite(motor_0_step_pin, 1);
            motor_0_current_angle -= 0.225;
        }
        break;
    case 2:
    {
        digitalWrite(motor_0_step_pin, 0);
        break;
    }
    default:
    {
        break;
    }
}
timer_execution_count += 1;
if (timer_execution_count > 9)
{
    timer_execution_count = 0;
```

```
}
```

Działanie powyższego fragmentu kodu realizuje obsługę jednego silnika, docelowy kod obsługuje wszystkie cztery silniki, jednak zasada działania jest jednakowa. W pierwszym wywołaniu timera – „`case 0`”, odpowiednie wyjścia są ustawiane w stan niski. W drugim wywołaniu – „`case 0`”, następuje sprawdzenie, czy docelowa pozycja każdego z wirników jest większa lub mniejsza od aktualnej pozycji (skorelowanej o wartość pojedynczego kroku dla zachowania stabilności układu). Jeśli wartość docelowa jest różna od aktualnej, wówczas następuje ruch w odpowiednią stronę oraz zmiana wartości aktualnego położenia danego wirnika. W ostatnim kroku – „`case 2`”, pin odpowiedzialny za realizację kroku jest ustawiany w stan niski. Ostatnią częścią realizowaną w pojedynczym wywołaniu jest zwiększenie wartości sterującej strukturą `switch case` oraz sprawdzenie czy nie wykracza ona poza maksymalną wartość.

Ważnym elementem który również zawiera się w strukturze wywołania timera jest część realizująca algorytm znajdujący pozycję odniesienia ramienia. W tym przypadku następuje ciągłe sprawdzanie sygnału pochodzącego z wyłącznika krańcowego, który sygnalizuje, że osiągnięta została odpowiednia pozycja. Po uzyskaniu odpowiedniej pozycji przez konkretną sekcję, następuje ruch do położenia określonego jako położenie startowe ramienia. Pozycjonowanie realizowane jest po kolej dla każdej z sekcji ramienia, zaczynając od pozycjonowania silnika odpowiedzialnego za obrót wzdłuż osi z, następnie pozycjonowana jest druga sekcja ramienia, następnie trzecia oraz czwarta. Całość ruchu składa się z 8 części, gdzie każda z nich realizowana jest za pomocą struktury `switch case`, a warunkiem wykonywania tej procedury jest wartość logiczna zmiennej `find_position`.

```
if (find_position)
{
    switch (positioning_stage)
    {
        case 0:
        {
            if (digitalRead(limit_switch_0_up) == LOW)
            {
                lambda_0 = up_limit_position_for_motor_0;
                updateAngles();
                motor_0_current_angle = motor_0_angle;
                positioning_stage += 1;
                break;
            }
            if (timer_execution_count == 2)
            {
                lambda_0 += 0.00675;
                updateAngles();
            }
            break;
        }
        case 1:
        {
            if (lambda_0 > target_lambda_for_motor_0)
            {
                if (timer_execution_count == 2)
                {
                    lambda_0 -= 0.00675;
                    updateAngles();
                }
            }
        }
    }
}
```

```

        else
        {
            positioning_stage += 1;
        }
        break;
    }
}
if (positioning_stage > 7)
{
    find_position = false;
    positioning_stage = 0;
}
}

```

Jak widać powyżej podczas realizacji „`case 0`” sprawdzane jest czy sygnał pochodzący z wyłącznika krańcowego (typ NO, zwierający do masy), jest niski (LOW), jeśli nie jest wówczas zwiększana jest wartość docelowa kąta w którym znajduje się wirnik. Rzeczywista wartość jaką powinna przechowywać ta zmienna nie jest znana. Dopiero w chwili kiedy zostaną zwarte styki wyłącznika krańcowego, następuje przypisanie odpowiedniej pozycji do omawianej zmiennej oraz następuje przejście do następnego etapu pozycjonowania. W kolejnym etapie wymuszone zostaje przejście do pozycji startowej dla każdego z silników. W jednakowy sposób realizowane jest pozycjonowanie pozostałych osi robota, jednak dla sekcji drugiej oraz trzeciej konieczne jest ciągłe korygowanie pozycji odpowiednich sekcji względem siebie, zgodnie z problematyką przedstawioną na rysunku 5.11. W przeciwnym wypadku możliwe byłoby uszkodzenie którejś z sekcji robota.

