



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

Projekt Robotu Trójnożnego Three-legged Robot

Autor: *Jakub Mazur*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr. inż. Łukasz Więckowski*

Kraków, 2022

*Serdecznie dziękuję ... tu ciąg dalszych podziękowań np. dla promotora, itd
TODO*

Spis treści

1. Wstęp.....	7
1.1. Historia robotów kroczących	7
1.2. Istniejące konstrukcje trójnożne	8
1.2.1. STriDER.....	8
1.2.2. Triped Martian	9
1.2.3. Inne konstrukcje.....	10
1.3. Cel Pracy.....	11
2. Model Matematyczny	13
2.1. Noga robota typu RRR	13
2.1.1. Kinematyka Prosta.....	13
2.1.2. Kinematyka prosta - metoda Denavita Hartenberga [8]	15
2.1.3. Kinematyka odwrotna.....	17
2.2. Cały Robot.....	22
2.2.1. Matematyka kroku	22
3. Sekcja Elektroniczna	25
4. Model i Druk 3D.....	27
5. Implementacja.....	31
5.1. Środowisko ROS [18]	32
5.1.1. Schemat implementacji.....	32
5.2. Polulu Maestro.....	32
5.3. Noga Robotyczna.....	35
5.3.1. Poprawki w interfejsach nogi robotycznej.....	37
5.4. Klasa kontrolująca trzy nogi i generator trajektorii.....	38
5.5. Plik uruchomieniowy	39
6. Algorytmy chodu.....	41
6.1. Modularna konstrukcja	41

6.2. Właściwe algorytmy chodu [21].....	41
6.2.1. Algorytm króliczych skoków.....	41
6.3. Algorytm pajęczy.....	42
7. Podsumowanie	43
7.1. Problemy konstrukcyjne i implementacyjne.....	43
7.2. Przeprowadzone eksperymenty	43
7.2.1. Zmiany czasówek.....	43
7.2.2. Długość i wysokość kroku	44
7.3. Wnioski.....	45
7.4. Przyszły rozwój projektu	46

1. Wstęp

1.1. Historia robotów kroczących

Pierwsza idea robota kroczącego pojawiła się już pod koniec XV wieku, a narodziła się w głowie nikogo innego jak Leonarda Da Vinci. Od tamtej pory wielu naukowców próbowało tworzyć konstrukcje, które używały nóg zamiast kół. Jednakże, pierwsze faktycznie udane roboty tego typu datuje się dopiero na początek lat 60 ubiegłego wieku. Pojawiły się wtedy pierwsze działające konstrukcje, na przykład robot czteronożny zbudowany przez Josepha Shingleya oraz roboty sześciu i ośmionożne zbudowane przez "Space General Corporation". [1]

Od tamtej pory powstało wiele różnych projektów, które kategoryzuje się w zależności od ilości nóg posiadanych przez robota:

- jednonożne,
- dwunożne (Humanoid, chicken-walkers),
- czteronożne (Quadrupedal),
- sześcionożne (Hexapod),
- ośmionożne,
- gąsiennicowe.

Można tu zaobserwować pewną tendencję spadkową, wraz z upływem czasu widać wzrost udanych konstrukcji o mniejszej liczbie nóg. Konstrukcje takie wymagają większej wiedzy naukowców, lepiej dobranych algorytmów ale za to można je skonstruować mniejszym nakładem materiałowym. Stąd naturalne jest dążenie do ograniczania liczby nóg w konstrukcjach robotów kroczących

Pojawia się także inna tendencja w ilości nóg robotów. Prawie wszystkie konstrukcje (poza jednonożnymi) opierają się na anatomii zwierząt. Jest to raczej logiczna tendencja, jako że do takich robotów mamy już algorytmy chodu opracowane przez miliony lat ewolucji. Biologiczne "konstrukcje", które nie mają sensu nie przetrwałyby do dziś. [1]

Co natomiast z konstrukcjami robotów trójnożnych? Można się zastanowić czy konstrukcje takie nie powstają ponieważ faktycznie nie mają sensu, czy może dlatego że temat bardziej "klasycznych", prostszych w implementacji, konstrukcji nie został po prostu jeszcze wyyczerpany przez naukowców. Jeżeli rozejrzymy się dookoła siebie możemy zaobserwować wiele przedmiotów codziennego użytku które posiadają właśnie trzy nogi, od wszelakich taboretów, przez wieszaki na ubrania po stoły. Są to jednak przedmioty statyczne i dla takich rozwiązań trzy nogi są wymaganym minimum aby dany przedmiot stał stabilnie. Co jednak z konstrukcjami dynamicznymi? Jeżeli robot trójnożny podniesie nogę, straci stabilność, zacznie się przewracać. Czyni to z niego dość ciekawą konstrukcję, gdzie w momencie stania w miejscu zachowuje się bardziej jak roboty o większej ilości nóg, nie przewraca się, a podczas ruchu zachowuje się jak roboty dwunożne, musi odpowiednio szybko odstawić nogę w odpowiednie miejsce aby się nie przewrócić.

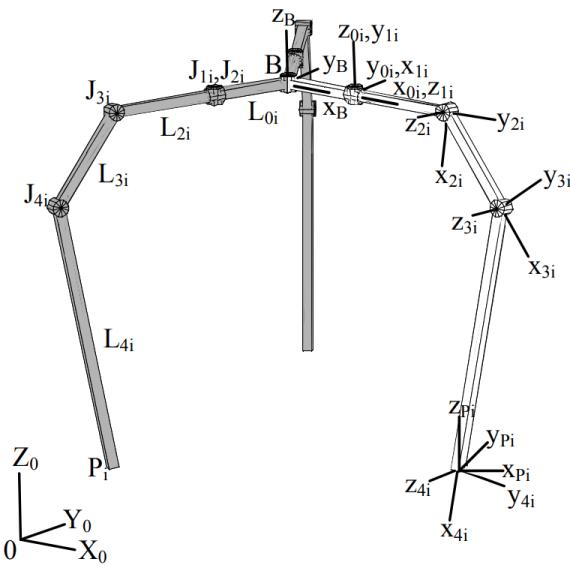
1.2. Istniejące konstrukcje trójnożne

Konstrukcje trójnożne pojawiały się w dziełach science fiction już od dawna, od "The War of the Worlds" z 1898 aż po "Mroczne Widmo" z 2001 i regularnie pojawiają się naukowcy, którzy próbują udowodnić że nie trzeba ogarniczać tego typu robotów do dzieł z obszaru science fiction.

1.2.1. STriDER

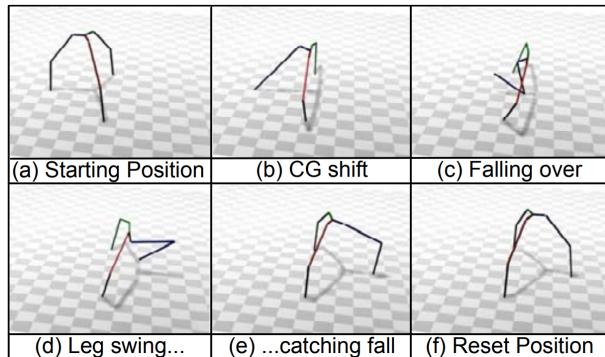
Robot STriDER (Rys. 1.1) został zbudowany w 2007 na Uniwersytecie Stanowym w Virginii. Jego celem były eksperymenty z algorytmami chodu i doclewo, prowadzenie obserwacji. Miały to umożliwić długie nogi, które znacznie podwyższały konstrukcję i sprawiały że górną platformą była idealna do instalowania wszelakiego rodzaju urządzeń typu kamery.

Konstrukcja sama w sobie posiada cztery przeguby obrotowe, jednakże na potrzeby pierwszego prototypu wykorzystano jedynie trzy z nich. Pominięty został obrót nogi dookoła osi z na "biodrzu" robota. Jako że, autorzy robota dodatkowo określają styczność nogi z podłożem jako sferyczny stopień swobody, kinematykę robota można określić jako $3 - SRRR$, a kinematykę jednej nogi jako RRR . Przy czym "pierwsze" R oznacza obrót dookoła osi x a dwa kolejne R oznaczają obrót wokół osi y . [2]



Rys. 1.1. Schemat kinematyczny robota Strider źródło: [2]

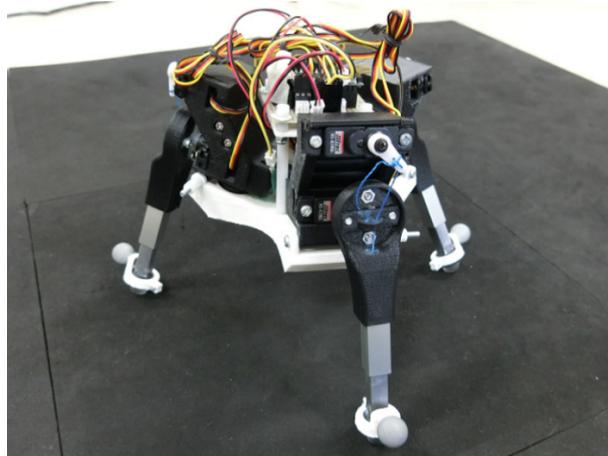
Ciekawym elementem tego robota jest sposób przemieszczania się. Robot pochyla się w kierunku, w którym ma być zrobiony krok. Po pochyleniu się, nogą znajdującej się najbardziej "z tyłu" podnosi się, robot zaczyna wywracać się w stronę kroku. Następnie robot podniesioną nogę przekłada pod sobą, aby odłożyć ją z przodu. Algorytm ten został przedstawiony na rysunku 1.2. [2]



Rys. 1.2. Algorytm chodu robota STriDER [2]

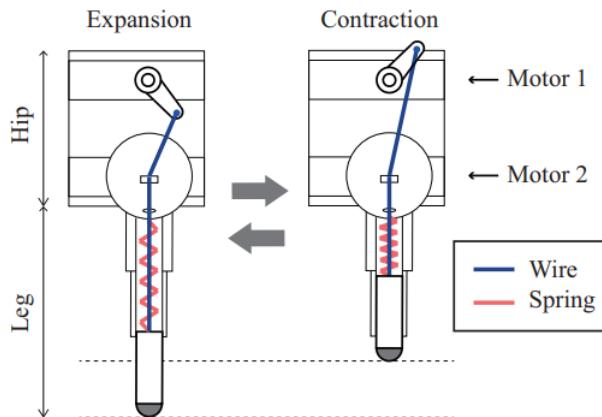
1.2.2. Triped Martian

Jest to robot zbudowany przez Yoichi Masudę na uniwersytecie w Osace w 2017 roku. Kinematyka tego robota opiera się na mechanizmie SLIP (Spring-Loaded-Inverted Pendulum). SLIP ma w pewien sposób symulować sposób poruszania się stosowany przez ludzi i zwierzęta. Sprężyna wewnętrz nogi robota jest naciągana, co powoduje skracanie się nogi, a zwalnianie



Rys. 1.3. Zdjęcie robota Martian źródło: [3]

sprężyny z powrotem wydłuża człon. Dodatkowo dodane jest serwo, które może obracać nogę dookoła osi y . Czyni to z nogi robota mechanizm o kinematyce typu RL . Został także dodany czujnik naprężenia, który jest w stanie zmierzyć siłę naciągu nici kompresującej sprężynę. [3]



Rys. 1.4. Model nogi robota Triped Martian źródło: [3]

1.2.3. Inne konstrukcje

W internecie można także znaleźć kilka różnych, niezbyt dobrze udokumentowanych konstrukcji zakończonych mniejszym lub większym sukcesem:

- Makerfaire 3-legged walking robot [4]
- Missel tripod robot [5]

Niestety konstrukcje te nie posiadają żadnej dokumentacji technicznej i nie da się dokładnie określić zasad ich działania. Nie można nawet mieć pewności że konstrukcje te faktycznie

istnieją a nie są oszustwem lub inną formą naginania rzeczywistości, na co mogłaby wskazywać jakość filmików i uboga ilość materiałów.

1.3. Cel Pracy

Celem pracy było zaprojektowanie konstrukcji robota o trzech nogach i opracowanie algorytmu pozwalającego na jego poruszanie się. Konstrukcja była częściowo wzorowana na robocie typu Hexapod "Zebulon", który od lat jest rozwijany pod kołem naukowym "Integra". Projekt ten, podobnie jak oryginalny "Zebulon" zakłada centralną platformę i równoodległe nogi o konstrukcji *RRR*. [6] Podstawową różnicą, poza ilością nóg, miała być modułarność. Robot powstały w ramach tej pracy został wykonany w technice druku 3D, aby w ramach eksperymentów z różnymi algorytmami chodu można było łatwo modyfikować i przedrukowywać kolejne człony nóg. Wspominana modułarność rozciąga się także na oprogramowanie - które to zostało napisane w środowisku ROS 2. Na potrzeby implementacji policzona także została kinematyka prosta i odwrotna nogi robotycznej, jak i został zamodelowany matematycznie krok robota. Dodatkowo w ramach tej pracy opracowany został uproszczony algorytm chodu.

