Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Sterowania i Elektroniki Przemysłowej

Praca dyplomowa magisterska

na kierunku Elektrotechnika

w specjalności Automatyka i Inżynieria Komputerowa

Sterowanie autonomicznym robotem kroczącym z wykorzystaniem jednopłytkowego komputera Raspberry PI2

Martyna Romanowska

Numer albumu 238769

promotor

dr inż. Witold Czajewski

Warszawa 2018

STEROWANIE AUTONOMICZNYM ROBOTEM KROCZĄCYM Z WYKORZYSTANIEM JEDNOPŁYTKOWEGO KOMPUTERA RASPBERRY PI2

Streszczenie

Zasadniczym celem niniejszej pracy magisterskiej było stworzenie oprogramowania w języku C++ dla maszyny kroczącej wyposażonej w system wizyjny. Ma ona za zadanie zlokalizować piłkę, podejść do niej i ją kopnąć. Robot porusza się dwoma wybranymi rodzajami chodu: trójpodporowym oraz pełzającym, do których realizacji zaprojektowano różne trajektorie nóg. Do określenia położenia piłki, wykorzystano kamerę oraz bibliotekę OpenCV i zaimplementowano algorytm rozpoznawania okrągłego, kolorowego obiektu na podłożu.

Pierwszym etapem realizacji projektu było stworzenie algorytmu do poruszania robotem w wybranym kierunku. Aby to wykonać najpierw zaimplementowano w programie sterowanie pojedynczym serwem, następnie nogą składającą się z trzech serw, a na końcu wszystkimi sześcioma nogami robota. Zaprojektowano trajektorie, po których porusza się każda z nóg w zależności od rodzaju oraz kierunku ruchu. Robot może przemieszczać się do przodu i do tyłu, w lewo i w prawo oraz obracać się wokół własnej osi. W pracy zaimplementowano 2 różne chody maszyny, lecz w praktyce najczęściej wykorzystywano chód trójpodporowy, gdyż jest on najszybszy z możliwych chodów maszyn sześcionożnych. Wszystkie algorytmy odpowiedzialne za ruch robota opisano w rozdziale 6.

Kolejnym etapem pracy, była detekcja małego, okrągłego obiektu, czyli piłki. Dla uproszczenia założono, że rozpoznawany jest tylko obiekt koloru czerwonego. Algorytm, zaimplementowany z wykorzystaniem biblioteki OpenCV, przeprowadza operację znalezienia piłki na zdjęciach z kamery umieszczonej na robocie. Najpierw wyszukuje wszystkie czerwone, okrągłe obiekty i odrzuca pozostałe. Następnie, poprzez optymalizację wartości progów dla nasycenia i jasności pikseli, dokonuje precyzyjnej segmentacji piłki, co jest istotne dla prawidłowej estymacji jej położenia w przestrzeni. Cały algorytm wraz ze szczegółami oraz krótkim wstępem do przetwarzania obrazu opisano w rozdziale 5.

Ostatecznym zadaniem, weryfikującym poprawność działania zaproponowanych rozwiązań, było podejście do zlokalizowanej piłki i jej kopnięcie. Robot podchodzi do piłki w dwóch etapach. W pierwszym, określa położenie piłki z dużego dystansu i zbliża się do niej na pewną niewielką odległość, po czym dokonuje kolejnego, dokładniejszego pomiaru jej położenia i koryguje swoją pozycję. Dzięki temu, pozycja piłki jest dokładnie określona, a robot może, w drugim etapie, precyzyjnie podejść do niej i ją kopnąć. Cały powyższy algorytm wraz ze szczegółami opisano w rozdziale 6.

W pierwszych trzech rozdziałach przedstawiono teorię potrzebną do wykonania tego projektu. Opisano m.in. klasyfikację maszyn, zagadnienia związane z kinematyką odwrotną i prostą oraz konstrukcję wykorzystanej w pracy maszyny kroczącej. Dodatkowo, w rozdziale 4 pokazano proces instalacji i konfiguracji środowiska oraz narzędzi niezbędnych do realizacji tego projektu.

Control of autonomous walking machine using Raspberry PI 2 single-board computer

Abstract

The main goal of this master's thesis was to create software in the C++programming language for a walking machine equipped with a vision system. Its task is to locate the ball, approach it and kick it. The walking machine moves with two selected gaits: tripod gait and crawling gait. For each of them different legs trajectories were designed. In order to determine the position of the ball, a camera and the OpenCV library were used with an algorithm capable of recognizing a round, colored object on the ground.

At the first stage of the project an algorithm for moving the robot in the desired direction was created. To this end, a single servo control program was first implemented, followed by a three-servos leg control program. Finally all six robot's legs were controlled. Moreover, legs' trajectories were designed specifically for different types of gait and direction of movement. The robot can move forward and backward, to the left and to the right, and rotate around its own axis. In this master's thesis, two different gaits were implemented, but most of the time the tripod gait was used as it is the fastest possible gait for six-legged machines. All the algorithms responsible for robot movement are described in Chapter 6.

The next stage of the thesis was the detection of a small, round object, i.e. a ball. For simplicity, it was assumed that only red objects were to be recognized. The algorithm, implemented using the OpenCV library, performs the operation of finding the ball in the robot's front mounted camera image. First, it searches for all red, round objects and rejects the others. Then, by optimizing the thresholds for pixel saturation and brightness, it performs precise segmentation of the ball, which is important for the correct estimation of its position in space. The entire algorithm with details and a brief introduction to image processing is described in Chapter 5.

The final task, that would verify the correctness of the proposed solutions, was making the robot approach the localized ball and kick it. The robot moves toward the ball in two stages. In the first one, it estimates the position of the ball from a long range and approaches it stopping at a small distance. Then it makes another, more accurate measurement of the ball's coordinates and corrects its own position. Thanks to this, the position of the ball is accurately determined and in the second stage the robot can precisely approach it and kick it. The entire algorithm and its details are described in Chapter 6.

The first three chapters present the theory necessary to carry out this project, i.e. walking machine classification, inverse and forward kinematics and the structure of the walking machine used within the thesis. Additionally, in Chapter 4 the installation and configuration of the environment as well as the tools necessary to complete this project are described.

załącznik do zarządzenia nr . 2016 r.
Rektora PW

Politechnika Warszawska

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

Warszawa 17.06.2018
miejscowość i data

Martyna Romanowska
imię i nazwisko studenta
238769
numer albumu
Elektrotechnika
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

– niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, – niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony, – niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych, – wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami, – znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w systemie iSOD są identyczne.

Martyna Romanowska
czytelny podpis studenta

Spis treści

1. Wstęp

W dynamicznie rozwijającym się świecie jest coraz większe zapotrzebowanie na roboty, które będą mogły wykonywać zadania zagrażające zdrowiu, a nawet życiu człowieka. Słowo "robot" [47] zostało wymyślone przez czeskiego artystę Josefa Čapka. Pochodzi ono od czeskiego wyrazu „robota” oznaczającego ciężką pracę. Aktualnie roboty są stosowane w badaniach naukowych, edukacji, przemyśle, medycynie oraz w rozrywce. Jednocześnie z powierzaniem im coraz większej liczby odpowiedzialnych zadań, rośnie popyt na nie oraz na ich wielofunkcyjność.