6. Badania laboratoryjne

Roboty przemysłowe odznaczają się bardzo wysoką precyzją. Wymogiem jaki musi spełnić taki robot jest zachowanie wysokiej powtarzalności oraz dokładności ruchów. Aby określić dokładność ruchów robota zostały przeprowadzone 2 rodzaje testów.

- Pierwszy z nich badał dokładność z jaką robot pojawia się w pozycji ustalonej jako pozycja startowa po wykonanym pozycjonowaniu – badanie dokładności pozycjonowania.
- Drugi test badał powtarzalność poprzez wielokrotne ruchy z pozycji wyjściowej do pozycji P1 – badanie powtarzalności ruchu.

Do badań został wykorzystany czujnik zegarowy marki HOLEX o numerze seryjnym 43 1980. Podziałka w tym mierniku wynosi 0.01mm, a błąd nie przekracza 6 μ m.

Badanie dokładności pozycjonowania

Badanie to zostało przeprowadzone dla dwóch układów położen miernika. W pierwszej serii pomiarów (pomiary 1 – 5) badana została dokładność ruchu w płaszczyźnie poziomej. W drugiej serii (pomiary 6 – 10) badana była dokładność ruchów w płaszczyźnie pionowej. Badanie zostało przeprowadzone w następującej kolejności:

- umieszczenie przyrządu badawczego w docelowej pozycji, którą jest pozycja startowa robota – punkt wodzący znajduje się we współrzędnych (0cm, 30cm, 30cm, 90°),
- odczytanie wartości pomiaru z przyrządu oraz uznanie jej za wzorcową,
- uruchomienie procedury pozycjonowania robota,
- odczytanie wartości z miernika,
- powtórzenie powyższych dwóch etapów czterokrotnie.

Tab. 6.1. Wyniki pomiarów dokładności pozycjonowania robota

nr pomiaru	wskazanie miernika [mm]	wskazanie wzorcowe [mm]	błąd pozycjonowania [mm]	średni błąd [mm]
1	4,34	4,65	0,31	0,22
2	4,75		0,10	
3	4,82		0,17	
4	4,48		0,17	
5	5,02		0,37	
6	3,25	2,71	0,54	0,69
7	2,07		0,64	
8	2,30		0,41	
9	1,96		0,75	
10	3,81		1,10	



Rys. 6.1. Widok układu pomiarowego do badania precyzji ruchów

Powyższe wyniki wykazują występowanie pewnych błędów podczas każdorazowego uruchomienia procedury pozycjonowania ramienia. Błędy te bezpośrednio przekładają się na występowanie błędów w pozostałych ruchach ramienia, ponieważ w przypadku błędnego określenia pozycji zerowej (początkowej) każdy ruch startujący z tej pozycji będzie obarczony błędem. Wartość średnia błędu bezwzględnego podczas pozycjonowania w płaszczyźnie poziomej jest mniejsza od wartości błędu dla płaszczyzny pionowej. Dzieje się tak, ponieważ ruch w płaszczyźnie poziomej jest realizowany głównie przez silnik odpowiedzialny za ruch wokół osi z, natomiast na ruch w płaszczyźnie pionowej składają się ruchy pozostałych silników. Jeśli każdy z napędów rozpoczęcie pracę z pewnym błędem, wówczas globalny błąd

będzie pewnego rodzaju sumą tych błędów, co przekłada się na większą wartość błędu bezwzględnego w płaszczyźnie pionowej. Analizując średnie wartości błędów można powiedzieć, że pod względem dokładności pozycjonowania konstrukcja robota radzi sobie bardzo dobrze. Zmierzone błędy nie przekraczają 1mm, co dla tak eksperimentalnej konstrukcji świadczy o wysokiej jakości wykonanych pasowań poszczególnych elementów. Do przeniesienia napędu zostały wykorzystane paski oraz koła zębate. Dzięki temu podejściu wykluczone zostały możliwe błędy nawrotne co również przekłada się na jakość wyników.

Badanie powtarzalności ruchu

Badanie to podobnie jak poprzednie zostało przeprowadzone w dwóch seriach po pięć pomiarów. Badanie to polegało na określeniu z jaką dokładnością ramie jest w stanie wykonać ruch z pozycji startowej do nowej, a następnie powrócić do pozycji startowej. Również w tym przypadku pomiary dotyczyły dokładności w płaszczyźnie poziomej oraz pionowej. Badanie zostało przeprowadzone w następujący sposób:

- wykonanie pozycjonowania ramienia,
- pomiar wartości stanowiącej wartość wzorcową
- ruch z punktu startowego o współrzędnych (0cm, 30cm, 30cm, 90°), do punktu P1 określonego jako (25cm, 45cm, 30cm, 90°), a następnie powrót do pozycji startowej,
- odczytanie wartości z miernika,
- powtórzenie powyższych dwóch etapów czterokrotnie.