2. Model Matematyczny

2.1. Noga robota typu RRR

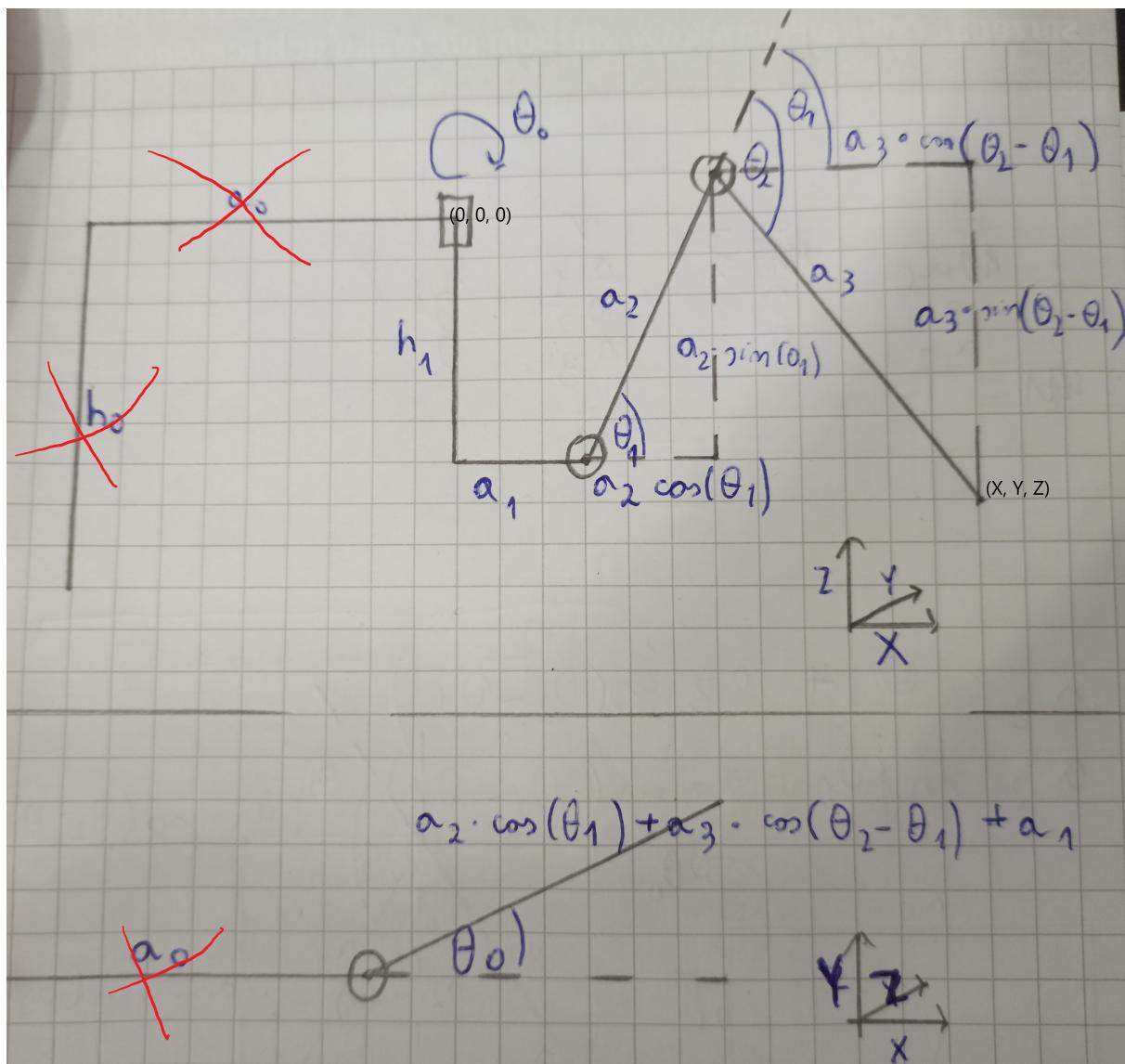
Noga ma trzy stopnie swobody. Wszystkie są typu obrotowego, co czyni z niej konstrukcję typu *RRR*. Przy czym dwa elementy rotacyjne obracają się dookoła osi poziomej (*X*) (podobnie jak w przypadku robota STriDER), a jedna dookoła osi pionowej (*Z*). Są to też te same osi obrotu co w przypadku ramienia robotycznego typu antromorficznego (zwanego także "angular" bądź "jointed"). Co za tym idzie, cały model matematyczny jest w zasadzie identyczny jak w przypadku manipulatora tego typu.

2.1.1. Kinematyka Prosta

Obliczanie kinematyki prostej polega na uzyskaniu równań końcówki ramienia robotycznego we współrzędnych kartezjańskich względem współrzędnych konfiguracyjnych. Inaczej mówiąc, wejściem algorytmu jest zbiór współrzędnych konfiguracyjnych, a na wyjściu otrzymamy współrzędne kartezjańskie.[7]

W przypadku tego konkretnego manipulatora, mamy do czynienia z trójwymiarowym układem współrzędnych kartezjańskich i trzema stopniami swobody. Da to liniowo niezależny układ trzech równań, w którym parametrami będą kąty na jakich mają ustawić się serwomechanizmy a wynikiem wektor współrzędnych kartezjańskich.

Najprostszą metodą liczenia kinematyki prostej jest rozrysowanie modelu matematycznego i geometryczne wyprowadzenie potrzebnych równań. Model taki dla tego manipulatora został przedstawiony na rysunku 2.1. Przyjęty początek układu współrzędnych został oznaczony jako $(0, 0, 0)$, a punkt którego współrzędne kartezjańskie są poszukiwane został oznaczony jako (X, Y, Z) . h_1 i a_{1-3} to stałe długości poszczególnych członów nogi. Natomiast θ_{0-2} to właśnie pozycje serwomechanizmów - parametry algorytmu. Na ich podstawie zostaną obliczone współrzędne końcówki manipulatora w systemie kartezjańskim. Same obliczenia



Rys. 2.1. Model Matematyczny

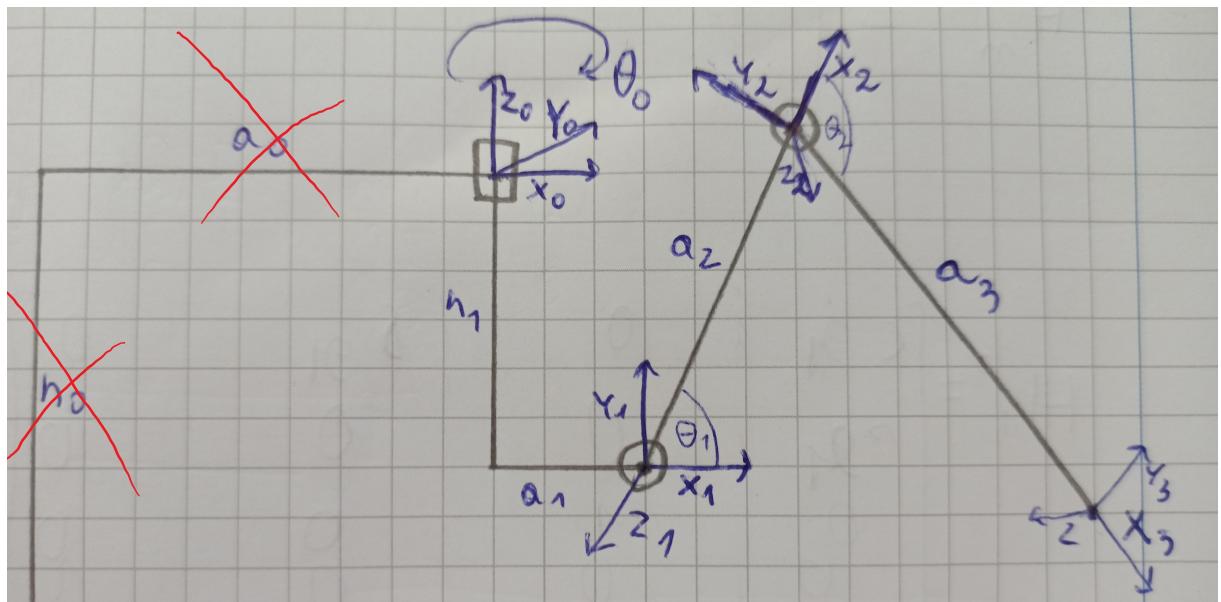
geometryczne są już w tym momencie dość trywialne, wystarczy do każdego członu a_{1-3} przenieść jego długość na osią X , Y lub Z za pomocą trygonometrii (sinus lub cosinus) i zsumować odpowiednie długości. Da to układ równań 2.1.

$$\begin{aligned}
 a_{temp} &= a_2 \cos \theta_1 + a_3 \cos (\theta_2 - \theta_1) + a_1 \\
 Y &= a_{temp} \cdot \sin \theta_0 \\
 X &= a_{temp} \cdot \cos \theta_0 \\
 Z &= a_2 \sin \theta_1 - a_3 \sin (\theta_2 - \theta_1)
 \end{aligned} \tag{2.1}$$

2.1.2. Kinematyka prosta - metoda Denavita Hartenberga [8]

Alternatywą dla zwykłych obliczeń geometrycznych jest metoda Denavita Hartenberga. Polega ona na przedstawieniu całkowitego przekształcenia jako iloczynu przekształceń jednorodnych kolejnych członów. Pojedyncze przekształcenie jednorodne składa się natomiast z sześciu przekształceń prostych (3 dla rotacji i 3 dla przesunięć). Wymnożenie tych przekształceń prostych da przekształcenie jednorodne. [9]

Metoda ta jest w szczególności użyteczna dla bardzo skomplikowanych manipulatorów, gdzie stopni swobody jest znacznie więcej niż ilość współrzędnych kartezjańskich.



Rys. 2.2. Model Denavit Hartenberg

Z praktycznego punktu widzenia obliczenia należy zacząć od stworzenia specjalnego rysunku (Rys. 2.2) z zaznaczonymi kolejnymi obrotami lokalnych układów współrzędnych, a następnie zebrać odpowiednie transformacje do tabelki (Tab. 2.1)

Joint i	θ_i	α_i	r_i	d_i
1	θ_0	$\frac{\pi}{2}$	a_1	h_1
2	θ_1	0	a_2	0
3	θ_2	0	a_3	0

Tabela 2.1. Tabela z parametrami DH

Legenda tabelki 2.1:

θ_i - kąt pomiędzy x_{n-1} i x_n , mierzony dookoła z_{n-1}

α_i - kąt pomiędzy z_{n-1} to z_n , mierzony dookoła x_n

r_i - odległość od początku ramki $n - 1$ a początkiem ramki n wzdłuż kierunku x_n

d_i - odległość od x_{n-1} do x_n wzdłuż kierunku z_{n-1}

Następnie macierze transformacji jednorodnej (pomiędzy ramkami $n - 1$ i n) oblicza się zgodnie ze wzorem 2.2. Wzór ten jest właśnie obliczony poprzez wymnożenie wzorów na wyżej wspomniane sześć przekształceń prostych. Ale w zasadzie wzór ten sprowadza się do dwóch zasadniczych elementów. R jest macierzą Rotacji o rozmiarze 3×3 a T to macierz Transformacji 1×3 . [10]

$$H_n^{n-1} = \left[\begin{array}{c|c} R & T \\ \hline 0 & 0 & 0 & 1 \end{array} \right] = \left[\begin{array}{cccc} \cos \theta_i & -\sin \theta_i \cdot \cos \alpha_i & \sin \theta_i \cdot \sin \alpha_i & a_i \cdot \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cdot \cos \alpha_i & -\cos \theta_i \cdot \sin \alpha_i & a_i \cdot \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{array} \right] \quad (2.2)$$

Następnie należy podstawić wartości dla każdego z trzech punktów i otrzymane macierze wymnożyć zgodnie ze wzorem 2.3. Mnożenie to zostało wykonane przy pomocy skryptu napisanego w programie Matlab.

$$H_n^{n-1} = H_1^0 \cdot H_2^1 \cdot H_3^2 =$$

$$\left[\begin{array}{cccc} c\theta_0 \cdot (c\theta_1 \cdot c\theta_2 - s\theta_1 \cdot s\theta_2) & -c\theta_0 \cdot (c\theta_1 \cdot s\theta_2 + s\theta_1 \cdot c\theta_2) & s\theta_0 & c\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot (c\theta_1 \cdot c\theta_2 - s\theta_1 \cdot s\theta_2)) \\ s\theta_0 \cdot (c\theta_1 \cdot c\theta_2 - s\theta_1 \cdot s\theta_2) & -s\theta_0 \cdot (c\theta_1 \cdot s\theta_2 + s\theta_1 \cdot c\theta_2) & -c\theta_0 & s\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot (c\theta_1 \cdot c\theta_2 - s\theta_1 \cdot s\theta_2)) \\ c\theta_1 \cdot s\theta_2 + s\theta_1 \cdot c\theta_2 & c\theta_1 \cdot c\theta_2 - s\theta_1 \cdot s\theta_2 & 0 & h_1 + a_2 \cdot s\theta_1 + a_3 \cdot (c\theta_1 \cdot s\theta_2 + s\theta_1 \cdot c\theta_2) \\ 0 & 0 & 0 & 1 \end{array} \right] =$$

$$\left[\begin{array}{cccc} c\theta_0 \cdot c(\theta_1 + \theta_2) & -c\theta_0 \cdot s(\theta_1 + \theta_2) & s\theta_0 & c\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot c(\theta_1 + \theta_2)) \\ s\theta_0 \cdot c(\theta_1 + \theta_2) & -s\theta_0 \cdot s(\theta_1 + \theta_2) & -c\theta_0 & s\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot c(\theta_1 + \theta_2)) \\ c\theta_1 \cdot s\theta_2 + s\theta_1 \cdot c\theta_2 & c(\theta_1 + \theta_2) & 0 & h_1 + a_2 \cdot s\theta_1 + a_3 \cdot s(\theta_1 + \theta_2) \\ 0 & 0 & 0 & 1 \end{array} \right] \quad (2.3)$$

Następnie, aby otrzymać właściwe równania, należy z równania 2.3 wziąć część odpowiedzialną za transformację. Efektem jest wzór 2.4, czyli finalna wersja równań kinametyki prostej obliczonej metodą Denavita Hartenberga

$$\begin{aligned} X &= c\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot c(\theta_1 + \theta_2)) \\ Y &= s\theta_0 \cdot (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot c(\theta_1 + \theta_2)) \\ Z &= h_1 + a_2 \cdot s\theta_1 + a_3 \cdot s(\theta_1 + \theta_2) \end{aligned} \quad (2.4)$$

Można jeszcze przetworzyć to równanie do takiej postaci w jakiej zapisane jest równanie 2.1. Otrzymamy wtedy równanie 2.5.

$$\begin{aligned} a_{temp} &= (a_1 + a_2 \cdot c\theta_1 + a_3 \cdot c(\theta_1 + \theta_2)) \\ X &= \cos \theta_0 \cdot a_{temp} \\ Y &= s\theta_0 \cdot a_{temp} \\ Z &= h_1 + a_2 \cdot s\theta_1 + a_3 \cdot s(\theta_1 + \theta_2) \end{aligned} \quad (2.5)$$

TODO - znak pomiędzy obliczeniami geometrycznymi a DH się nie zgadza

2.1.3. Kinematyka odwrotna

Kinematyka odwrotna jest natomiast - jak sama nazwa wskazuje - procesem odwrotnym do kinematyki prostej. Polega na obliczeniu zestawu współrzędnych konfiguracyjnych na podstawie współrzędnych kartezjańskich końcówki manipulatora. Czyli inaczej mówiąc - wejściem algorytmu są współrzędne kartezjańskie, a wyjściem współrzędne konfiguracyjne.

Zwykle odwrotną kinematykę można obliczyć poprzez rozwiązywanie równań kinematyki prostej. Jest to w tym przypadku także teoretycznie możliwe, jako że mamy do czynienia z trzema niewiadomymi (θ_{0-2}) i układem trzech równań nr. 2.1 (lub 2.5). (Równanie na a_{temp} jest jedynie pomocnicze, trzeba je traktować jak część równań Y i X).