Roboty można podzielić na: przemysłowe (manipulatory), mobilne, latające, pływające oraz kroczące, wśród których można wyróżnić specjalną grupę, czyli humanoidy. Najbardziej osobliwą i zadziwiającą z wyżej wymienionych grup jest grupa maszyn kroczących. Naśladują one sposób poruszania się zwierząt, dlatego wymagają od konstruktorów obszernej wiedzy z mechaniki, elektroniki i informatyki. Co więcej, ich wyjątkowość podkreśla fakt, że przyjmują one niemal dowolne kształty w zależności od wyobraźni ich autorów i rozwoju techniki. Mogą poruszać się po niemal dowolnej powierzchni w niemal dowolny, wymyślony sposób przez konstruktorów.

W niniejszej pracy wykorzystano maszynę sześcionożną, której konstrukcja wzorowana była na karaluchu. Został wybrany taki model robota, ponieważ ma on wystarczającą liczbę nóg do stabilnego poruszania się i utrzymania równowagi podczas postoju. Co więcej, można zaimplementować więcej niż jeden rodzaj chodu. Dzięki dość długiemu i szerokiemu korpusowi maszyny, można było na niej umieścić dodatkowe elementy takie jak: kamera, podkładki pod elektroniczne urządzenia oraz zasilanie typu powerbank, co umożliwiło jej większą swobodę ruchu w porównaniu do wersji z zasilaniem i sterowaniem przewodowym.

1.1. Cel i układ pracy

Głównym celem pracy było stworzenie oprogramowania w języku C++ dla maszyny kroczącej wyposażonej w system wizyjny. Ma ona za zadanie zlokalizować piłkę, podejść do niej i ją kopnąć. Robot porusza się dwoma wybranymi rodzajami chodu: trójpodporowym oraz pełzającym, do których realizacji zaprojektowano różne trajektorie nóg. Do określenia położenia piłki, w programie zaimplementowano algorytm rozpoznawania okrągłego, kolorowego obiektu na podłożu. Robotem, na którym była wykonywana praca magisterska, jest aluminiowa maszyna krocząca, mająca sześć odnóży, która swoim wyglądem przypomina karalucha. Siłę napędową konstrukcji stanowią serwomechanizmy analogowe, które są zasilane i kontrolowane przez sterownik serw. Zaprojektowane oprogramowanie zostało umieszczone w pamięci mikrokomputera, do którego podłączono kamerę oraz sterownik serw. Dzięki takiemu połączeniu, otrzymano system zdolny do szybkiej i niezawodnej kontroli działania maszyny oraz do akwizycji obrazów w wysokiej rozdzielczości w krótkim czasie.

W początkowej fazie wykonywania projektu, zaimplementowano trapezową trajektorię nogi z uwzględnieniem jej fizycznych ograniczeń. Na tej podstawie powstały różne trajektorie nóg, których użyto w celu wygenerowania chodu trójpodporowego oraz pełzającego. Dzięki temu, maszyna krocząca porusza się w wybranych kierunkach za pomocą jednego z dwóch wymienionych wcześniej rodzajów chodu. Następnie, zaimplementowano algorytm detekcji małego, okrągłego obiektu o wybranym kolorze na otrzymanym z kamery obrazie. Inne przedmioty nieposiadające, chociaż jednej z trzech podanych wyżej cech, zostają odrzucone przez filtr. Po prawidłowym zlokalizowaniu piłki, obliczana jest odległość pomiędzy nią, a robotem. W końcowej fazie projektu, zaimplementowano kod realizujący podejście robota do piłki i jej kopnięcie.

W rozdziale drugim przedstawiono historię najważniejszych wydarzeń w robotyce. Opisano także klasyfikację maszyn kroczących ze względu na liczbę nóg oraz stabilność ruchu. Dodatkowo, pokazano sposób poruszania się robotów sześcionożnych.

W rozdziale trzecim opisano układ współrzędnych nogi oraz obliczenia rozwiązań zadań kinematyki odwrotnej i prostej. Co więcej, przedstawiono konstrukcję robota oraz poszczególne elementy układu sterowania.

W rozdziale czwartym opisano proces instalacji i konfiguracji mikrokomputera Raspberry PI oraz poszczególnych elementów oprogramowania potrzebnych do wykonania projektu.

W rozdziale piątym zawarto opis algorytmu detekcji piłki, który składa się z filtracji obiektów, doboru optymalnych wartości, odcienia i nasycenia barwy czerwonej oraz wyliczenia odległości pomiędzy piłką, a robotem. Dodatkowo, opisano podstawowe pojęcia związane z biblioteką OpenCV i z przetwarzaniem obrazu w przestrzeni dwu i trójwymiarowej.

W rozdziale szóstym przedstawiono sposób poruszania się maszyny kroczącej, a w szczególności trajektorię trapezową i pozycję bazową nogi, ułożenie nogi do kopnięcia piłki, algorytm generacji chodu trójpodporowego oraz pełzającego. Ponadto, pokazano sposób zamiany odległości robot–piłka uzyskanej z systemu wizyjnego na liczbę iteracji algorytmu realizującego chód maszyny. Na końcu opisano proces podejścia robota do piłki oraz jej kopnięcia.

W rozdziale siódmym przedstawiono wnioski dotyczącej realizacji projektu oraz propozycje jego dalszej rozbudowy.

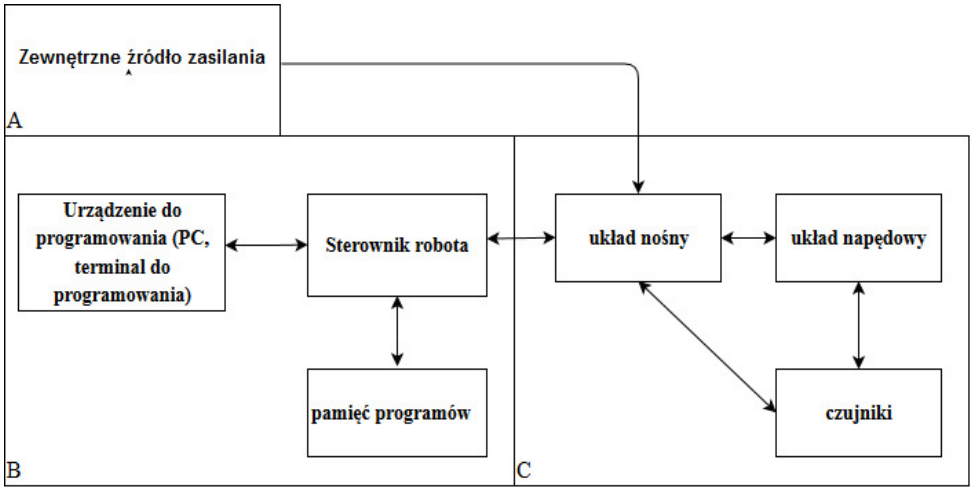
2. Wprowadzenie do maszyn kroczących

Maszyny kroczące to urządzenia, które poruszają się za pomocą kończyn naśladując ruch zwierząt. Dzięki temu, prowadzone są na nich badania, które mają na celu bliższe poznanie zasad poruszania się fauny. Dodatkowo, testowane są na nich także różne sztuczne sieci neuronowe. Badane są zdolność oraz szybkość takich sieci do podejmowania efektywnych decyzji np. dotyczących wykonania następnych kroków na nierównym podłożu. Maszyny kroczące są także stosowane w eksploracjach np. wulkanów [25], terenów podwodnych [14] oraz terenach objętych katastrofą [22]. W porównaniu do robotów mobilnych, mogą one szybko poruszać się po skomplikowanym terenie oraz wejść w miejsca niedostępne dla tych robotów.