Tab. 6.2. Wyniki pomiarów powtarzalności ruchów robota

nr pomiaru	wskazanie miernika [mm]	wskazanie wzorcowe [mm]	błąd pozycjonowania [mm]	średni błąd [mm]
1	7,4	7,14	0,26	0,20
2	7,14		0,00	
3	6,89		0,25	
4	7,35		0,21	
5	7,4		0,26	
6	3,57	3,95	0,38	0,49
7	3,14		0,81	
8	4,68		0,73	
9	4,12		0,17	
10	3,58		0,37	

Analiza powyższych wyników utwierdza w przekonaniu, że niemal każdy ruch robota obarczony jest błędem, jednak błąd ten nie zwiększa się wraz z ilością ruchów (przynajmniej dla branej pod uwagę liczby ruchów). Średni błąd w płaszczyźnie poziomej charakteryzuje się wartością na poziomie 0.20mm , natomiast w płaszczyźnie pionowej błąd ten wynosi 0.49mm. Wyniki te ponownie świadczą o dobrej konstrukcji ramienia pod względem sztywności oraz spasowania elementów.

Powyższe pomiary były wykonywane przy programowym ustawieniu prędkości na 10%. Dla wartości powyżej 20% robot wpadał w drgania podczas zatrzymywania się w punktach docelowych. Drgania te szybko przeradzały się w oscylacje. Jednakowe zachowanie można uświadomić kiedy robot znajduje się w bezruchu oraz nagle zostanie wyprowadzony z położenia równowagi np. poprzez ręczne wymuszenie zmiany położenia. Ze względu na wykorzystanie silników krokowych pracujących w pętli sprzężenia zwrotnego układ napędowy dąży do likwidacji powstałego uchybu za pomocą zaimplementowanych regulatorów. Podczas niwelowania błędu występuje przeregulowanie, natomiast ze względu na dużą bezwładność układu przeregulowanie to przybiera na sile oraz przeradza się w oscylacje które powoli gasną. W celu poprawy działania układu konieczne jest wprowadzenie zmian w parametrach nastaw regulatorów położenia. Wykorzystane karty umożliwiają ich parametryzację za pomocą komunikacji poprzez protokół RS232. Niestety w przypadku wykorzystanych w projekcie kart, komunikacja ta nie działa. Pomimo licznych prób nawiązania komunikacji poprzez dedykowane oprogramowanie nie udało się nawiązać stabilnego połączenia z żadną z kart sterowniczych. Przeszukując różne fora internetowe można natrafić na wiele podobnych wątków opisujące jednakowe problemy. Wykorzystane karty pomimo bycia opisanymi jednakowym numerem identyfikacyjnym różnią się pod względem wykorzystanych elementów elektronicznych. Przekłada się to na brak możliwości określenia czy podczas zakupu trafimy na model karty w której komunikacja będzie działała, czy nie. Ze względu na powyższy problem nie jest możliwe osiąganie większych prędkości od wartości około 20% prędkości maksymalnej, ponieważ inaczej pojawiają się oscylacje, które uniemożliwiają precyzyjny ruch ramieniem.

Badania udźwigu robota

Ze względu na bezpośrednie powiązanie położenia punktu wodzącego robota wraz z momentami obciążenia na każdym ze stawów udźwig robota będzie również zależny od położenia w którym znajduje się ramię. Teoretyczny udźwig może zostać ustalony za pomocą obliczeń obciążień jakie zostały zrealizowane w rozdziale – Obliczenia obciążzeń dla napędów. Znając moment napędowy jaki jest generowany przez silniki wraz z odpowiednimi przekładniami możemy za pomocą stworzonej aplikacji wstępnie określić, czy robot podoła planowanemu zadaniu. W trakcie pracy robota najczęściej wykorzystywane pole robocze znajduje się w pobliżu osi z. Przekłada się to na fakt, że generowane są niewielkie momenty obciążzeń, z czym idzie zdolność do manewrowania elementami o wadze do nawet 4.5kg. Ze względu na wykorzystany elektromagnes będący elementem trzymającym przemieszczane przez robota części maksymalna masa jaką jest w stanie unieść robot wynosi 2kg. Ustawiając robota w maksymalnym wychyleniu (generując największe z możliwych momentów obciążzeń) element o największej wadze, jaki był w stanie unieść robot ważył 0.4kg

Podczas tego badania odnotowano, że wraz z obciążeniem robota masą 4kg, wcześniej obserwowane oscylacje uległy zmniejszeniu, jednak w dalszym ciągu miały niegasnący charakter.