2.1.3.1. Obliczenia θ_0

Jest to jak najbardziej wykonalne w przypadku θ_0 , można to zrobić łącząc wzór na X i Y , dzieląc go obustronnie, zamieniając $\frac{\sin}{\cos}$ na tan i wyciągając θ_0 na lewą stronę. Daje to wzór 2.6.

$$\theta_0 = \arctan \frac{Y}{X} \quad (2.6)$$

2.1.3.2. Obliczenia θ_2

Większy problem pojawia się jednak w przypadku obliczania θ_1 i θ_2 , ponieważ wartości te da się obliczyć z wyżej wspomnianego równania, ale nie da się wyznaczyć na te wartości równań w postaci jawnej. A bez postaci jawnej poprawna implementacja tych równań będzie znacznie utrudniona.

Można natomiast zastosować pewnego rodzaju uproszczenie. Jeżeli wyznaczanie równań θ_1 i θ_2 sprowadzi się do problemu dwuwymiarowego, obliczenia stają się w zasadzie identyczne jak w przypadku obliczeń kinematyki odwrotnej dla robota typu SCARA, co było już robione wielokrotnie. [11]

Aby obliczyć kąt θ_2 można powrócić do rysunku 2.1 i zastosować na trójkącie utworzonym z a_2 i a_3 twierdzenie cosinusów połączone z twierdzeniem pitagorasa. Daje to wzór 2.7. [12]

$$(x - a_1)^2 + (z + h_1)^2 = a_2^2 + a_3^2 - 2 \cdot a_2 \cdot a_3 \cdot \cos(180^\circ - \theta_2) \quad (2.7)$$

Następnie na podstawie równania 2.7 aby ostatecznie otrzymać θ_2 należy zastosować wzór $\cos(180^\circ - \theta) = -\cos \theta$ i za pomocą prostych przekształceń wyciągnąć z równania θ_2 . Daje to wzór 2.8 [12]

$$\theta_2 = \arccos \left(\frac{(x - a_1)^2 + (z + h_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \right) \quad (2.8)$$

2.1.3.3. Obliczenia θ_1

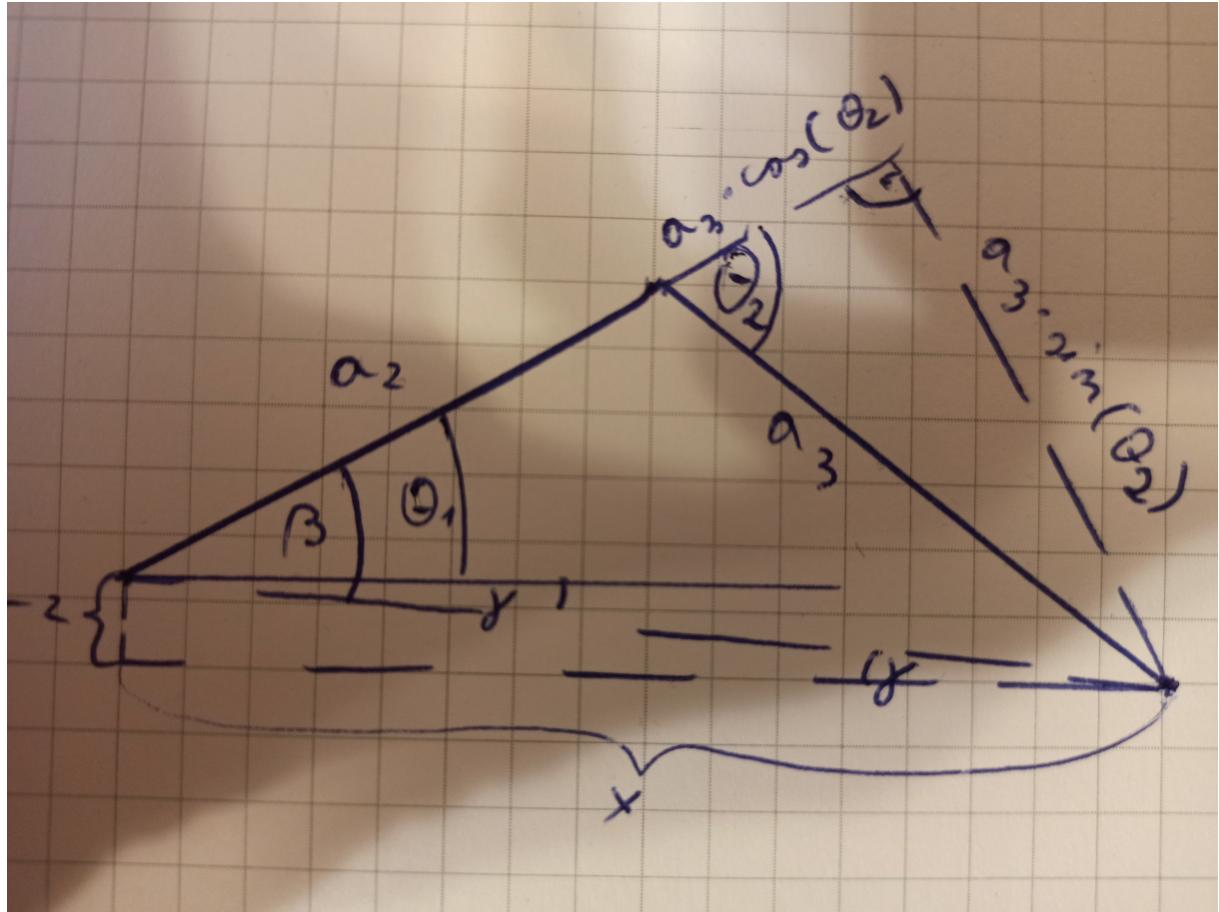
Obliczenia dla θ_1 są natomiast trochę bardziej skomplikowane. Aby dało się je prosto zvisualizować, wykonany został rysunek 2.3. Na podstawie rysunku, stosując wzory trygonometryczne, można otrzymać zależności 2.9 [12]

$$\begin{aligned} \beta &= \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2} \\ \gamma &= \arctan \frac{-\Delta z}{\Delta x} = -\arctan \frac{\Delta z}{\Delta x} \\ \theta_1 &= \beta - \gamma \end{aligned} \quad (2.9)$$

Do rozwiązania układu 2.9 potrzebne są jeszcze Δz i Δx . Na podstawie rysunku 2.1 można stwierdzić że $\Delta z = Z + h_1$ a $\Delta x = X - a_1$. Podstawiając te dane pod równanie drugie układu 2.9 i podstawiając równania pierwsze i drugie pod odpowiednie kąty z równania trzeciego wspomnianego układu otrzymujemy ostateczne rozwiązanie na θ_1 - równanie 2.10.

$$\theta_1 = \arctan \frac{z + h_1}{x - a_1} + \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2} \quad (2.10)$$

Jednakże należy zaadresować także pewien problem który pojawi się w równaniu 2.10. Należy zauważać że dla $\theta_2 > 90^\circ$ człon $a_3 \cdot \cos \theta_2$ zacznie przyjmować wartości ujemne. Dalej zwiększając ten kąt, możemy osiągnąć sytuację, gdzie $-a_3 \cdot \cos \theta_2 > a_2$. Wtedy cały mianownik równania $\frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2}$ zmieni znak z dodatniego na ujemny. Pewien kąt graniczny, dla którego ta zmiana znaku nastąpi, można policzyć bardzo prosto - należy przyrównać



Rys. 2.3. Fragment modelu nogi do obliczeń θ_{2g}

mianownik do 0. Otrzymamy wtedy równanie 2.11.

$$\theta_{2g} = \arccos \left(-\frac{a_2}{a_3} \right) \quad (2.11)$$

Aby obliczany kąt θ_1 pomimo "przekręcenia" się znaku dalej poprawnie zwiększał się, należy zastosować pewnego rodzaju oszustwo. dla wartości kąta $\theta_2 > \theta_{2g}$, równanie $\theta_{2g} = \beta - \gamma$ przyjmie postać 2.12.

$$\theta_1 = -\gamma + 2 \cdot \beta_{\theta_2 - \theta_{2g}} + \beta \quad (2.12)$$

Przy czym zachodzi także równanie 2.13.

$$\begin{aligned}
 \beta_{\theta_2 \rightarrow \theta_{2g}} &= \\
 \lim_{\theta_2 \rightarrow \theta_{2g}} \left(\arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2} \right) &= \\
 \arctan \frac{a_3 \cdot \sin \theta_2}{0} &= \\
 \arctan \infty &= \frac{\pi}{2}
 \end{aligned} \tag{2.13}$$

Zbierając wszystkie możliwe przypadki i wykonując odpowiednie podstawienia, otrzymamy ostateczne równanie na θ_1 - równanie 2.14.

$$\theta_1 = \begin{cases} \arctan \frac{z+h_1}{x-a_1} + \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2}, & \text{jeżeli } \theta_2 < \theta_{2g} \\ \arctan \frac{z+h_1}{x-a_1} + \frac{\pi}{2}, & \text{jeżeli } \theta_2 = \theta_{2g} \\ \arctan \frac{z+h_1}{x-a_1} + \pi + \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2}, & \text{jeżeli } \theta_2 > \theta_{2g} \end{cases} \tag{2.14}$$

2.1.3.4. Gotowy układ równań

Ostatecznie zbierając równania 2.6, 2.14, 2.8 otrzymujemy układ równań 2.15 który stanowi uproszczoną kinematykę odwrotną opisywanej nogi robotycznej.

$$\begin{aligned}
 \theta_0 &= \arctan \frac{Y}{X} \\
 \theta_1 &= \begin{cases} \arctan \frac{z+h_1}{x-a_1} + \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2}, & \text{jeżeli } \theta_2 < \theta_{2g} \\ \arctan \frac{z+h_1}{x-a_1} + \frac{\pi}{2}, & \text{jeżeli } \theta_2 = \theta_{2g} \\ \arctan \frac{z+h_1}{x-a_1} + \pi + \arctan \frac{a_3 \cdot \sin \theta_2}{a_2 + a_3 \cdot \cos \theta_2}, & \text{jeżeli } \theta_2 > \theta_{2g} \end{cases} \\
 \theta_2 &= \arccos \left(\frac{(x-a_1)^2 + (z+h_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \right)
 \end{aligned} \tag{2.15}$$

2.1.3.5. Ograniczenia ruchu

Z punktu widzenia czysto matematycznego ruch będzie ograniczony przez torus. Równanie 2.16 opisuje tą figurę - R oznacza promień okręgu dookoła osi z , a \sqrt{r} oznacza promień "wałka".

$$z = \sqrt{r - (R - \sqrt{x^2 + y^2})^2} \tag{2.16}$$

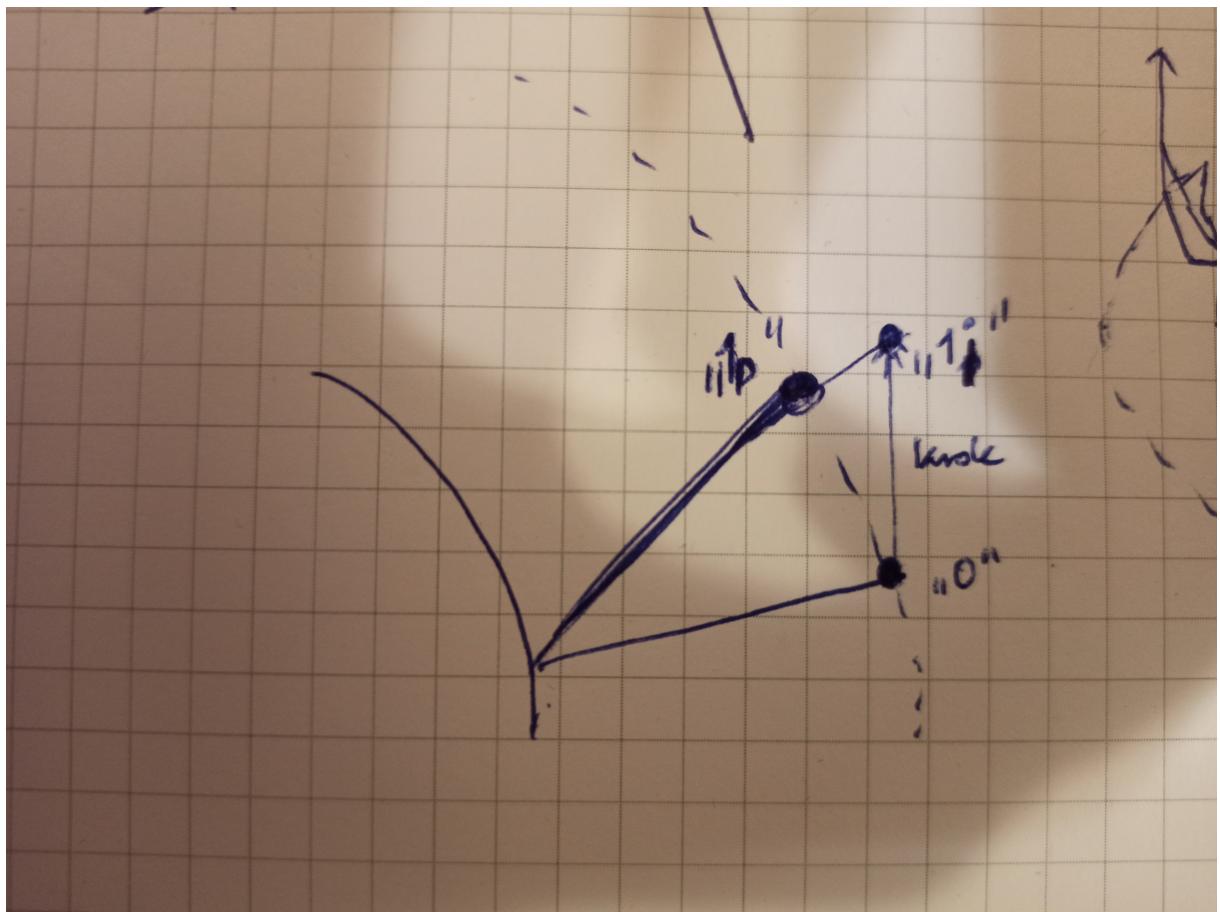
Aby otrzymać właściwe równanie specyficzne dla tego projektu, należy zastosować podstawienia przedstawione na równaniu 2.17.

$$\begin{aligned} R &= a_1 \\ r &= \sqrt{a_2 + a_3} \end{aligned} \tag{2.17}$$

Po wykonaniu podstawienia otrzymamy równanie 2.18.

$$z = \sqrt{a_1 - (\sqrt{a_2 + a_3} - \sqrt{x^2 + y^2})^2} \tag{2.18}$$

2.1.3.6. Przybliżenia



Rys. 2.4. Przedstawienie błędu spowodowanego zastosowaniem modelu robota SCARA przy kinematyce odwrotnej

Natomiast trzeba cały czas pamiętać, że całe obliczenia kinematyki odwrotnej są tylko uproszczeniem, które będzie owocować pewnymi drobnymi błędami. Należy zauważyć, że zmiana współrzędnej Y nie wpływa na wartości kątów θ_1 i θ_2 , tylko na θ_0 . Oznacza to, że wraz ze zmianą kąta θ_0 , na potrzeby uproszczenia obliczeń, obraca się także o ten kąt oś X układu współrzędnych. W praktyce oznacza to, że odsunięcie nogi od punktu $(0, 0, 0)$ zawsze będzie rzutowane na cylinder o promieniu równym odsunięciu o takim samym X i Z , ale $Y = 0$. Problem jest zaprezentowany na rysunku 2.4. Widać tam pozycję początkową robota oznaczoną

jako "0". Następnie jeśli robot wykona krok, to stosując ten algorytm, znajdzie się w pozycji "1p", a w idealnym przypadku powinien znaleźć się w "1i". Jednakże błąd powinien być pomijalnie mały a równania powinny umożliwić skuteczną implementację algorytmu chodu.