Największą popularność maszyny kroczące zdobyły w przemyśle rozrywkowym i zabawkarskim. Zazwyczaj swoim wyglądem przypominają zwierzęta np. psy lub człowieka. Służą one głównie do zabawy i wykonują nieskomplikowane polecenia. Powstają także robotyczne zespoły muzyczne [40] oraz piłkarskie [36]. Maszyny grają na instrumentach proste melodie. W przypadku piłki nożnej, organizowane są co roku zawody, RoboCup, gdzie roboty tworzą drużynę i grają w piłkę.

Roboty kroczące są z wielką chęcią wybierane przez pasjonatów robotyki. Powstają modele o różnych wielkościach i kształtach. Wykonane są z dowolnego materiału, a dzięki szybkiemu postępowi w technice, urządzenia elektroniczne takie jak kamery, mikrokomputery są łatwo dostępne, niedrogie oraz nietrudne do zaprogramowania. Co więcej, konstrukcją maszyn kroczących zainteresowane jest wojsko. Powstają roboty mogące przenosić ciężkie rzeczy, prowadzić rekonesans terenów oraz rozładowywać miny lądowe.

Najczęściej roboty kroczące przypominają swoją budową owada. Składają się z: zewnętrznego źródła zasilania, układu sterującego oraz struktury konstrukcyjnej: układu napędowego, czujników oraz układu nośnego, co pokazano na poniższym rysunku 2.1.



Rys. 2.1 Schemat blokowy robota kroczącego
A – zewnętrzne źródło zasilania, B – układ sterujący, C – struktura konstrukcyjna. Źródło [2]

Ciało robota może być wykonane z aluminium, metalu lub plastiku, do którego przytwierdzone są odnóża z stopami umożliwiającymi ruch. W rozwiązaniach amatorskich występują dodatkowo materiały takie jak drewno, papier [57] lub styropian. Na korpusie i na odnóżach często umieszczane się czujniki pozwalające na interakcję robota z otoczeniem oraz określenie jego lokalizacji. Do powszechnie stosowanych czujników zaliczają się: akcelerometry, kamery, żyroskopy etc. W układzie napędowym wykorzystuje się silniki obrotowe prądu stałego DC wraz z enkoderami, małe silniki krokowe oraz serwonapędy.

2.1. Zarys historii maszyn kroczących

Już w czasach starożytnych idea sztucznej maszyny, która będzie bezwarunkowo wykonywać rozkazy, fascynowała ludzi. W dziele Homera „Iliadzie" pojawiła się koncepcja „złotych sług" stworzonych przez boga–kowala Hefajstosa. Grecki matematyk Archytes z Tarentu (ok. 420 r. p.n.e) stworzył drewniany model gołębia, który potrafił latać. Domniemanym źródłem jego napędu było powietrze. Bardzo dużego przełomu w robotyce dokonał arabski inżynier Ismail al–Jazari, który żył na dworze króla Anatolii (dzisiejsza Turcja) w średniowieczu. W swojej księdze pt. „Księga Wiedzy o Pomysłowych Urządzeniach Mechanicznych" opisał zasady działania oraz konstrukcji ponad 100 urządzeń mechanicznych. Przedstawił w niej m.in. pierwszą maszynę podobną do człowieka–humanoida, której zadaniem było podawanie wody. Także tak wielki geniusz jakim był Leonardo da Vinci interesował się robotami.

Stworzył on robota żołnierza na podstawie wyników badań ludzkiej anatomii. Do jego zadań należało siadanie, ruszanie rękoma i podnoszenie przyłbicy. Żołnierz składał się z rolek, sznurków oraz zbroi.

Przez długie lata renesansowi konstruktorzy czerpali inspiracje z pomysłu Leonarda. Co więcej, w tamtym czasie (około roku 1525) powstał, zachowany do dziś, najstarszy robot na świecie, który znajduje się aktualnie w muzeum w Wiedniu. Jest to Dama z Lutnią, która została skonstruowana przez Włocha Juanelo Turriano [10]. Niestety, obecnie maszyna nie jest już sprawna. Przez wiele stuleci roboty–zabawki zdominowały wyobraźnię konstruktorów, chociaż zdarzały się wyjątki, które wynikały z ówczesnych potrzeb. Skonstruowano bardziej praktyczne maszyny takie jak: automat do pisania tekstu (konstruktor Knauss [10]) oraz urządzenia przędzalnicze (konstruktor E.Cartwright [10]).

Wielkim przełomem, który miał wpływ nie tylko na robotykę, lecz także i na radiotechnikę, było stworzenie przez wybitnego naukowca Nikolę Teslę, zdalnie sterowanej, bezzałogowej, mini łodzi podwodnej „Teleautomaton". Sterowanie odbywało się za pomocą fal radiowych, a transmisja była zabezpieczona bramką logiczną, której unowocześniona i zminiaturyzowana wersja jest używana we współczesnych komputerach. Dzięki temu wynalazkowi rozpoczęła się era zdalnie sterowanych robotów. W 1921 roku czeski artysta Karel Čapek [47] w swojej sztuce R.U.R (Rossum's Universal Robots) nazwał maszyny stworzone przez człowieka, robotami. Wkrótce potem, w 1940 roku, amerykański pisarz i profesor biochemii Isaac Asimov [46] wymyślił cztery prawa robotyki o następującej treści:

„0. Robot nie może skrzywdzić ludzkości, lub poprzez zaniechanie działania doprowadzić do uszczerbku dla ludzkości."

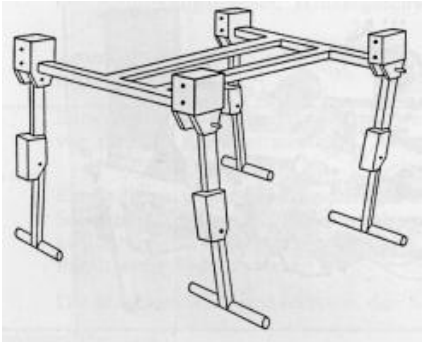
„1.Robot nie może skrzywdzić człowieka, ani przez zaniechanie działania dopuścić, aby człowiek doznał krzywdy."

„2.Robot musi być posłuszny rozkazom człowieka, chyba, że stoją one w sprzeczności z Pierwszym Prawem."

„3.Robot musi chronić samego siebie, o ile tylko nie stoi to w sprzeczności z Pierwszym lub Drugim Prawem."

Po zakończeniu drugiej wojny światowej nastąpił gwałtowny rozwój technologii. W szczególności, wynalezienie tranzystora i układu scalonego przyczyniło się do zmniejszenia rozmiarów ówczesnych komputerów oraz zwiększenia ich mocy obliczeniowej. Dzięki temu, w 1962 roku powstała pierwsza na świecie maszyna przemysłowa, której zadaniem było wykonywanie powtarzalnych i prostych czynności na linii produkcyjnej w firmie General Motors. Rok później, zostaje opracowany pierwszy na świecie system wizji, a cztery lata później w robocie mobilnym zostaje po raz pierwszy zastosowana sztuczna inteligencja. Maszyna była „świadoma" swoich ruchów oraz reagowała na nie.

W 1966 roku na uniwersytecie Południowej Karoliny została skonstruowana pierwsza na świecie maszyna krocząca, która była sterowana komputerowo (rys. 2.2).