7. Podsumowanie

Tematem pracy jest budowa eksperimentalnego robota ramieniowego. Znaczenie słowa „eksperiment” w tej pracy jest kluczowe, ponieważ zaprojektowany robot, mimo sprawnego działania i prostej w porównaniu do przemysłowych rozwiązań konstrukcji posiada jednak wiele wad. Zarówno konstrukcja jak i układ sterowania robota nie są idealne oraz wymagają poprawek. Wynika to z wielu kompromisów, które należało przyjąć przy budowie urządzenia. Niedociągnięcia te nie skreślają jednak efektów pracy, ich analiza pozwoli na ulepszenie zaproponowanej konstrukcji. Budowa robota jest zadaniem, które przysparza licznych problemów inżynierijnych na wielu płaszczyznach. Główne problemy dotyczą zagadnień takich jak rozwiązania mechaniczne (np. wybór odpowiednich przekładni), elektryczne, elektroniczne (układy sterujące oraz napędy) a także informatyczne (projekt interfejsu graficznego dla operatora). Wszystkie wspomniane zagadnienia są złożone, przez co opisywany projekt zawiera liczne uproszczenia, które jednak zgodnie z założeniami pracy umożliwiają osiągnięcie zamierzonych efektów.

Sposób sterowania robotem za pomocą gamepada jest zgodny z zamierzeniami oraz skutecznie uniemożliwia wykonanie przypadkowego ruchu a wprowadzona możliwość automatyzacji działa w sposób poprawny. Zastosowana interpolacja liniowa daje duże możliwości ruchu ramion robota, jednak uniemożliwia wykonanie bardziej złożonych ruchów. W przyszłości mogą zostać wprowadzone ulepszenia, takie jak dodanie innych trybów ruchu (np. ruch po okręgu) lub dodanie obsługi prostej komunikacji z zewnętrznymi procesami w postaci odczytu stanu na wejściach logicznych.

Z mechanicznego punktu widzenia elementem, który wymaga poprawy jest sposób przeniesienia napędu. Wysunięcie silników poza samo ramię jest dobrym podejściem, które znacząco pozwoliło odciążyć konstrukcję, jednak dużą wadą wykonanego napędu jest budowa wykonanych przekładni. W celu zapewnienia płynniejszego ruchu konieczne jest zastosowanie innych przekładni, np. cykloidalnych lub falowych, które łączą zalety takie jak wysokie przełożenia, zwarta i kompaktowa konstrukcja oraz bardzo małe luzy. Niestety, dużą wadą takich rozwiązań jest ich wysoka cena, ponieważ koszt zakupu pojedynczej przekładni był większy od kwoty przeznaczonej na realizację całego robota. Opisywane problemy wynikają w dużej mierze z budżetu, który miał znaczący wpływ na wykorzystane rozwiązania. Innym, ważnym problemem okazały się karty sterujące silnikami krokowymi, które przekreśliły możliwość dostrojenia regulatorów, co znacząco odbija się na jakości ruchów robota oraz ogranicza prędkość robota.

Wykonanie elementów konstrukcyjnych robota stanowi ogromne wyzwanie. W tym przypadku konieczna jest znajomość zagadnień związanych z mechaniką i obróbką skrawaniem oraz dostęp do odpowiedniego parku maszynowego. Osobą, która znacząco wsparła wykonanie pracy dyplomowej jest ojciec autora, którego doświadczenie oraz umiejętności pozwoliły na odpowiednie wykonanie zaprojektowanych części robota. Poza nakładem pracy jaki był konieczny do wykonania pracy niezbędne było sfinansowanie zakupu elementów, takich jak silniki czy układy sterowania. W tym miejscu autor pracy pragnie podziękować rodzicom za okazaną pomoc oraz nieocenione wsparcie.

Biorąc pod uwagę uzyskane rezultaty należy uznać, że cel pracy został osiągnięty, z zakres zrealizowany zgodnie z założeniami.