2.2. Cały Robot

Zwykle konstruując roboty wzorujemy się na zwierzętach występujących w naturze. W tym przypadku nie mamy jednak tego luksusu, dlatego na początek należy przyjąć jakieś najbardziej intuicyjne założenie. Dlatego też uznałem że najlepszym rozwiązaniem będzie rozmieścić nogi na jednej płaszczyźnie w równych odstępach - co 120 deg.

W poprzednim podrozdziale przyjąłem osie układu relatywne do ułożenia początkowego nogi robota. Oznacza to że robot trójnożny będzie miał trzy niezależne osie x i trzy niezależne osie y , tylko oś z zgadza się między kolejnymi nogami. Idzie za tym konieczność stworzenia pewnej metody "obracania" wszystkich tych osi x i y do jednego zunifikowanego układu współrzędnych. Pozwoli to obliczyć kinematykę całego robota.

2.2.1. Matematyka kroku

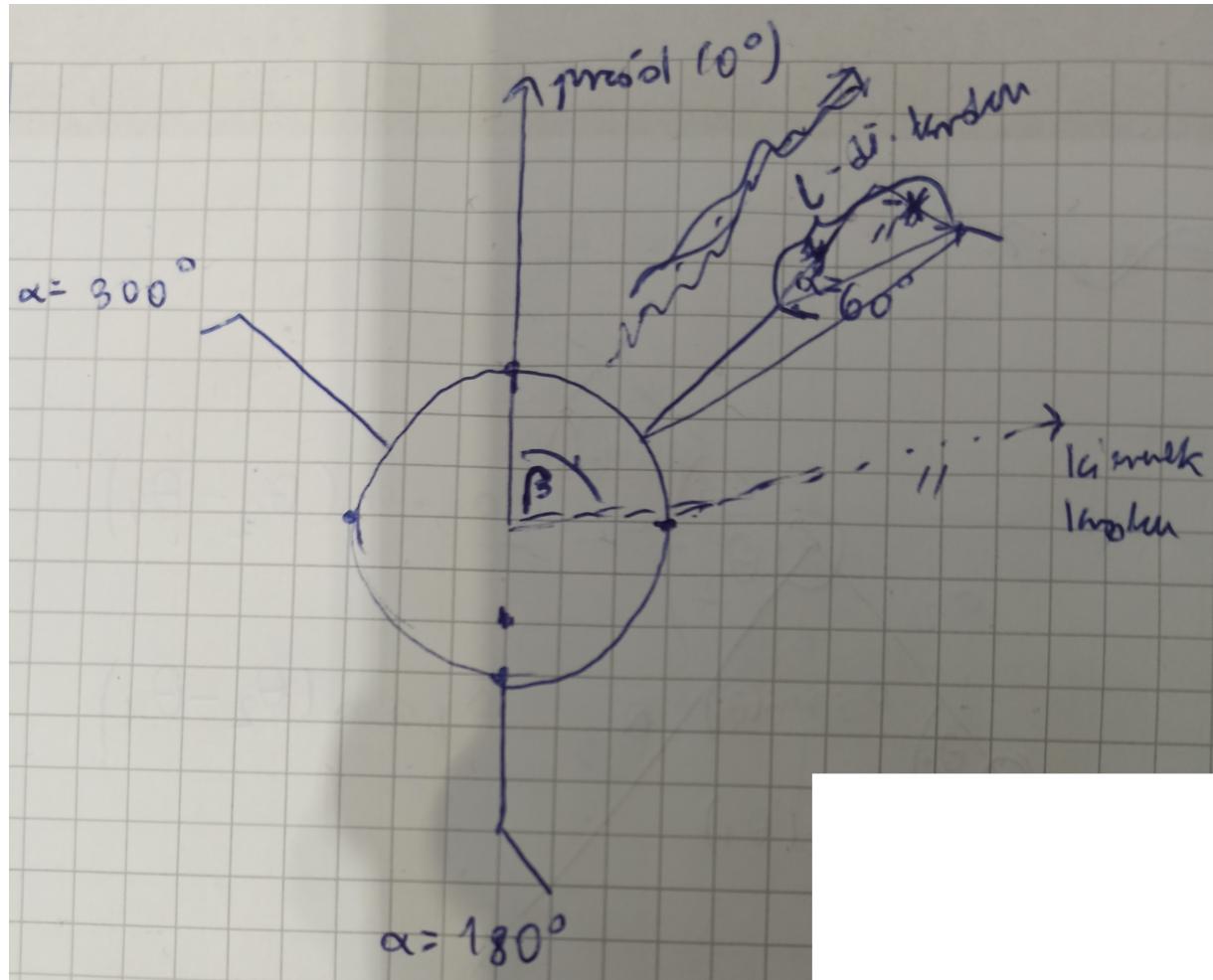
W celu uproszczenia zarówno obliczeń jak i późniejszej generacji kolejnych kroków algorytmu chodu, można pominąć liczenie całej kinematyki prostej i odwrotnej całego robota. Zamiast tego wystarczy pojedynczy krok odpowiednio sparametryzować. Jeżeli dla każdego kroku pojedynczej nogi przyjmiemy:

- kąt α na "tarczy" robota, na którym ustawniona jest noga,
- kąt β względem "przodu" robota, w którą ma zostać wykonany krok,
- długość kroku l , równolegle do osi kroku robota,

to możemy łatwo otrzymać algorytm który z tych dwóch zmiennych i jednej stałej da zmianę pozycji końcówki robota ($\Delta x, \Delta y$) we współrzędnych kartezjańskich względem układu zmiennych nogi robota.

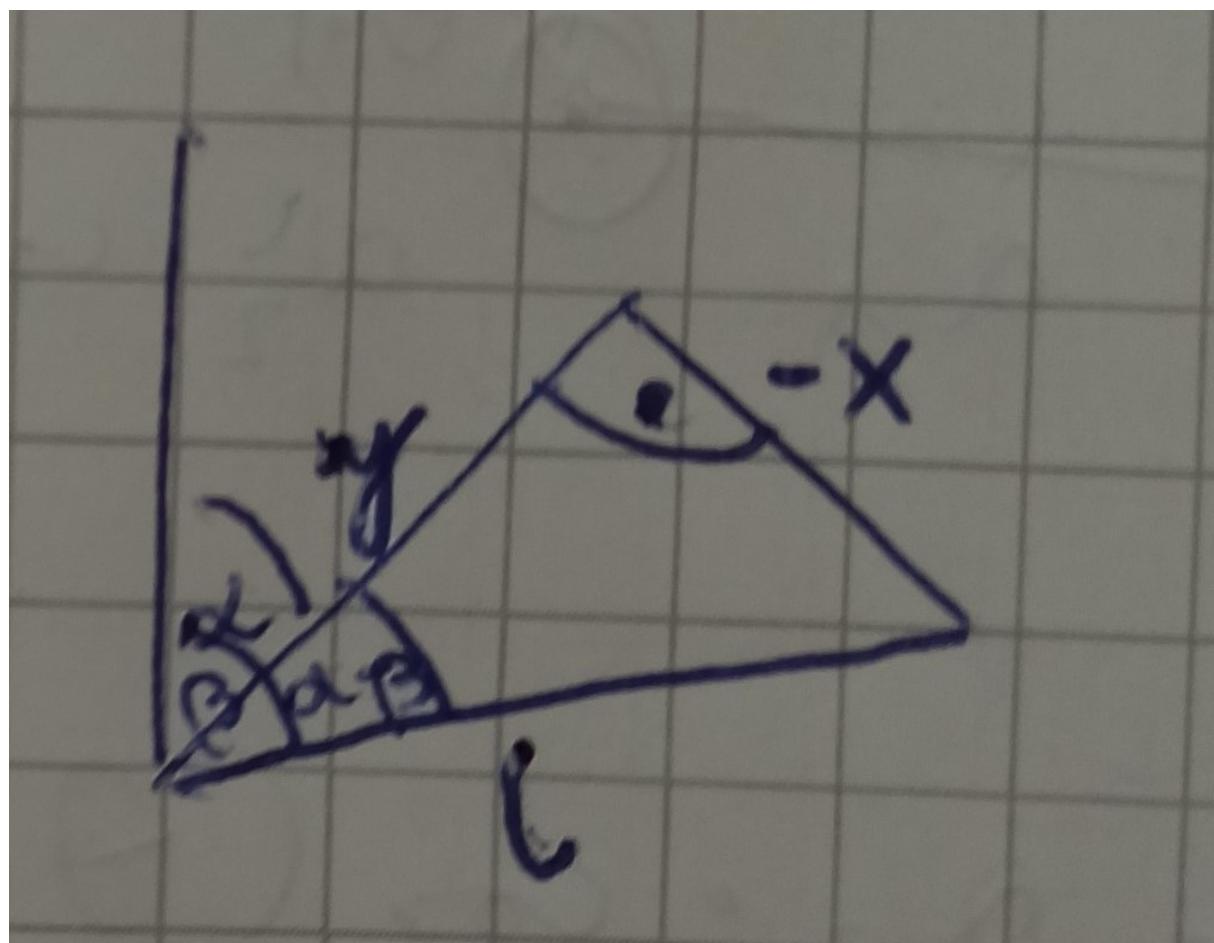
Zostało to przedstawione na rysunku 2.5 na przykładzie nogi na pozycji $\alpha = 60^\circ$. Na wspomnianym rysunku widać przód robota oznaczony jako 0° , kierunek kroku odsunięty o kąt β od osi "przodu" robota i wynikające z tego kroku przestawienie nogi. Przestawienie to składa się z długości kroku l i dwóch pozycji nogi - przed krokiem i po kroku. Wynikające z tego zależności geometryczne zostały wyizolowane na rysunku 2.6. Na podstawie tych zależności można napisać równanie 2.19

$$\begin{aligned}\Delta x &= l \cdot \cos(\beta - \alpha) \\ \Delta y &= l \cdot \sin(\beta - \alpha)\end{aligned}\tag{2.19}$$



Rys. 2.5. Schemat matematyczny wykonywania kroku

Równanie 2.19 mówi tyle, że wykonanie nogą na pozycji α kroku o długości l w kierunku β wymusza przestawienie końcówki nogi o $(\Delta x, \Delta y)$



Rys. 2.6. Wyizolowane zależności geometryczne w czasie wykonywania kroku

3. Sekcja Elektroniczna

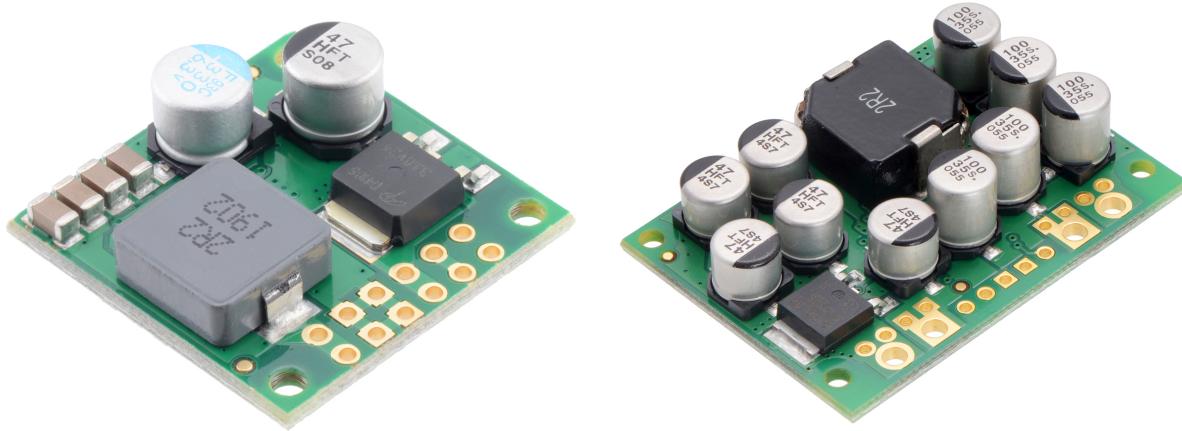
Projekt elektroniczny na potrzeby tej pracy został ograniczony do minimum i był częściowo wzorowany na sekcji elektronicznej projektu Zebulon 2.0. Zasilanie składa się z akumulatora typu LiPo i minimum dwóch przetwornic. Jako mózg urządzenia zastosowany został mini-komputer Raspberry Pi 4B i to do jego zasilenia potrzebna jest jedna z przetwornic. Zgodnie z dokumentacją Raspberry zasilanie jest ze źródła o napięciu 5V i prądzie przynajmniej 3A.[13] Dlatego została wybrana przetwornica D36V28F5 (Pololu 3782). Natomiast ze względu na brak dostępności zakupiono i zainstalowano przetwornicę D36V50F5 (Pololu 4091) (Rys. 3.2, po lewej), która to niewiele różni się ceną, ale za to ma lepszy prąd wyjściowy. [14]



Rys. 3.1. Serwomechanizmy wybrane do projektu. [15] [16]