Rys. 2.2 Pierwsza na świecie maszyna krocząca. Źródło [60]

W roku 1971 został zademonstrowany pierwszy na świecie mikroprocesor, który przyczynił się do kolejnego gwałtownego rozwoju techniki. Dzięki temu, powstały maszyny z systemami czujników, sondy kosmiczne: Voyager 1 i Voyager 2 etc. Roboty przemysłowe zaczęły zastępować ludzi przy ciężkich pracach przy liniach produkcyjnych w zakładach. Pionierem w robotyzacji produkcji była firma General Motors w Stanach Zjednoczonych. We wczesnych latach ’90, na uczelni MIT w Stanach Zjednoczonych powstało laboratorium zajmujące się badaniem ruchu maszyn kroczących. Pierwszym robotem skonstruowanym w nim był jednonożny robot Hopper, a potem powstały maszyny 2,3,4- nożne. Dość znaczący wkład w rozwój robotyki miał Instytut Robotyki na Uniwersytecie w Carnegie Mellon. Pracownicy instytutu, jako pierwsi opracowali dynamikę chodu pojedynczej nogi, która stała się podstawą dla ruchu innych robotów, w tym i dla Hoopera. Co więcej, instytut ten stał się wiodącym ośrodkiem w rozwoju robotyki, co potwierdzają fakty, że jako pierwsi zbudowali autonomiczny samochód w roku 1986 i w późniejszych latach robota Dante, który zszedł na dno wulkanu Mt. Erebus na Antarktydzie. Współcześnie jest to bardzo ważny ośrodek badawczy, który wykonuje przełomowe projekty takie jak np. robotyczny asystent dla niewidomych osób. W roku 1989 pracownik Laboratorium Sztucznej Inteligencji MIT napisał artykuł pt. „Fast, Cheap and Out of Control”, który zwrócił uwagę czołowych producentów robotów i naukowców na małe roboty mobilne. W tym czasie, powstał robot RoboTuna naśladujący ruchy ryby oraz zorganizowano po raz pierwszy walki robotów. W 1996 roku firma Honda przedstawiła pierwszego na świecie, niezależnego robota humanoidalnego – P3 (rys. 2.3). Głównym jego zadaniem było odzwierciedlenie chodu człowieka. Co więcej, potrafił on omijać przeszkody w postaci mebli, chodzić po schodach, pchać wózek, przenosić lekkie obiekty do 4 kg i przechodzić przez drzwi. 10 lat później kolejny model firmy Honda był dyrygentem, podczas występu orkiestry symfonicznej w Detroit.

Rys. 2.3 Humanoid P3 firmy Honda. Źródło [11]

Wraz z zmniejszającymi się kosztami produkcji poszczególnych elementów, budowanie robotów stało się tańsze, dlatego coraz częściej wykonują one za człowieka proste czynności np. autonomiczne odkurzanie oraz wymagające zadania np. lakierownie samochodów w fabryce.

Współcześnie robotyka to jedna z najbardziej dynamicznie rozwijających się branży na świecie. Wielkie firmy przemysłowe, wojsko, jak i dopiero co powstałe firmy specjalizujące się w robotyce, prześcigają się w dodawaniu i ulepszeniu funkcjonalności robotów. Co więcej, w roku 1997 odbył się pierwszy na świecie turniej robotów w piłce nożnej Robot Soccer World Cup w Japonii. Współcześnie niekwestionowanymi liderami w branży robotycznej, obok Instytutu Robotyki w Carnegie Mellon, są: niemiecka firma Festo, która specjalizuje się w automatyce przemysłowej oraz amerykańska firma Boston Dynamics, która zajmuje się tworzeniem robotów dla amerykańskiej armii. Aktualnie ich innowacyjnymi i ciekawymi projektami są: bioniczne mrówki (ang. bionicANTs) firmy Festo oraz humanoid Atlas firmy Boston Dynamics.

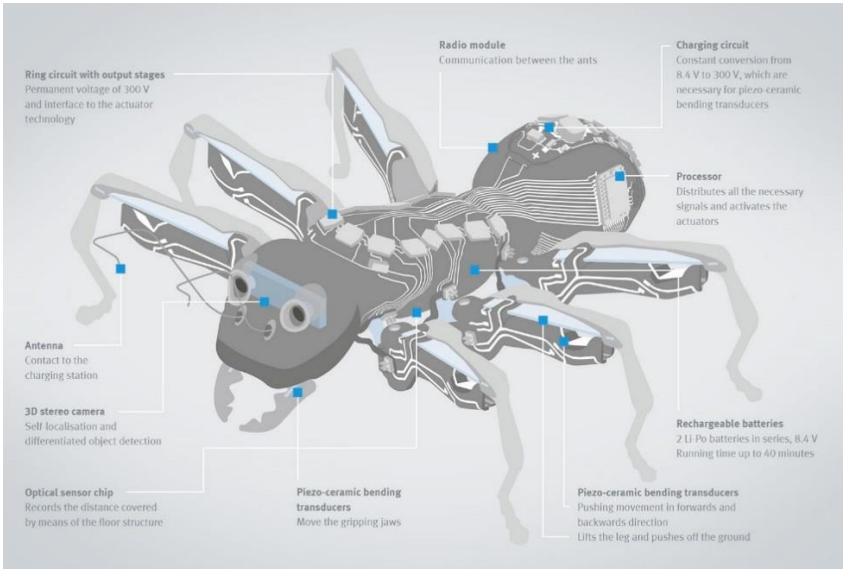
Bioniczne mrówki (rys. 2.4) to miniaturowe roboty o długości 135 mm, wysokości 43 mm i szerokości 150 mm. Wykonane są w rewolucyjnej technologii nadrukowywania elementów elektronicznych – 3D MID.

Rys. 2.4 Bioniczna mrówka firmy Festo. Źródło [17]

Mają wbudowane moduły radiowe wraz z antenami do komunikacji pomiędzy sobą i

kamery stereowizyjne w celu identyfikacji obiektów oraz określenia swoich położeń.

Zasilone są z dwóch małych, szeregowo połączonych baterii litowo–polimerowych o napięciu równym 8,4 V. Podnoszenie nóg i zginanie szczypiec odbywa się przy pomocy piezoceramicznych przetworników (rys. 2.5).



Rys. 2.5 Czujniki umieszczone na mrówce firmy Festo. Źródło [17]

Zadaniem bionicznych mrówek jest znalezienie rozwiązania danego problemu w grupie. Dzięki badaniu zachowań mrówek, w niedalekiej przyszłości fabryki będą mogły składać się z takich autonomicznych i inteligentnych elementów, które będą same podejmować decyzje.

Atlas (rys. 2.6) to dwunożny robot o wysokości 150 cm i wadze 75 kg. Jego kończyny mają łącznie 28 stopni swobody. Może poruszać się zarówno na zewnątrz, jak i wewnątrz budynku. Jego głównymi zadaniami są: poruszanie się, omijanie przeszkód, przenoszenie rzeczy oraz skakanie przez małe przeszkody. Zbudowany jest z aluminium i tytanu. Co więcej, oświetlony jest niebieskimi światłami LED. W celu ominięcie przeszkód oraz utrzymania równowagi w terenie, używane są sensory LIDAR oraz system stereowizyjny. Robot Atlas w przyszłości ma być zastosowany w misjach poszukiwawczo-ratowniczych oraz w miejscach szkodliwych dla człowieka.