Literatura

- [1] Honczarenko J., „Roboty przemysłowe”, WNT, 2010r.
- [2] Kaczmarek. W., Panasiuk J., Robotyzacja procesów produkcyjnych, PWN, 2017r.
- [3] Kevin M. Lynch and Frank C. Park, „MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL”, Cambridge University Press, 2017r.
- [4] Riazollah F, „Servo Motors and Industrial Control”, Springer IPS, 2008r.
- [5] „Beyond Asimov: The Three Laws of Responsible Robotics”, IEEE Intelligent Systems
- [6] wykład “Napędy robotów i obrabiarek” uczelnia: PWR, autor: prof. Kowalski
- [7] wykład “Podstawy robotyki”, uczelnia: AGH
- [8] operating manual IRC5 with FlexPendant, ABB
- [9] Programowanie robotów przemysłowych Mitsubishi w języku MELFA-BASIC V.
Zadanie typu Pick & Place autorstwa mgr inż. Grzegorza Piecucha
- [10] <https://www.bostondynamics.com/atlas> (dostęp w dniu 20.12.2021)
- [11] <https://www.omc-stepperonline.com/closed-loop-stepper-motor/economic-nema-34-closed-loop-stepper-motor-4-8-nm-679-87oz-in-encoder-1000cpr.html> (dostęp w dniu 20.12.2021)
- [12] <https://www.omc-stepperonline.com/closed-loop-stepper-motor/e-series-nema-34-closed-loop-stepper-motor-9-nm-1274-76oz-in-encoder-1000cpr.html> (dostęp w dniu 20.12.2021)
- [13] <https://www.omc-stepperonline.com/closed-loop-stepper-motor/economic-nema-34-closed-loop-stepper-motor-12-0-nm-1699-68oz-in-encoder-1000cpr.html> (dostęp w dniu 20.12.2021)
- [14] <https://www.omc-stepperonline.com/closed-loop-stepper-motor/page/1> (dostęp w dniu 20.12.2021)
- [15] <https://pl.aliexpress.com/i/1005001941019614.html> (dostęp w dniu 20.12.2021)
- [16] <https://www.omc-stepperonline.com/support-power-supply/how-to-choose-a-power-supply-for-my-stepper-motors> (dostęp w dniu 20.12.2021)
- [17] <https://pl.aliexpress.com/i/2028432537.html> (dostęp w dniu 20.12.2021)
- [18] https://pl.wikipedia.org/wiki/Moment_si%C5%82y (dostęp w dniu 20.12.2021)
- [19] <https://www.ebmia.pl/przekladnie-planetarne-seria-sc/196073-przekladnia-planetarna-silnika-krokowego-nema-34-1-15.html> (dostęp w dniu 20.12.2021)
- [20] <http://silniki-krokowe.com.pl/informacje-techniczne/silniki-krokowe-zasada-dzialania-2/> (dostęp w dniu 20.12.2021)
- [21] <https://www.ebmia.pl/wiedza/porady/budowa-i-sterowanie-maszyn-cnc/silnik-krokowy-budowa-dzialanie-zastosowanie/> (dostęp w dniu 20.12.2021)

- [22] <https://www.ebmia.pl/715-kola-do-pasow-at5> (dostęp w dniu 20.12.2021)
- [23] <https://www.ebmia.pl/kola-do-pasow-at5/17836-kolo-zebate-21-at5-18-do-pasa-szerokosci-10mm.html> (dostęp w dniu 20.12.2021)
- [24] <https://www.automation24.pl/kompaktowa-obudowa-sterownicza-rittal-ae-1054-500-600-x-600-x-250> (dostęp w dniu 20.12.2021)
- [25] https://www.tme.eu/pl/katalog/zlacza-weipu_112972/ (dostęp w dniu 20.12.2021)
- [26] <https://www.tme.eu/pl/details/6100.3100/zlacza-iec-60320/schurter/> (dostęp w dniu 20.12.2021)
- [27] <https://motioncontrolsrobotics.com/discover-benefits-fanuc-teach-pendant/> (dostęp w dniu 20.12.2021)
- [28] <https://allegro.pl/oferta/gamepad-genesis-pv65-bezprzewodowy-do-pc-i-ps3-10467630396> (dostęp w dniu 20.12.2021)
- [29] <https://www.pjrc.com/store/teensy41.html> (dostęp w dniu 20.12.2021)

Załącznik 1