Druga przetwornica ma za zadanie zasilić serwomechanizmy. Serwomechanizmy wybrane do tego projektu to Feetech FT5715M (sztuk 3) i PowerHD LF-20MG (sztuk 6). (Rys. 3.1) W czasie zakupu serwa te wypadały najlepiej spośród wszystkich dostępnych pod kątem prędkości ruchu do ceny. Serwa firmy Feetech są zasilanie napięciem z zakresu 4.8 do 6V [15] a Power HD napięciem 4.8 do 6.6V [16]. Dlatego jako wspólne napięcie zasilania ustalona została wartość

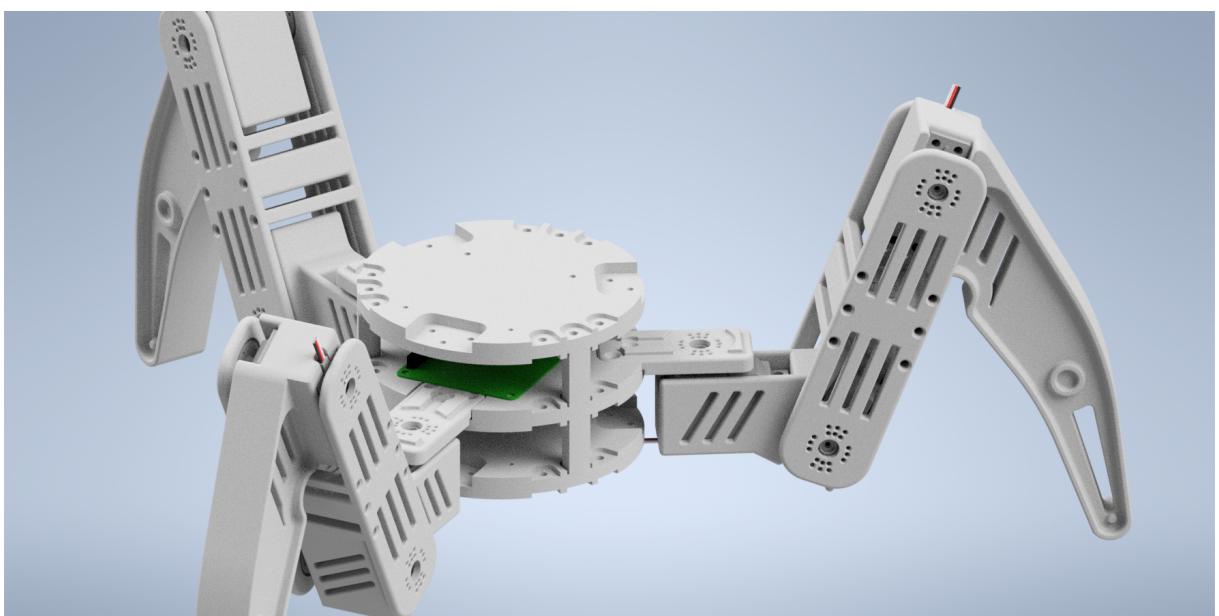
6V. Jako że, zwykle serwo pobiera do około 1A prądu, to do ich zasilenia potrzeba przetwornicy o wyjściu 6V i minimum 9A [17]. Natomiast należy pamiętać, że dobieranie przetwornicy "na styk" przy czymś takim jak zasilanie serwomechanizmów może spowodować później problemy przy większych obciążeniach. Dlatego, aby uwzględnić pewien zapas prądowy, wybrana została przetwornica D24V150F6 (Pololu 2882). (Rys. 3.2, po prawej) Ma ona aż 15A prądu wyjściowego, co jest znacznie więcej niż wymagane 9A. Jednakże na chwilę obecną firma Pololu nie oferuje żadnych przetwornic o napięciu wyjściowym 6V i wydajności prądowej pomiędzy 9A a 15A. Dlatego została wybrana najmniejsza dostępna przetwornica spełniająca wymóg dziewięciu lub więcej amperów prądu wyjściowego. [14]



Rys. 3.2. Zakupione przetwornice [14]

4. Model i Druk 3D

Jednym z głównych założeń projektu jaki powstał w ramach tej pracy (przedstawiony na rys. 4.1) była możliwość wydrukowania całej konstrukcji w 3D. Miało to na celu znaczne obniżenie kosztów produkcji, ale przede wszystkim umożliwienie znacznie szybszych przeróbek. Jest to o tyle istotne, że prowadzone będą badania z algorytmami chodu - w przypadku większości algorytmów wydłużenie pewnych elementów nogi może zmniejszyć wymagane prędkości ruchu serw. Eksperymenty takie mogą wymusić liczne przedruki poszczególnych członów nóg robota.



Rys. 4.1. Model złożeniowy.

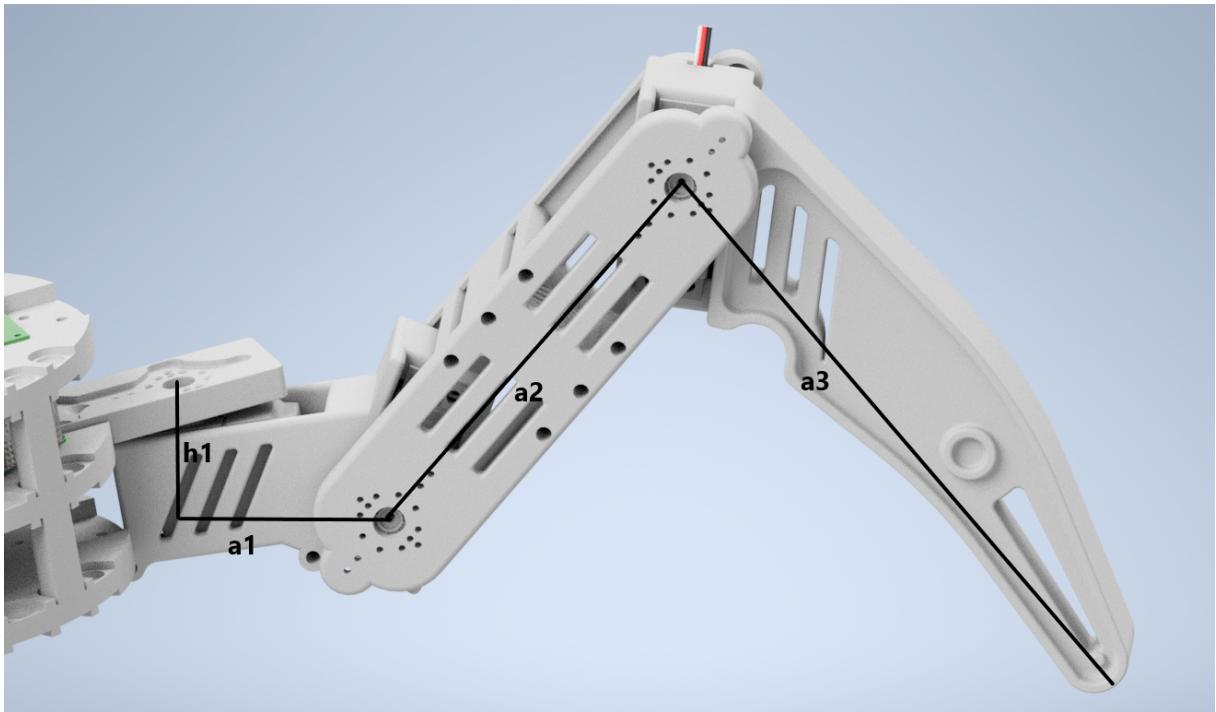
Drugim założeniem projektowym jest modularność. Bardzo podobną modularność oferuje robot TurtleBot 3, który także był inspiracją stojącą za konstrukcją. Turtlebot składa się z wielu identycznych warstw, które zawierają liczne otwory montażowe umożliwiające przykręcenie w zasadzie dowolnego elementu. Projekt robiony w ramach tej pracy został oparty o podobną ideę. Także składa się z wielu identycznych warstw zawierających liczne otwory montażowe. Otwory te zostały dobrane pod kątem elektroniki jaka będzie montowana, ale na każdej z

warstw można zamontować dowolny element w jednej z kilku konfiguracji.

Zostały także dokładnie zwymiarowane otwory montażowe znajdujące się na orczykach do zakupionych serw i przeniesione na poszczególne elementy nóg. Do montażu wspomnianych orczyków zakupione zostały śruby $M1.6$, co stanowi swojego rodzaju drobny eksperyment. Zwykle orczyki montuje się za pomocą kleju lub wkrętów dostarczanych wraz z serwomechanizmem. Są to jednak metody przynajmniej częściowo destrukcyjne - nie umożliwiają szybkiego demontażu i wymiany elementów, co jest bardzo ważnym elementem tego projektu. Zastosowanie śrub z nakrętkami rozwiązuje ten problem - jednak pojawia się pytanie czy nie generuje to innych problemów. Bardzo prawdopodobne jest pojawienie się problemu z samoodkręcającymi się śrubkami przy dłuższym użytkowaniu - zjawisko spowodowane drganiami generowanymi przez serwa. Było to już problemem w przypadku innych projektów - gdzie serwa odkręcały się od orczyków. Jednakże projekty tamte wykonane były w technologii CNC z blachy lub włókna węglowego - nigdy nie były drukowane w 3D. Bardzo możliwe jest że filament wytlumi drgania.

Projekt ten zawiera także pewną wadę konstrukcyjną - jest to bardzo cienki element łączący nogę z tułowiem (element nogi zero). Istnieje duże ryzyko gięcia a może nawet łamania się wyżej wymienionego elementu. Aby zmniejszyć to ryzyko, wspomniany element został miejscami pogrubiony. Dodatkowo, zakup metalowych orczyków także mógłby zmniejszyć ryzyko gięcia się konstrukcji. Najtrwalszym rozwiązaniem jednak byłoby stworzenie dodatkowego elementu który byłby przytwierdzony do dolnej części obudowy i posiadałby oś obrotu z pierwszym elementem nogi - "podtrzymywałby" ten element od dołu.

Całość konstrukcji została zaprojektowana w programie Autodesk Inventor. Program ten został wybrany tylko i wyłącznie ze względu na fakt, że był autorowi projektu dość dobrze znany. Podczas projektowania należało także zdecydować się na pewne długości poszczególnych członów robota. Centralny okrąg został zaprojektowany na bazie RPi, można powiedzieć że jest to lekko powiększony okrąg opisany na prostokącie nakreślonym przez PCB Raspberry Pi. Długości członów nóg natomiast, zostały dobrane tak, aby przede wszystkim zmieściły się serwa i elementy nie obijały się o siebie podczas normalnej pracy. W tym miejscu, dla lepszej wizualizacji tworzonych modeli matematycznych, można także przenieść wymiary z rysunku 2.1 na zaprojektowany model. Da to rysunek 4.2. Wykonanie projektu umożliwia także przypisanie konkretnych wartości do parametrów z rozdziału 2. Przypisanie to jest widoczne w równaniu 4.1.



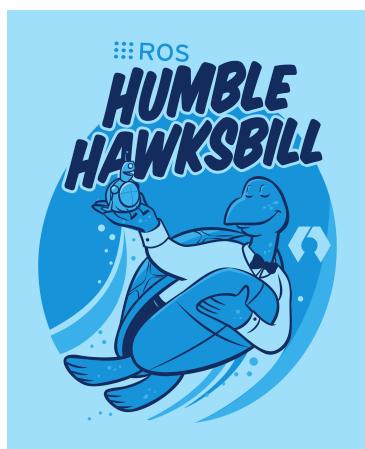
Rys. 4.2. Model złożeniowy pojedynczej nogi z naniesionymi wymiarami.

$$\begin{aligned}
 h_1 &= 40 \\
 a_1 &= 55 \\
 a_2 &= 125 \\
 a_3 &= 180
 \end{aligned} \tag{4.1}$$

Modele zostały wydrukowane na drukarce Zortrax M200. Została ona wybrana ze względu na jej dostępność na uczelni. Dodatkowo, aby przygotować pliki pod druk, należało model przetworzyć programem Z-Suite. W programie większość ustawień pozostawiana była bez zmian, jedynie dwa istotne ustawienia zostały dostosowane do projektu. Zostało ustawione minimalne, niezerowe wypełnienie, około 10%. Warstwy wierzchnia i spodnia zostały ustawione na najgrubszą możliwą opcję. Ustawienia te znalezione zostały eksperymentalnie - wydają się najlepiej balansować między wytrzymałością a czasem druku i ilością zużytego materiału. Dodatkowo, przed ostatecznym wydrukiem, zwiększone zostały o około 0.3 – 0.4mm wszystkie otwory montażowe. Modele podczas druku "puchną" i w pierwszych wydrukach otwory te były znacznie mniejsze niż na tworzonych szkicach.

5. Implementacja

Implementacja oparta została o minikomputer Raspberry Pi 4B 2GB. Urządzenie to zostało wybrane losowo - była to platforma, która akurat była dostępna "pod ręką". Na minikomputerze zainstalowany został system operacyjny Linux Ubuntu, w wersji 22.04. System ten był wybrany, jako system wspierany zarówno przez RPi jak i przez środowisko ROS 2. Wersja 22.04 była natomiast najnowszą wersją w czasie rozpoczęcia tej części projektu. Bardziej popularny system operacyjny dla tej platformy, czyli Raspbian OS został odrzucony, ponieważ zainstalowanie na nim ROS-a wymaga użycia dockera. Na Ubuntu natomiast wystarczy wykonywać komendy podane wprost w dokumentacji środowiska ROS, co znacznie uprościło podstawową konfigurację środowiska pracy. Wybrana dystrybucja oprogramowania ROS to Humble Hawksbill (rys. 5.1). Podobnie jak w przypadku Ubuntu, wersja ta została wybrana ze względu na fakt, że była to w czasie rozpoczęcia prac nad projektem najnowsza. W celu zdalnego łączenia się z minikomputerem użyty został protokół SSH - po stronie RPi otwarty został serwer SSH, a klient (komputer PC) łączył się z nim przy pomocy aplikacji PuTTY. Dodatkowo do sterowania serwami zastosowany został sterownik Polulu Maestro. Urządzenie to zostało wykorzystane ze względu na dostępność i znajomość obsługi.



Rys. 5.1. Logo dystrybucji Humble oprogramowania ROS 2 [18]

5.1. Środowisko ROS [18]

Robot Operating System (ROS) to zestaw narzędzi programowych służący do tworzenia aplikacji z myślą o robotach. ROS wprowadza sieć niezależnych, działających równolegle, węzłów (ang. node). Węzły te komunikują się za pomocą tematów (ang. topic), które to stanowią niezmienny interfejs między węzłami.