Rys. 2.6 Robot Atlas. Źródło [18]

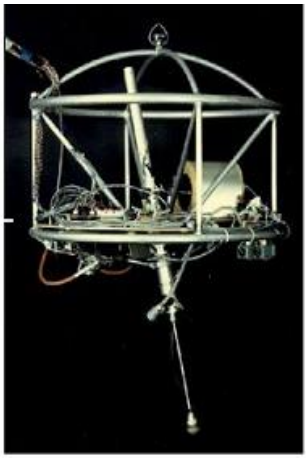
2.2. Klasyfikacja maszyn kroczących

Maszyny kroczące klasyfikuje się na podstawie liczby nóg, stopni swobody nóg oraz rodzaj stabilności chodu. Podział ze względu na stopnie swobody nóg jest uzależniony od liczby i rodzaju zastosowanych silników. Wyróżniamy roboty o nogach:

- z dwoma stopniami swobody – noga składa się z dwóch członów: uda i podudzia. Maszyna ma stałą wysokość. Nie może chodzić bokiem (ang. sidewalk). Skręt realizowany jest poprzez wydłużanie i skracanie kroku po wybranych stronach robota w zależności od kierunku skrętu;
- z trzema stopniami swobody – noga składa się z trzech członów: biodra (łac. coxa), uda (łac. femur) i goleni (łac. tibia). Dzięki temu, noga może wykonać skomplikowane manewry oraz ciało robota może być ustawione na różnej wysokości;
- z czterema stopniami swobody (ekstremalny przypadek) np. Quadruped robot 4DOF [58];

W klasyfikacji ze względu na liczbę nóg wyróżniamy roboty:

- jednonożne np. skoczek (Hopper), uczelnia MIT (rys 2.7);



Rys. 2.7 Jednonożny robot (ang. hopper). Źródło [29]

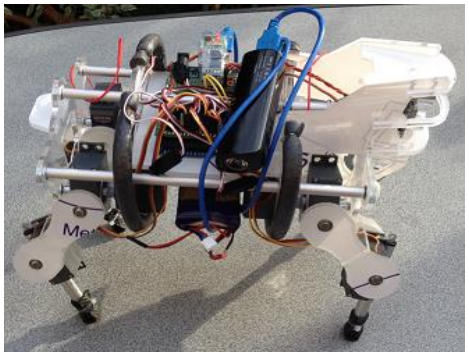
Na korpusie maszyny przymocowane są czujniki i elementy elektroniki, przy pomocy, których utrzymywany jest stan równowagi. Noga ma zmienną długość i zaczyna się od dwuosiowego biodra z napędem. Ruch odbywa się na zasadzie odwróconego, sprężynowego wahadła, a obrót wykonywany jest w fazie lotu maszyny. Robot może kontrolować swoją prędkość i wysokość skakania oraz dokonywać korekcji swojej pozycji.

- dwunożne – bipedy (ang. bipeds) np. Spring Flamingo, uczelnia MIT (rys. 2.8);

Rys. 2.8 Robot dwunożny Spring Flamingo. Źródło [29]

W ciele robota znajdują się silniki napędzające układ. Do bioder, kolan i kostek przymocowane są sensory np. obrotowe potencjometry oraz 6 napędów. Ruch nóg odbywa się na zasadzie odwróconego, sprężynowego wahadła.

- czworonożne (ang. quadrupeds) np. Quadruped Robot, amatorski projekt użytkownika Wingman (rys. 2.9);



Rys. 2.9 Czworonożny robot Quadruped. Źródło [43]

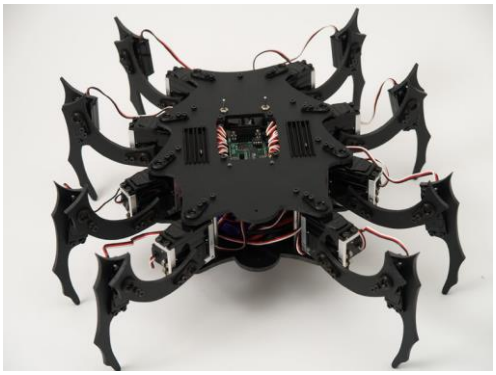
Naśladują one sposób przemieszczania się płazów, gadów i ssaków. Same utrzymują równowagę podczas postoju. Najczęściej poruszają się dwoma rodzajami chodu, kłusem (ang. trot/amble gait) oraz pełzaniem (ang. creep gait). Pełzanie polega na uniesieniu jednej nogi w górę, podczas gdy pozostałe trzy przesuwają korpus po ziemi. Jest to najwolniejszy z chodów. Natomiast, kłus polega na uniesieniu do góry dwóch nóg przekątnych, podczas gdy pozostałe dwie przesuwają korpus po ziemi. Jest to najszybszy z chodów.

- sześcionożne (ang. hexapods) np. Lynxmotion Phoenix 3DOF Hexapod, sklep Robotshop (rys. 2.10);

Rys. 2.10 Robot sześcionożny Lynxmotion Phoenix 3DOF Hexapod. Źródło [37]

Naśladują one sposób poruszania się owadów np. much, karaluchów. Typowymi rodzajami chodu dla tych maszyn są: pełzanie (ang. crawl gait), trójpodporowy (ang. tripod gait) oraz czterotaktowy (ang. quadruped gait). Powyższe chody zostały przedstawione w następny rozdziale.

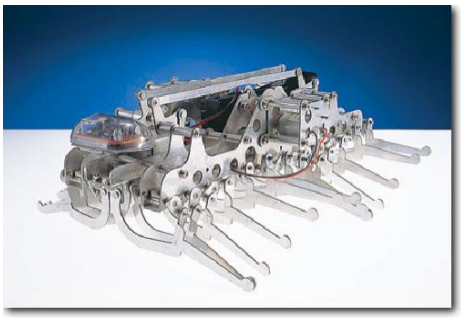
- ośmionożne (ang. octapod) np. Oxyopus, projekt użytkownika Kåre Halvorsen (rys. 2.11);



Rys. 2.11 Robot ośmionożny Oxyopus. Źródło [23]

Naśladują one sposób przemieszczania się pająków. Najczęściej, poruszają się dwoma rodzajami chodu: skorpiona (ang. scorpion) lub pająka (ang. simulated arachnid). Chód skorpiona jest kombinacją dwóch rodzajów chodu: pełzającego i trójpodporowego, z tym wyjątkiem, że wykorzystywane są do tego cztery nogi naprzemiennie. Chód pająka polega na uniesieniu w górę czterech odnóży diagonalnie i jest połączeniem dwóch rodzajów chodu: kłusu oraz pełzania.

- wielonożne (ang. multi-legged) np. Carl`s Electronic Gakkan Mechamo Centipede Robot (rys. 2.12);



Rys. 2.12 Robot wielonożny Gakkan Mechamo Centipede Robot. Źródło [59]

Przedstawiony na powyższym rysunku robot składa się z wielu segmentów z odnóżami. Jego sposób poruszania się przypomina ruch stonogi lub gąsienicy.

Klasyfikacja ze względu na rodzaj stabilności ruchu uzależniona jest od liczby nóg oraz rozmiaru maszyn kroczących. Wyróżnia się:

- ruch statycznie stabilny – dużo aktywnych stopni swobody, precyzyjnie zadane trajektorie nóg;
- ruch quasi–statycznie stabilny – mniej stopni swobody. Fazy niestabilne występują pomiędzy fazami stabilnymi, ale maszyna się nie przewraca;
- ruch dynamicznie stabilny – od kilku do kilkudziesięciu stopni swobody. Zmienna konfiguracja stabilności powoduje ruch postępowy, który jest stabilny dynamicznie.