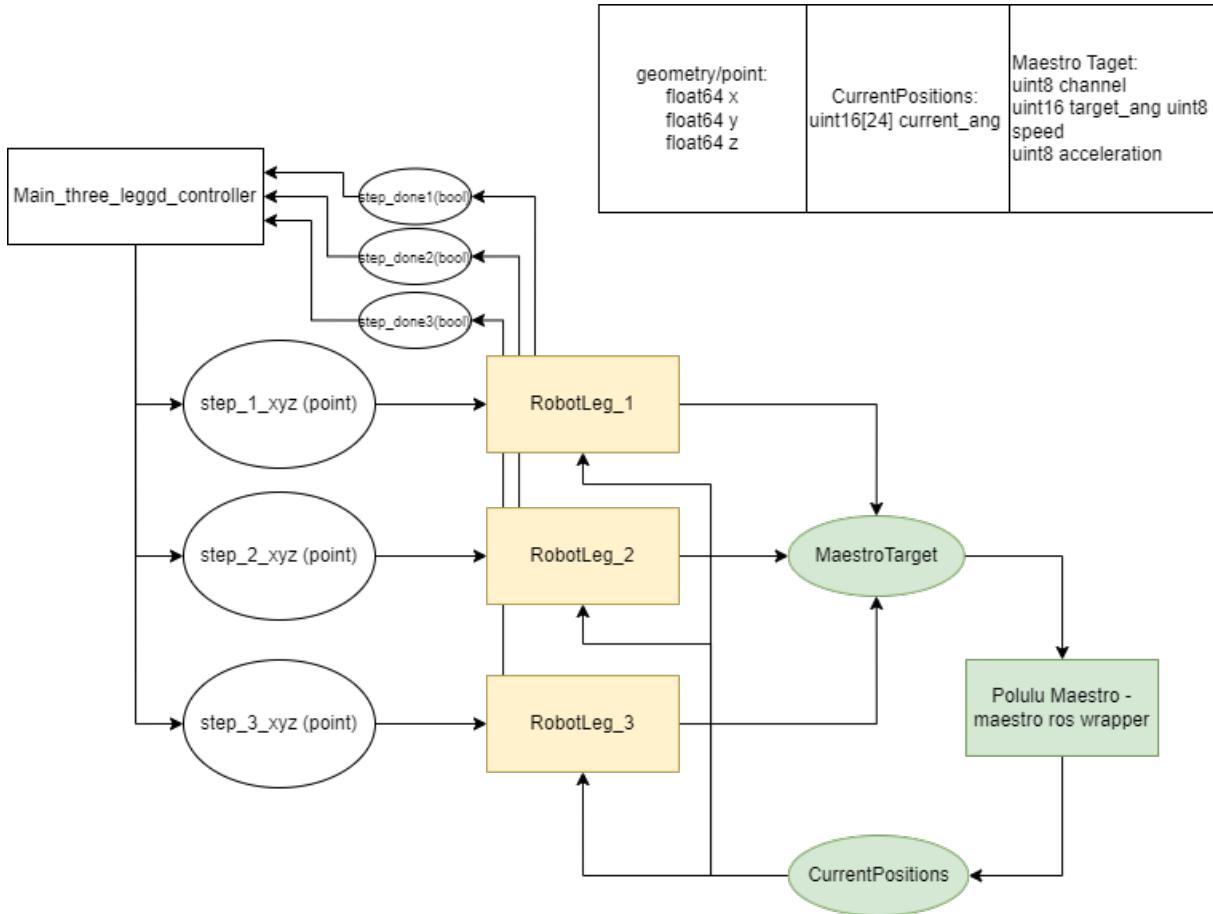
Węzły mogą na tematy dane publikować lub je odbierać a implementacja węzła znajdującego się po drugiej stronie nie ma absolutnie znaczenia. Co więcej, nawet nie ma znaczenia czy ten węzeł tam jest. Węzły mogą publikować dane na tematy, z których żaden inny węzeł tych danych nie odbiera. Sieć taka jest szczególnie przydatna w przypadku robotów o pewnym stopniu modularności, bądź w przypadku dowolnych modyfikacji. Zwykle jeden węzeł odpowiada jednemu fizycznemu elementowi robota. Zmiana tego elementu, nawet na element wymagający zupełnie innego oprogramowania, nie jest wtedy problemem. Wystarczy usunąć z sieci odpowiadający mu węzeł i na to miejsce wstawić inny. Jest to bardzo proste, dopóki interfejs (temat na jaki dany węzeł publikuje) pozostaje bez zmian. [18]

5.1.1. Schemat implementacji

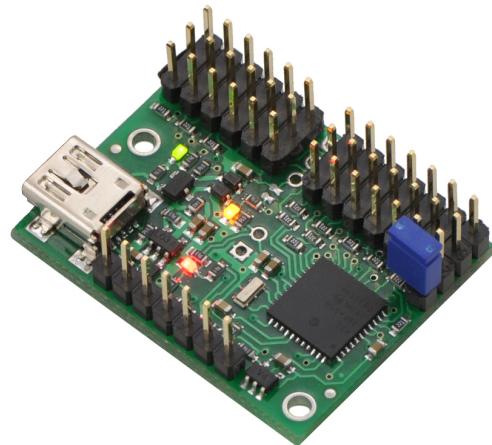
Rysunek 5.2 pokazuje schemat komunikacji między poszczególnymi węzłami. Na potrzeby aplikacji zostały stworzone trzy węzły. Jeden odpowiedzialny za obsługę wybranego sterownika do serw-Polulu Maestro, drugi odpowiedzialny za obsługę nogi. Trzeci zaś stanowi główny węzeł sterujący. Ma za zadanie generować algorytm chodu i wysyłać polecenia do nóg, aby układały się na docelowe pozycje.

5.2. Polulu Maestro

Do obsługi serwomechanizmów zastosowany został 24-kanałowy sterownik Polulu Maestro. Do uruchomienia tego robota wystarczyłby oczywiście Polulu Maestro 12 (rys 5.3), lecz wersja 24-kanałowa była zakupiona na potrzebny innego projektu i mogła być tutaj wykorzystana bez dodatkowych wydatków. Natomiast sterowniki do serw firmy Polulu są wymienne, można w każdej chwili przepiąć serwomechanizmy na sterownik o innej ilości kanałów i dokładnie ten sam program będzie w stanie go także obsłużyć. W przyszłości nie będzie problemu z podmianą tego sterownika na mniejszy.



Rys. 5.2. Schemat implementacji w środowisku ROS

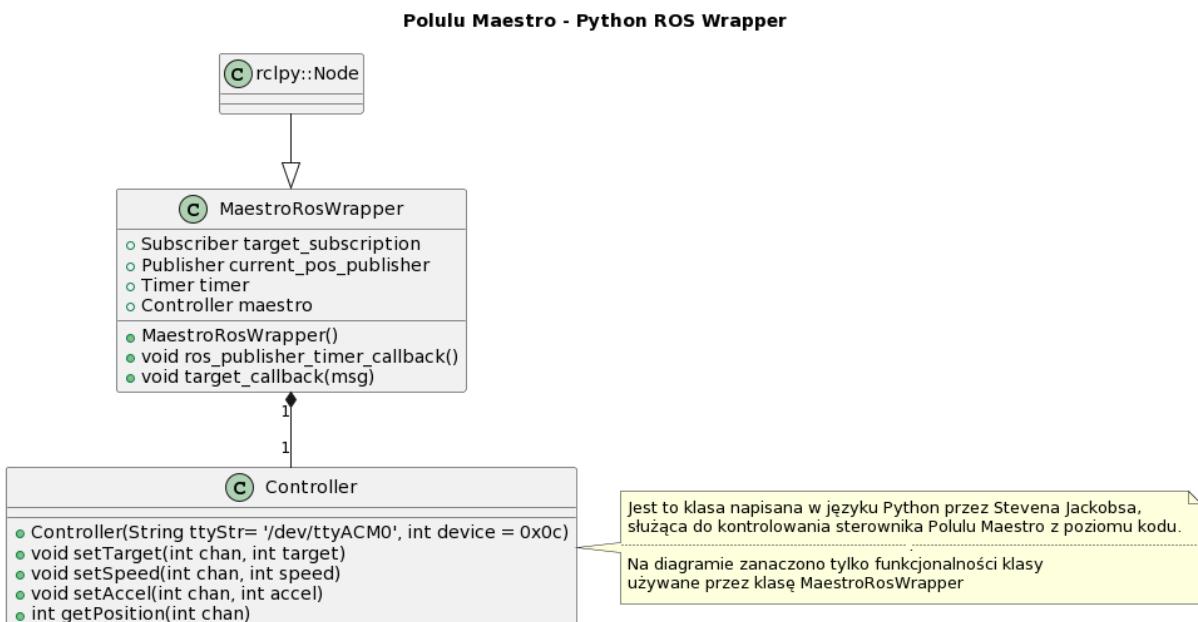


Rys. 5.3. Polulu Maestro 12 - kanałowe [19]

Komunikacja ze sterownikiem odbywa się za pomocą protokołu UART Serial. Nie było natomiast potrzeby aby tworzyć wiadomości wysypane tym protokołem od zera, ponieważ istnieje biblioteka do komunikacji z tym sterownikiem, napisana przez Stevena Jackobsa w języku Python [20]. Jest ona szeroko stosowana w projektach opartych na sterownikach firmy

Polulu. W tym projekcie należało jednak otoczyć tą bibliotekę pewnego rodzaju dekoratorem (ang. wrapper) aby połączyć ją z funkcjonalnościami ROSa. Dlatego napisana została klasa `MaestroRosWrapper` która przez kompozycję zawiera w sobie instancje klasy `Maestro` z wyżej wymienionej biblioteki `Maestro`. Stworzona klasa `MaestroRosWrapper` przede wszystkim implementuje:

1. Wysyłanie wiadomości z odczytem obecnej pozycji wszystkich serw
2. Subskrybowanie wiadomości maestro target, która zawiera informacje o:
 - kanale
 - pozycji docelowej
 - prędkości
 - przyspieszeniu



Rys. 5.4. diagram UML węzła Polulu Maestro Wrapper

Na potrzeby tej klasy stworzony został dodatkowy pakiet implementujący dwa niestandardowe (ang. custom) interfejsy:

- `MaestroTarget`
- `CurrentPositions`

Są to właśnie te dwa wyżej wspomniane interfejsy, za pomocą których węzeł ten komunikuje się ze światem zewnętrznym.

Całość kodu służącego do obsługi tego sterownika została napisana w języku Python 3. Język ten został w tym przypadku wybrany, ponieważ Steven Jacobs zaimplementował swoją bibliotekę w tym właśnie języku. Oczywiście przepisanie jej do innego języka (np. C++) nie stanowiłoby dużego problemu, jednakże nie jest to częścią tej pracy. Może to być jednak ciekawe ulepszenie tego projektu, jeżeli węzeł ROS-owy napisany w Pythonie okaże się zbyt pamięciożerny i wolny. Dokładny interfejs stworzonego węzła został przedstawiony na diagramie 5.4.

5.3. Noga Robotyczna

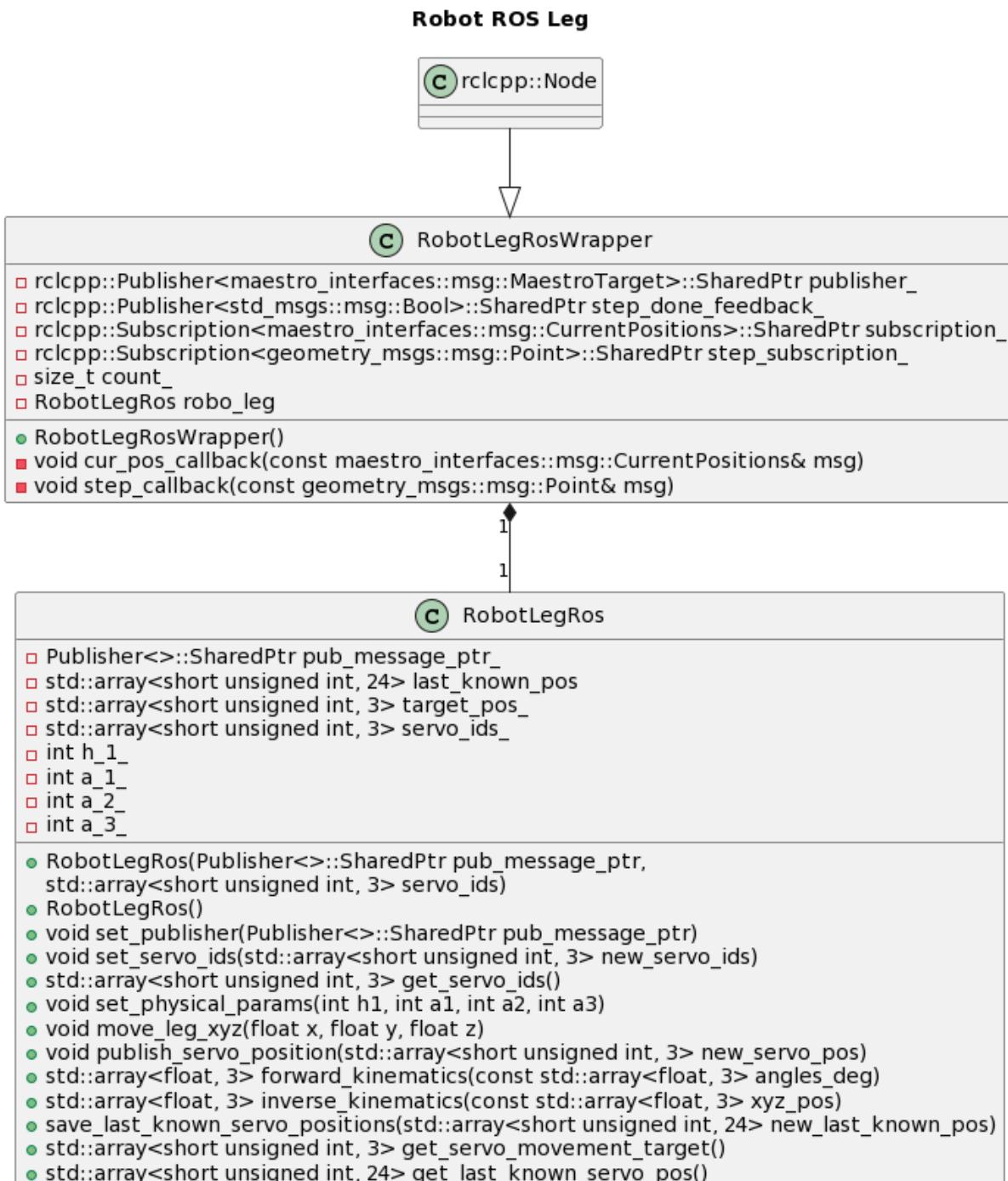
Poziom wyżej - nad sterownikiem do serw - znajduje się pojedyncza nogą robota. Cały program ją obsługujący był pisany na potrzeby tego projektu zupełnie od zera, co dało zupełną dowolność języka i ogólnej struktury implementacji.

Jako język został wybrany C++, z powodu ogólnej preferencji autora programu i aby lepiej zoptymalizować całość robota. Pisanie w języku Python jest znacznie prostsze, ale uruchomienie zbyt wielu węzłów napisanych w tym języku może powodować znaczne problemy z wydajnością.

Struktura natomiast, była częściowo wzorowana na poprzednim węźle - sterowniku do serw. Także przyjęto zasadę bardziej "generycznej" klasy wewnętrznej i stricte ROS-owego wrappera. W tym przypadku utworzono klasę RobotLeg, przede wszystkim odpowiedzialną za trzymanie informacji o fizycznych parametrach nogi i na ich podstawie przeliczania kinematyk prostej i odwrotnej. Natomiast klasa RobotLegWrapper jest odpowiedzialna za komunikację ze "światem zewnętrznym", czyli innymi węzłami ROS-owymi. Podział ten przedstawiony jest na rys 5.5 - diagramie UML tych klas.

Najważniejszymi dwoma interfejsami realizowanymi przez ten węzeł jest przyjmowanie nowej pozycji końcówki robota we współrzędnych kartezjańskich. Jak tylko takowa się pojawi, przeliczana jest kinematyka odwrotna zgodnie ze wzorem 2.15 i publikowana jest pozycja w kątach. Dodatkowo, węzeł ten oczekuje sprzężenia zwrotnego od węzła podległego - operującego sterownikiem. Sprzężenie to jest realizowane przez temat zawierający pozycje wszystkich serw.

Węzeł ten powinien także zapewnić sprzężenie zwrotne dla węzła nadziednego - tego który publikuje nowe pozycje końcówki nogi. Nie jest to jednak pełne sprzężenie zwrotne, takowe nie byłoby możliwe ze względu na fakt, że kinematyka odwrotna nie jest idealna - zawiera



Rys. 5.5. diagram UML węzła Robot Leg Ros

pewien błąd (co zostało dokładnie opisane w rozdziale Model Matematyczny/Noga Robotyczna), natomiast kinematyka prosta nie jest tym błędem obciążona. Prawdziwe sprzeżenie zwrotne spowodowałyby, że technicznie rzecz ujmując, nogą nigdy by nie osiągnęła punktu docelowego. Dlatego zostało ono uproszczone do wymaganego minimum - jak tylko pojawi się informacja o obecnych pozycjach serw (informacja zwrotna od sterownika) i będą one zgodne

z pozycjami docelowymi policzonymi za pomocą kinematyki odwrotnej, to węzeł zmienia wartość w temacie step done typu bool na prawdę. (W czasie wykonywania kroku publikowana jest cyklicznie cały czas wartość fałsz.) Taka informacja zwrotna dla węzła nadzorowanego jest jak najbardziej wystarczająca dla poprawnego ruchu robota.

5.3.1. Poprawki w interfejsach nogi robotycznej

Aby uczynić węzeł ten bardziej uniwersalnym i ułatwić jego zastosowanie w innych projektach, można by dodać dwa dodatkowe tematy, na które publikuje ten węzeł - "oszuksaną" kinematykę prostą, taką która uwzględnia błąd kinematyki odwrotnej. Dałoby to możliwość zrobienia prawdziwego sprzężenia zwrotnego i liczenia czy krok się faktycznie zakończył wewnątrz węzła nadzorowanego. Byłoby to rozwiązanie bliższe poprawnej "sztuki" implementowania układów sterowania. Dodatkowo warto by było także dodać publikację prawdziwego sprzężenia zwrotnego. Może ono być bardzo przydatne w wielu sytuacjach gdzie potrzebna jest znajomość realnej pozycji końcówki nogi.

Innym problemem z interfejsami który wymaga poprawek aby węzeł stał się bardziej uniwersalny są tematy za pomocą których noga komunikuje się ze sterownikiem do serw. Tematy te przenoszą wartości w ćwierć-mikrosekundach, które są jednostką stosowaną przez linię sterowników Polulu Maestro. Powoduje to że konwersja radiany na ćwierć-mikrosekundy jest realizowana już na poziomie nogi robotycznej, a potencjalna wymiana na sterowniki innych producentów może nie być możliwa bez modyfikowania samego kodu nogi. Jest to sprzeczne z ideą ROSa, gdzie wymiana sterownika powinna wiązać się jedynie z wymianą węzła obsługującego ten sterownik. Zamiana tych interfejsów na kąty w stopniach lub radianach i dodanie przeliczania po stronie węzła sterownika poprawiłoby znacznie uniwersalność tej implementacji.

Drobnego poprawek wymaga także podział funkcjonalności pomiędzy właściwą klasę RobotLeg a dekorator RobotLegROSWrapper. Jak już wspomniano wcześniej, wrapper ma być odpowiedzialny za komunikację a klasa wewnętrzna za obliczenia. Jednakże na wcześniejszych etapach implementacji podział ten nie był jeszcze tak jasny i ze względów historycznych, publikacja na temat Maestro Targets jest realizowana przez klasę wewnętrzną. Jest to o tyle problematyczne, że wrapper musi przekazać do klasy wewnętrznej wspólny wskaźnik (Shared Pointer), który wskazuje na publishera, na którego temat jest publikowany. Właśnie z tego powodu, i z powodu braku spójności obecnej implementacji, funkcjonalność ta powinna być

realizowana przez wrapper nie przez klasę wewnętrzną.

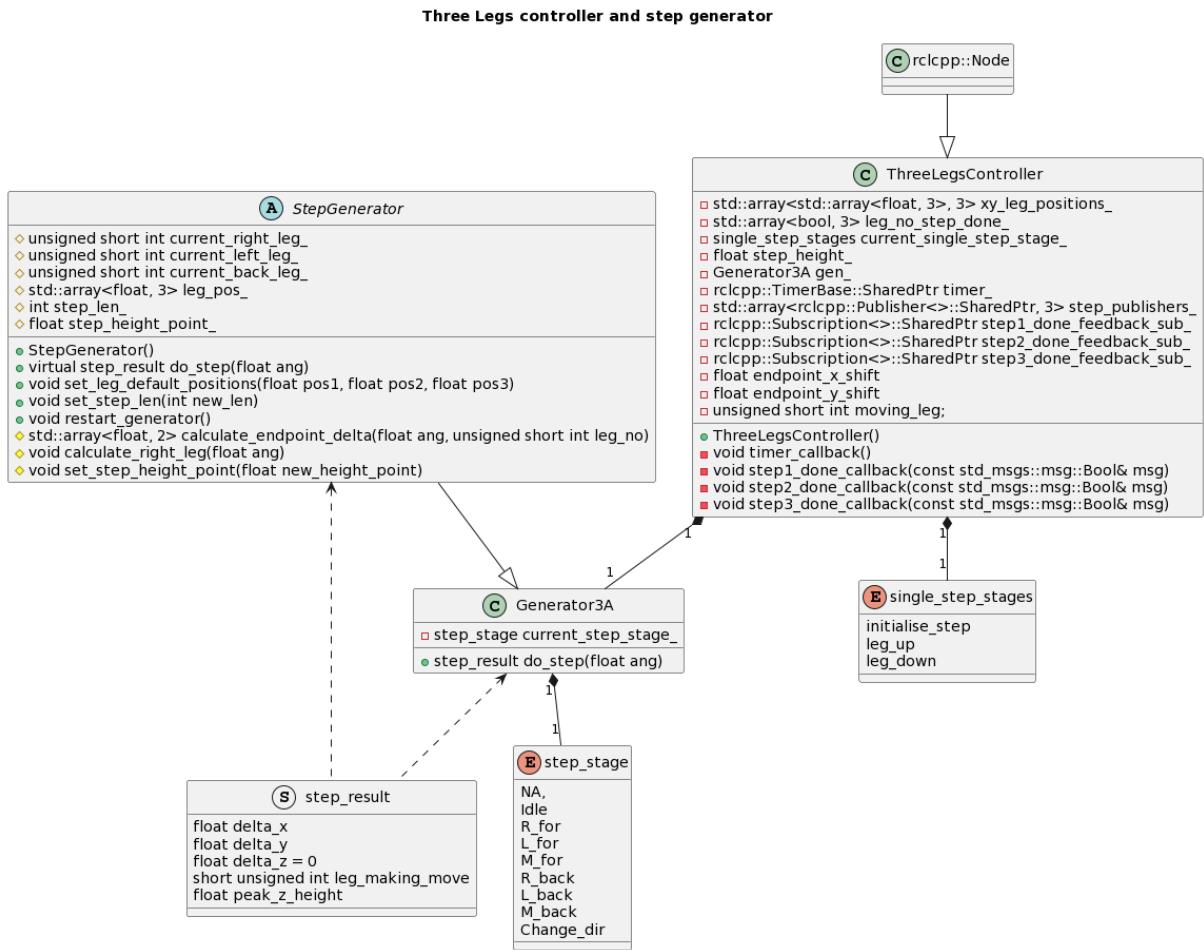
5.4. Klasa kontrolująca trzy nogi i generator trajektorii

Podobnie jak w przypadku dwóch poprzednich węzłów zastosowano tutaj podział na ROSowego wrappera i klasę implementującą właściwą logikę. Różnicą jest jednak że w przypadku tego węzła klasą "podzieloną" będzie generator trajektorii, który ma za zadanie jedynie generować nowe punkty na które nogi ma zostać przedstawiona. Jako że istnieje wiele możliwych algorytmów chodu za pomocą których robot ten może się przemieszczać i jednym z celów tego projektu jest stworzenie platformy do eksperymentów z właśnie tymi algorytmami, to należy odpowiednio modularnie zaprojektować interfejs między klasą publikującą a klasą generatora. Modularność taką można bardzo prosto zrealizować korzystając z mechanizmu dziedziczenia. Stworzona została abstrakcyjna klasa Generator3Legs, która implementuje wszystkie podstawowe funkcjonalności generatora jak konstruktor, ustawianie pozycji początkowych czy obliczanie ruchu konkretnej nogi. Jedna funkcja jest natomiast wirtualna - ta odpowiadająca za faktyczne generowanie kolejnych pozycji kolejnych nóg i zwracanie ich do klasy nadzorowanej. (Tutaj funkcja ta jest nazwana `do_step`.) Później napisana została klasa Generator3A która nadpisuje tylko funkcję `do_step`. Zależności te zostały przedstawione na diagramie UML na rysunku 5.6

Dodatkowo implementacja taka umożliwia bardzo ciekawy rozwój projektu. Idąc o krok dalej niż dziedziczenie i stosując mechanizm polimorfizmu, można w czasie rzeczywistym przełączać się między algorytmami. Dodając sterowanie np. za pomocą gamepada, można przyporządkować do kolejnych przycisków zmianę klasy potomnej, która odpowiada za generowanie kroków i zrealizować zmianę algorytmów w czasie pracy robota.

Klasa "wrappera"(ThreeLegsController) w przypadku tego węzła ma obecnie funkcjonalność ograniczoną do minimum. Po pierwsze, wywołuje wewnątrz odwołania (ang. callback) od zegara (ang. timer) generowanie kolejnych kroków. Równolegle też zbiera ze sprzężeń zwrotnych od kolejnych nóg informację czy wykonują one krok i pozwala na faktyczne przedstawienie nogi, tylko jeżeli dana nogi już się nie porusza. Oczywiście aby faktycznie nogi się przestawiły, węzeł ten musi publikować nowe pozycje na trzy tematy, po jednym dla każdej nogi.

W ramach rozwoju projektu klasa ta powinna przede wszystkim zostać rozbudowana o subskrybcję tematu, który zawiera informację o kierunku w jakim robot ma się poruszać i procentowej wartości prędkości tego poruszania. Takie informacje mogą już być publikowane przez



Rys. 5.6. diagram UML węzła kontrolera trzech nóg

przeróżne węzły, najbardziej oczywistym wydaje się jednak węzeł odpowiedzialny za obsługę gamepada. Jest to najbardziej podstawowy sposób sterowania takimi robotami. Oczywiście później można także implementować wszelakie systemy wizualne, autonomizujące tą platformę. Są to jednak bardzo odległe plany, wymagające wiele pracy.

5.5. Plik uruchomieniowy

Aby ułatwić uruchamianie całej konstrukcji stworzony został plik uruchomieniowy (ang. launch file). Plik ten to po prostu plik .py gdzie wewnątrz odpowiedniej funkcji tworzy się listę instancji wcześniej napisanych węzłów. Każdemu węzłowi można nadać nową nazwę i jest to w szczególności przydatne w przypadku węzła Leg. Jako że potrzebne są trzy identyczne węzły, należy zmienić ich nazwy tak aby nie pojawiały się konflikty przy ich uruchamianiu. Dodatkowo można zmienić wartości parametrów. Jest to także najbardziej użyteczne w przypadku nogi, ponieważ do poprawnej komunikacji każda nogą musi mieć ustawiony swój unikatowy

numer aby wiedzieć, które wiadomości od innych węzłów są przeznaczone dla niej. Ustawia się także numery kanałów sterownika Polulu Maestro, na które należy przeliczone kąty publikować.

6. Algorytmy chodu

Główym celem stworzonej platformy są eksperymenty z różnymi algorytmami chodu robotów trójnożnych. Umożliwiają to elementy przede wszystkim takie jak:

1. drukowana, łatwa w modyfikacjach konstrukcja,
2. przygotowana klasa abstrakcyjna StepGenerator

6.1. Modularna konstrukcja

Wpływ zmian konstrukcyjnych na algorytmy chodu wydaje się być raczej oczywisty - zmieniając długości poszczególnych członów nogi lub średnicę talerza centralnego dajemy robotowi więcej lub mniej czasu na odłożenie nogi, która jest w powietrzu zanim się wywróci. Ważniejszy jednak z punktu widzenia tego rozdziału jest wpływ dobrze przygotowanej klasy stanowiącej interfejs dla tworzenia algorytmów. Klasa ta przede wszystkim implementuje długość kroku i jego wysokość, ale także funkcję która zwraca informacje o kolejnych wykonywanych krokach. Dzięki temu można tworzyć nowe algorytmy dodając tylko klasę pochodną z jedną funkcją.