2.3. Lokomocja maszyn kroczących

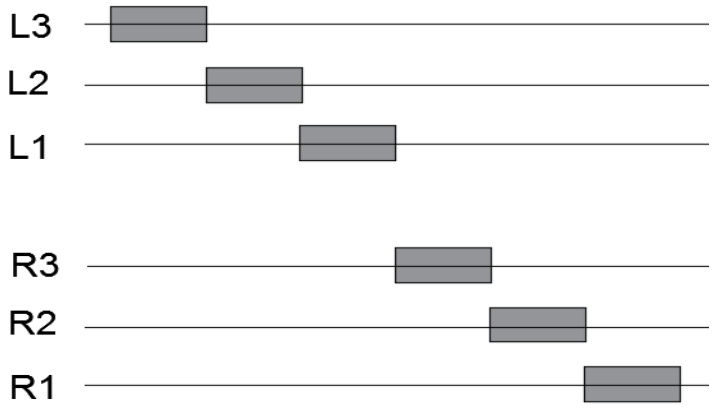
Chód maszyny kroczącej jest to forma lokomocji polegająca na przemieszczeniu ciała w dowolnym kierunku, z wykorzystaniem kończyn, jako aktuatorów podporowo–napędowych. Odbywa się on poprzez naprzemienne uniesienie każdej z nóg do góry, przesunięcie ich w kierunku ruchu, a następnie postawienie ich na podłożu oraz odepchnięcie w kierunku przeciwnym do ruchu. Wyjątek stanowią roboty jednonożne. W trakcie wykonywania tych manewrów mogą wystąpić wahania tułowia oraz nieutrzymywanie równowagi maszyny. Na sposób przemieszczenia się nóg mają wpływ: kolejność ich przestawiania, otaczające je środowisko oraz ich zastosowania. Można wyróżnić chody periodyczne lub swobodne. Co więcej, maszyna może poruszać się chodami statycznie lub dynamicznie stabilnymi.

2.3.1. Chody periodyczne

Chody periodyczne polegają na cyklicznym przestawianiu odnóży. Ciało robota przemieszcza się po linii prostej lub po lekkim łuku. Sześcionożne maszyny kroczące poruszają się jednym z trzech rodzajów chodu:

a) pełzający (ang. crawl gait);

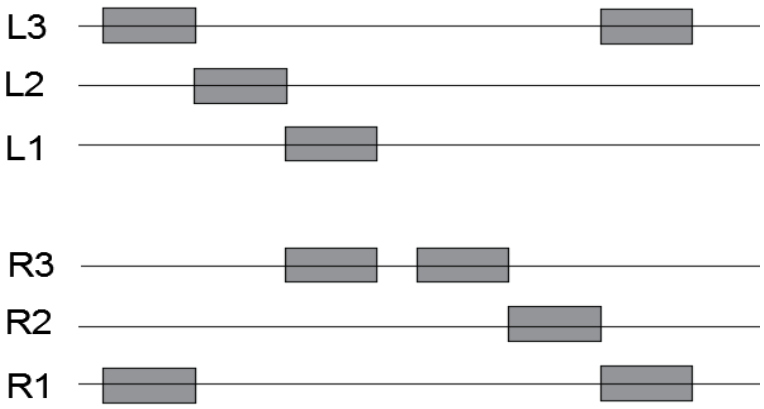
Chód pełzający (rys. 2.13), inaczej zwany chodem metachronicznym, jest najwolniejszym z dostępnych rodzajów chodu. W trakcie tego ruchu tylko jedna noga jest uniesiona do góry, podczas gdy pozostałe podtrzymują i przemieszczają korpus po ziemi. Ruch nóg następuje falowo, czyli jedna po drugiej. Faza przenoszenia nogi jest pięć raz krótsza niż faza przesuwania się nogi po podłożu. Cechuje się bardzo dużą stabilnością.



Rys. 2.13 Diagram chodu pełzającego. L1,L2,L3 – nogi po lewej stronie robota. R1, R2, R3 – nogi po prawej stronie robota. Szare prostokąty oznaczają fazy przenoszenia poszczególnych nóg. Źródło [4]

b) czterotaktowy (ang. quadruped gait);

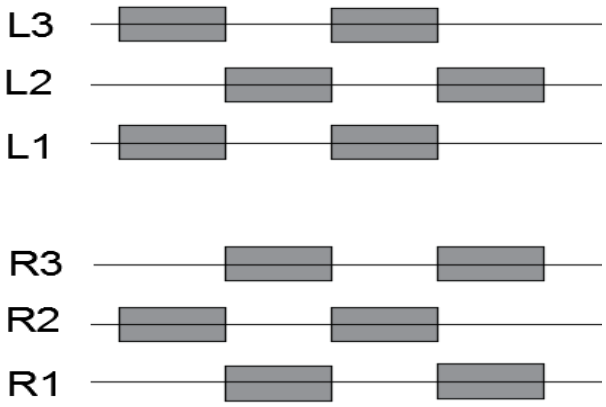
Chód czterotaktowy (rys. 2.14) jest znacznie szybszy od chodu pełzającego, ale rzadko stosowany u robotów sześcionożnych. Polega on na podniesieniu dwóch nóg jednocześnie w dwóch fazach, a w następnych dwóch już tylko jedna noga jest uniesiona, podczas gdy reszta przesuwa ciało po podłożu.



Rys. 2.14 Diagram chodu czterotaktowego. L1,L2,L3 – nogi po lewej stronie robota. R1, R2, R3 – nogi po prawej stronie robota. Szare prostokąty oznaczają fazy przenoszenia poszczególnych nóg. Źródło [4]

c) trójpodporowy (ang. tripod gait);

Chód trójpodporowy (rys 2.15) to najszybszy sposób poruszania się robota statycznie lub quasistatycznie stabilnego W trakcie poruszania się trzy diagonalne nogi przemieszczają ciało robota po podłożu, podczas gdy trzy pozostałe są uniesione w górze.



Rys. 2.15 Diagram chodu trójpodporowego. L1,L2,L3 – nogi po lewej stronie robota. R1, R2, R3 – nogi po prawej stronie robota. Szare prostokąty oznaczają fazy przenoszenia poszczególnych nóg. Źródło [4]

2.3.2. Chody swobodne

W chodzie swobodnym na bieżąco jest wybierana noga, która ma być podniesiona do góry. Decyzja jest podejmowana na podstawie badań otaczającego robota środowiska. Nie powtarza się ta sama sekwencja ustawienia nóg. Chód ten używany jest w trudnych warunkach m, in. przez owady, które szybko dostosowują się do otaczającego ich środowiska. Zaletą ruchu swobodnego jest to, że maszyna może łatwo przemieszczać się po skomplikowanym terenie. Jego wadą jest trudna implementacja w kodzie. Jedną z jego odmian jest chód za przewodnikiem (ang. follow the leader). Polega on na wyborze miejsc na stabilnym podłożu, gdzie mają być położone nogi przednie. Ich poprzednie miejsca zajmują nogi następujące po nich. Każda z nóg zajmuje miejsce nogi ją poprzedzającej.