6.2. Właściwe algorytmy chodu [21]

Nauce znane są dwa typy algorytmów, jakie można zaimplementować na robotach trójnożnych. Oczywiście żaden z tych dwóch nie jest przeniesiem jeden do jeden algorytmów znanych naturze, są to raczej modyfikacje i reinterpretacje tego co można zaobserwować wśród zwierząt.

6.2.1. Algorytm króliczych skoków

Pewnym przykładem naturalnego przemieszczania się na trzech nogach jest sposób skakania królików czy kangurów. Te pierwsze podpierają się na przednich łapkach i przeskakują tylnymi do przodu, a kangury natomiast podpierają się na ogonie i przeskakują nogami wprzód. Pewną modyfikacją tego algorytmu jest sposób poruszania się ludzi o lasce. Algorytm "człowieka o lasce" polega na wysunięciu środkowej nogi do przodu, odbiciu się od dwóch

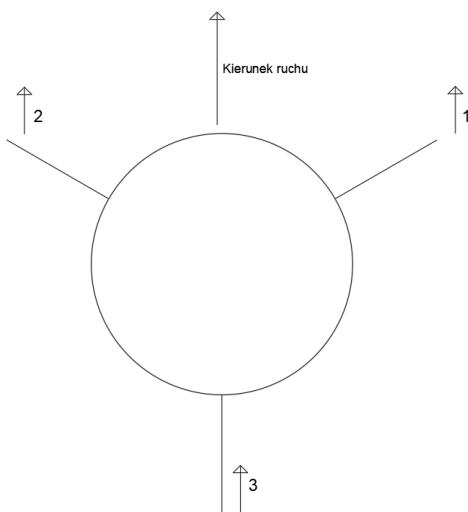
tylnych nóg i przestawieniu ich do przodu.

Pewną modyfikacją tego algorytmu jest sposób przemieszczania się zaimplementowany przez robota Strider (dokładny opis w podrozdziale 1.2.1).

6.3. Algorytm pajęczy

W świecie przyrody nie istnieją pająki o trzech nogach, pajęczaki zawsze mają osiem nóg. Dlatego możemy mówić tylko o pewnej reinterpretacji algorytmów ośmionożnych na trzy nogi. W praktyce algorytm ten polega na przestawianiu po jednej nodze do przodu.

Jako że jest to możliwie najprostszy algorytm, to taki właśnie został wstępnie zaimplementowany. Polega on na tym, że wpierw obliczane jest, która noga względem kierunku ruchu jest prawa, która lewa a która tylna. Następnie do przodu przestawiane są po kolejno nogi prawa, lewa, tylna (koleność przedstawiona na rys. 6.1). Później w tej samej kolejności nogi odsuwają się do tyłu (bez podnoszenia ponad ziemię). Całą sztuką tego algorytmu jest przestawienie nogi na tyle szybko, aby robot się nie zdążył w tym czasie wywrócić.



Rys. 6.1. Kolejność przestawiania nóg podczas chodu

Ta konkretna kolejność przestawiania nóg została wymyślona raczej przypadkowo. Nie wydaje się żeby kolejność w jakiej te nogi są przestawiane miała znaczenie, ważniejsze jest aby każda noga osobno przemieściła się w konkretnym kierunku.

7. Podsumowanie

7.1. Problemy konstrukcyjne i implementacyjne

Największe dwa problemy z powstałą platformą to:

1. Drukowana obudowa. - Konstrukcja taka pozostawia wiele do życzenia. Olbrzymie luzy na orczykach i ogólna giętkość wydrukowanych elementów powoduje, że nie da się jednoznacznie ustalić parametrów kroku wystarczających do "zwyczajnego" przedstawienia nogi. Trzeba brać znaczną poprawkę na giętkość konstrukcji, np. poprzez znacznie wyższy krok niż byłoby to faktycznie potrzebne.
2. Platforma o małej mocy obliczeniowej i węzeł napisany w Pythonie. - Niestety okazuje się że węzeł napisany w Pythonie nie jest w stanie nadążyć za węzłami napisanymi w C++ i nie jest w stanie przetworzyć ramek tak szybko jak pozostałe węzły je wysyłają.

W tym miejscu warto także wspomnieć że przybliżona kinematyka odwrotna okazała się jak najbardziej wystarczająca, przy tak drobnych krokach nie wprowadziła istotnego błędu w ruchy robota.

7.2. Przeprowadzone eksperymenty

Platforma została stworzona w taki sposób, aby dało się przeprowadzić każdy możliwy eksperyment. Jednakże w ramach tej pracy skupiono się tylko na podstawowych testach.

7.2.1. Zmiany czasówek

Kod programu zawiera trzy czasówki które można modyfikować i w ramach eksperymentów podjęto próbę jak największego zmniejszenia tych wartości.

Pierwsza wartość to okres odpytywania sterownika Polulu Maestro o obecne pozycje serw i wysyłania sprzężenia zwronego z tymi informacjami. Ostatecznie udało się zejść do wartości $XXms$. Inaczej mówiąc, jest to czas co jaki wywoływana jest funkcja `ros_publisher_timer_callback` z klasy `MaestroRosWrapper` (rys. 5.4).

Druga wartość to czas, co jaki jest wysyłana jest ramka z kolejnymi ruchami serwa pojedynczej nogi. Czas ten jest bezpośrednio związany z ilością ramek jakie węzeł Pythonowy jest w stanie przetworzyć w jdnostce czasu. Ostatecznie udało się zejść do wartości $XXms$. Wspomniane wcześniej ramki są wysyłane z poziomu funkcji `RobotLegRos::publish_servo_position` (rys. 5.5).

Tercia wartość to czas, co ile algorytm próbuje wykonać kolejne kroki. Wartość ta jest bezpośrednio związana z dwoma poprzednimi czasówkami. Im bardziej zmniejszymy powyższe okresy, tym bardziej możemy przyspieszyć sam algorytm chodu. Jeżeli natomiast kolejne kroki algorytmu zostaną zbyt przyspieszone bez przyspieszania wartości leżących "pod spodem" algorytmu to robot będzie wykonywał bliżej nieokreślone ruchy, bez podnoszenia nogi nad ziemię. Ruch ten wygląda jakby u źródła problemu leżało zatrzymywanie się kroku i przechodzenie do kolejnego etapu algorytmu zanim krok zostanie w pełni wykonany. Może to być wywołane wieloma różnymi elementami i niestety przyczyna nie została dokładnie zweryfikowana. Jednakże jako że zmniejszanie dwóch powyższych czasówek powoduje że problem pojawia się przy coraz to szybszym algorytmie, to najprawdopodobniej winowającą jest brak aktualnego sprzężenia zwrotnego lub nieopublikowane wartości skolejkowane gdzieś w środku algorytmu. Ostatecznie udało się zminimalizować ten czas do XX. Inaczej mówiąc, jest to okres wywołania funkcji `ThreeLegsController::timer_callback` (rys. 5.6).

7.2.2. Długość i wysokość kroku

Po ustaleniu najszybszych możliwych ruchów dla tej platformy przeprowadzono eksperymenty z długością i wysokością kroku.

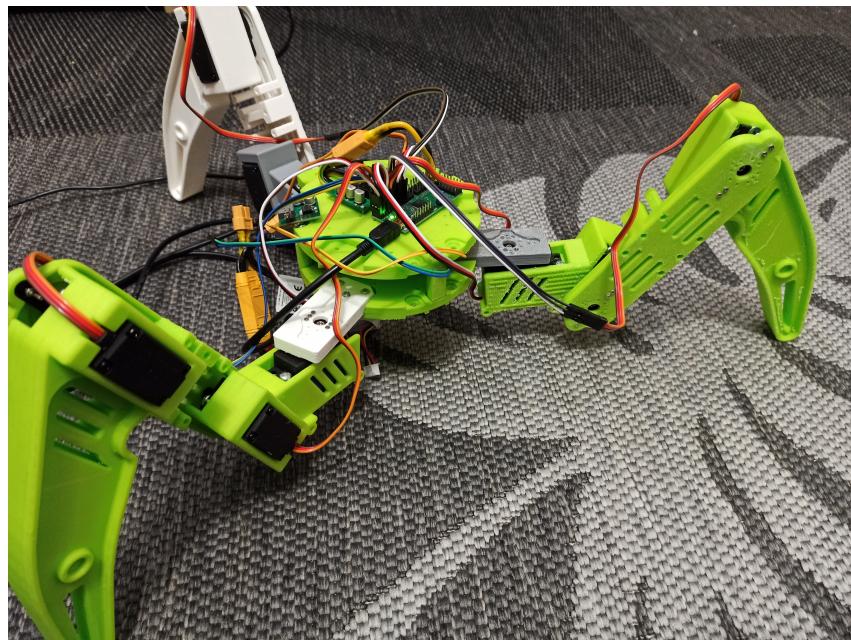
Początkowe wartości zostały przyjęte intuicyjnie. Długość wynosiła $100mm$ a wysokość $50mm$. Jednakże tylna nogą nie była w stanie osiągnąć takiej pozycji (końcówka nogi musiałaby znajdować się zbyt blisko środkowego serwa), dlatego długość ta została zmniejszona do $70mm$. Z wysokością także był problem, ponieważ z powodu gięcia się całej konstrukcji, trzeba było wysokość podnieść do $75mm$. Przy wartościach tych robot był w stanie wykonać krok. Co prawda każdorazowo uderzał podwozem o ziemię, ale przemieszczał się do przodu.

Z powodu gnącej się konstrukcji także wszelkie próby matematycznego obliczania takich parametrów kroku, aby robot wyrobił się z odstawieniem nogi zanim podwozie uderzy o ziemię zakończyły się niepowodzeniem. Została tylko opcja empirycznego dochodzenia do najlepszych wartości.

Pomimo licznych prób jednakże nie udało się znaleźć takich parametrów kroku, aby robot dał radę zrobić "czysty" krok, bez zaczepiania o ziemię. Ostatecznie ustalono wartości na $step_l = XXmm$ i $step_h = XXmm$, ponieważ przy takich wartościach krok wydawał się wyglądać najlepiej - czas styku nogi z podłożem był minimalny.

7.3. Wnioski

W ramach pracy udało się zrealizować z mniejszym lub większym sukcesem w zasadzie wszystkie założone punkty. Jako że robot faktycznie przemieszcza się do przodu, można by powiedzieć że projekt zakończył się sukcesem, jednak ostatecznej konstrukcji (przedstawionej na zdj. 7.1) która w ramach tej pracy powstała nie można uznać za udany produkt. Jest to raczej pierwszy prototyp na długiej drodze rozwoju tego projektu. Jest to jednak pierwszy prototyp który jak na pierwsze prototypy działa bardzo dobrze i spełnia swoją rolę idealnie - dokładnie pokazuje występujące problemy, możliwe sposoby ich naprawienia i potencjalne ścieżki rozwoju projektu.



Rys. 7.1. Zdjęcie konstrukcji powstałej w ramach pracy.

7.4. Przyszły rozwój projektu

Pierwszym krokiem rozwojowym powinno być naprawienie problemów wymienionych w podrozdziale 7.1. W pierwszej kolejności należałoby poprawić konstrukcję robota, dodając po przeciwniej stronie od każdego serwa na jego osi obrotu dodatkowy element usztywniający. Zapobiegłoby to gnięciu się elementów i luzom na serwach. Kolejną poprawką oczywiście byłoby przepisanie sterownika do serw do języka C++. Dałoby to drugą wersję prototypu, która byłaby już w pełni przygotowana do właściwego rozwoju.

W ramach trzeciej wersji można by już postarać się przede wszystkim o dodanie węzła sterującego całą konstrukcją z poziomu gamepada i stworzenie dedykowanej płytki PCB aby dało się wygodnie powpinać w nią wszelakie komponenty. Dalej możliwości są nieskończone - można zrobić wszystko od eksperymentów z nogami o innej długości aż po montowanie lidarów i wszelakich innych systemów wizyjnych, sterujących platformą.

Bibliografia

- [1] J. A. Tenreiro Machado Manuel F. Silva. „*A Historical Perspective of Legged Robots*”. 2006.
- [2] Dennis Hong Ping Ren Ivette Morazzani. „*Forward and Inverse Displacement Analysis of a Novel Three Legged Mobile Robot Based on the Kinematics of In Parallel Manipulators*”. 2007.
- [3] Masato Ishikawa Yoichi Masuda. „*Simplified Triped Robot for Analysis of Three-Dimensional Gait Generation*”. 2017.
- [4] „*Maker Faire 3-legged walking robot*”. 2020.
- [5] „*Missel tripod robot*”. 2008.
- [6] Julia Szymura Paweł Bańska Jacek Chmiel. „*Robot kroczący Hexapod Zebulon*”. 2011.
- [7] Richard P. Paul. „*Robot Manipulators: Mathematics, Programming, and Control : the Computer Control of Robot Manipulators*”. 1981.
- [8] automaticaddison. „*How to Find Denavit-Hartenberg Parameter Tables, blogpost by Automatic Addison*”. 2020.
- [9] Wojciech Paszke. „*Kinematyka prosta: reprezentacja Denavita-Hartenberga*”.
- [10] automaticaddison. „*Homogeneous Transformation Matrices Using Denavit-Hartenberg, blogpost by Automatic Addison*”. 2020.
- [11] Andrzej Rak Adam Labuda Janusz Pomirski. „*Model manipulatora o dwóch stopniach swobody*”. 2009.
- [12] Peter Corke. „*Inverse Kinematics for a 2-Joint Robot Arm Using Geometry*”.
- [13] Vishnu Mohan. „*Raspberry Pi 4 Power Requirements: Everything You Need to Know*”. 2022.
- [14] „*Polulu Step-Down Voltage Regulators*”.
- [15] „*Dokumentacja Feetech FT5715M*”.
- [16] „*Dokumentacja PowerHD LF-20MG*”.

- [17] „*Electrical characteristics of servos and introduction to the servo control interface*”. 2011.
- [18] „*Dokumentacja środowiska ROS2 Humble*”. 2022.
- [19] „*Dokumentacja Polulu Maestro 12*”.
- [20] „*Biblioteka Python Maestro*”.
- [21] TriPed Project Team. „*TriPed Project - Walking on three legs*”.