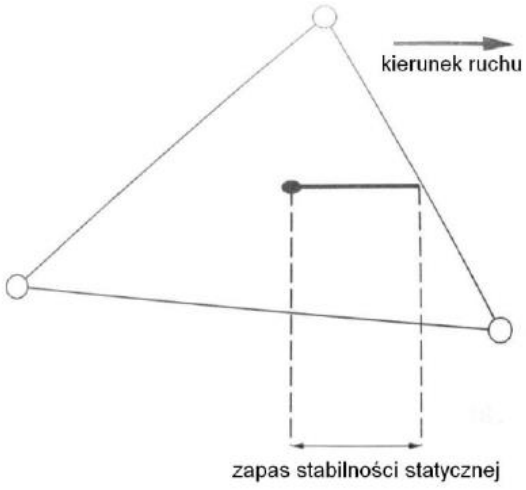
Ciekawy przykład ruchu swobodnego został zastosowany w czteronożnej maszynie kroczącej „Little Dog" (rys. 2.16), skonstruowanej przez firmę Boston Dynamics na zamówienie amerykańskiej agencji wojskowej DARPA. Maszyna ma 12 stopni swobody. Na przodzie i po bokach znajdują są kamery, które badają otaczające środowisko. Generacja ruchu odbywa się w trzech fazach: wybór chodu (ang. gait selection), etap ustawienia ciała robota (ang. quad shift phase) i etap wykonania kroku (ang. swing phase).

Rys. 2.16 Maszyna czteronożna „Little Dog" firmy Boston Dynamics. Źródło [5]

W fazie pierwszej następuje wybór algorytmu generującego ruch maszyny kroczącej. Dobierany jest z uwzględnieniem rodzaju podłoża, po jakim porusza się robot. Dla podłoża płaskiego, bez przeszkód wybierany jest galop. Dla terenu bardziej skomplikowanego generowany jest ruch za pomocą wielokierunkowego algorytmu chodu. W fazie drugiej dobierana jest odpowiednia stabilizacja ciała maszyny, tak, aby jak najlepiej zachowywało równowagę podczas poruszania się po skomplikowanym terenie. Odbywa się to kosztem utraty znacznej prędkości maszyny. W ostatniej fazie określany jest teren i dobierana jest pozycja, jaką dana stopa ma przyjąć. Teren klasyfikowany jest jako do przejścia lub nie. Wybierane są także: wysokość kroku i trajektoria nogi.

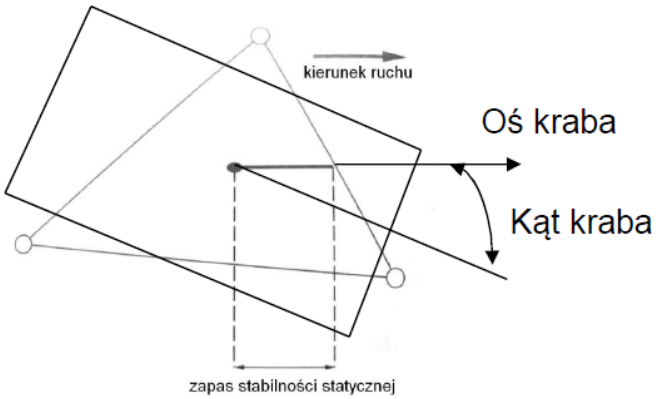
2.3.3. Stabilność chodu

Zazwyczaj maszyny kroczące poruszają się chodem statycznie stabilnym. W tym chodzie, pionowy rzut środka ciężkości umiejscowiony jest we wnętrzu wielokąta podparcia, który opiera się na punktach styku nogi z podłożem. Odległość między rzutem środka ciężkości, a krawędzią wielokąta podparcia określana jest mianem zapasu stabilności statycznej
(rys. 2.17).



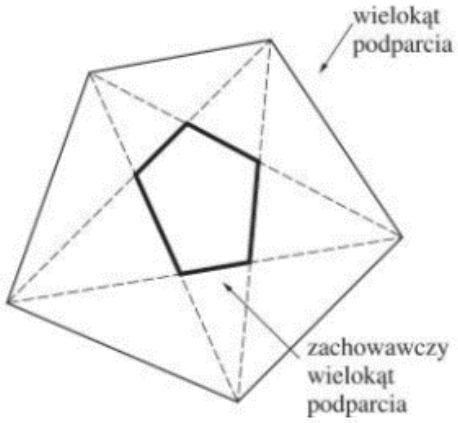
Rys. 2.17 Zapas stabilności statycznej. Źródło [1]

Zapas stabilności statycznej dobierany jest eksperymentalnie. Powinien być tak dobrany, aby pominięte efekty dynamiczne i siły zewnętrzne nie wpływały na stabilność maszyny kroczącej. Według innej definicji, zapas stabilności to najmniejsza odległość rzutu środka ciężkości od krawędzi wielokąta podparcia. Z tą definicją wiążą się pojęcia osi i kąta kraba. Oś kraba to oś przechodząca przez rzut środka ciężkości i skierowana jest tak jak wektor kierunku ruchu. Kąt kraba to kąt pomiędzy osią symetrii ciała, a osią kraba (rys. 2.18). Chód kraba to taki sposób poruszania się, w którym kat kraba jest różny od zera.



Rys. 2.18 Oś i kąt kraba. Źródło [1]

W dynamicznie stabilnym chodzie stosuje się zachowawczy wielokąt podparcia (rys. 2.19).



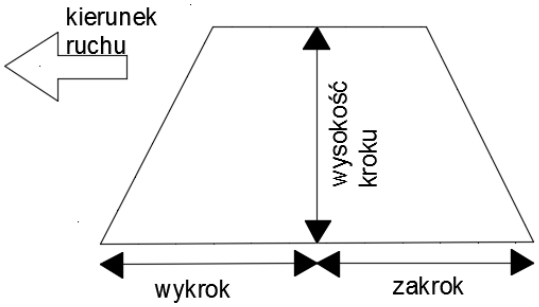
Rys. 2.19 Zachowawczy wielokąt podparcia. Źródło [1]

W tym ruchu rzut środka ciężkości może znajdować się poza wielokątem podparcia pod warunkiem, że nie wpływa on na stabilność robota. Dopuszczalne są zakłócenia, które nie powodują zbyt dużych odchyleń w trajektorii zadanej oraz brany jest pod uwagę brak punktu podparcia dla nogi. Zazwyczaj ruch dynamicznie stabilny stosowany jest w robotach z maksymalnie czterema nogami, a statycznie stabilny, z co najmniej czterema nogami.

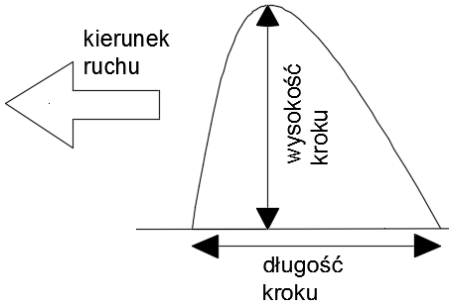
W tej pracy założono, że maszyna krocząca porusza się tylko chodami periodycznymi, które są statycznie stabilne, bo oprócz kamery, nie ma innych czujników. Implementacja ruchu dynamicznie stabilnego bez czujników jest bardzo trudnym zadaniem. Dodatkowo, robot ma wystarczającą liczbę stopni swobody, by poruszać się ruchem statycznie stabilnym.

2.3.4. Trajektoria nogi

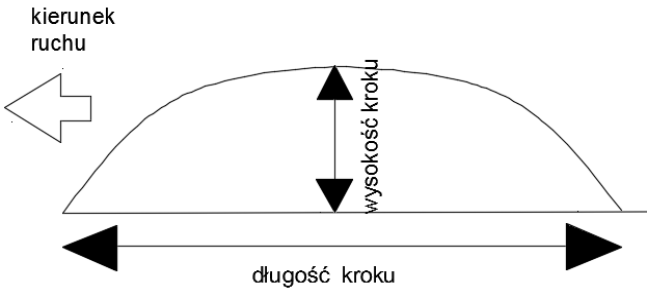
Podczas chodu, noga może poruszać się po trajektorii trapezowej (rys. 2.20), eliptycznej (rys. 2.21) lub trójkątnej (rys. 2.22). Zaletą trajektorii trójkątnej jest jej mała liczba odcinków, co powoduje, że programowa implementacja tej trajektorii jest prosta. Trajektoria trapezowa nogi zalecana jest w przypadku omijania przeszkód, ponieważ końcówka nogi podnosi się znacznie szybciej niż w przypadku trajektorii trójkątnej. Trajektoria eliptyczna jest najtrudniejsza do implementacji, ale najdokładniej z wszystkich wyżej wymienionych trajektorii, odzwierciedla ruch nogi owada. Poprzez zmianę parametrów takich jak środek ciężkości elipsy oraz współrzędnej osi głównej i małej można wygenerować różnego rodzaju chody.



Rys. 2.20 Trajektoria nogi w kształcie trapezu



Rys. 2.21 Trajektoria nogi w kształcie trójkąta. Źródło [3]

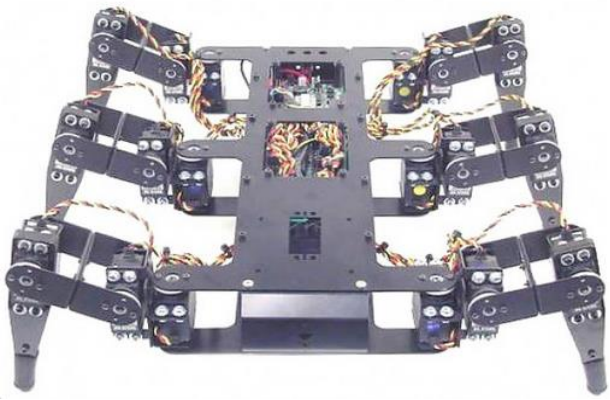


Rys. 2.22 Trajektoria nogi w kształcie elipsy. Źródło [3]

3.Opis robota kroczącego

3.1. Konstrukcja maszyny kroczącej

Do zrealizowania projektu została wykorzystana konstrukcja maszyny sześcionożnej Lynxmotion BH3 Hexapod Robot Kit (rys. 3.1) ze sklepu internetowego RobotShop.com. Swoim kształtem przypomina ona karalucha, co uwidacznia poniższe zdjęcie.



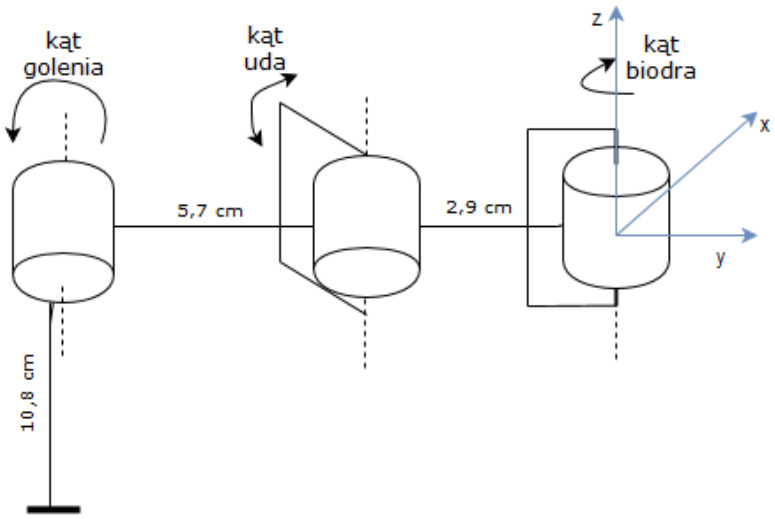
Rys. 3.1 Robot Lynxmotion BH3 Hexapod Robot Kit. Źródło [35]

Robot ma dwupoziomowy, prostokątny i aluminiowy korpus o długości 33 cm, szerokości 8,9 cm i wysokości 5,1 cm [35]. Zarówno z przodu jak i z tyłu wygląda tak samo, dlatego przód robota został przyjęty umownie. Wzdłuż korpusu zostały przymocowane po bokach symetrycznie 3 pary aluminiowych odnóży. Zakończone są one kwadratową podstawą, na którą naciągnięto gumową powłokę tak, aby maszyna podczas stania lub chodu się nie ślizgała. Dodatkową funkcją gumowych powłok jest zabezpieczenie przed zniszczeniem końcówek nóg.

Rys. 3.2 Jedna z nóg maszyny kroczącej

W skład każdej nogi wchodzą 3 serwomechanizmy (rys. 3.2), które poruszają ich poszczególnymi odcinkami: udo–biodro, biodro–goleń, goleń–końcówka nogi. Dzięki takiej liczbie serw, możliwe było zaimplementowanie więcej niż jednego rodzaju chodu. Łącznie maszyna ma 18 serwomechanizmów HiTecHS – 645MG [34]. W zależności od potrzeb konstruktora, serwomechanizmy mogą być zasilane napięciem od 4,8 do 6 V. Do celów pracy została wybrana opcja zasilania 5 V. Serwa obracają się w przedziale od 0^0 do 180^0. Ich zaletami są metalowe tryby z zębatką MP i cicha praca. Cała długość nogi wynosi 19,4 cm. Poszczególne długości odcinków nogi zostały przedstawione poniżej:

- biodro–udo wynosi 2,9 cm,
- udo–piszczel wynosi 5,7 cm,
- goleń–końcówka nogi wynosi 10,8 cm.



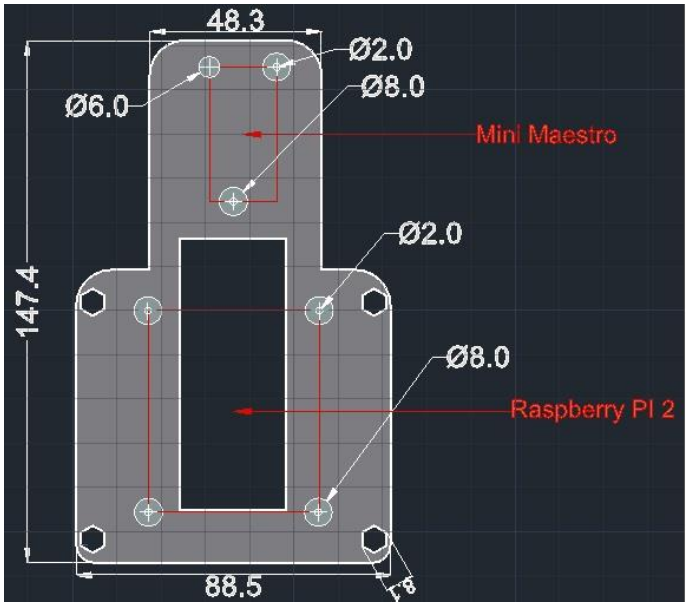
Rys. 3.3 Schemat blokowy nogi robota

3.1.1. Podkładki dla układu sterowania i wizji robota

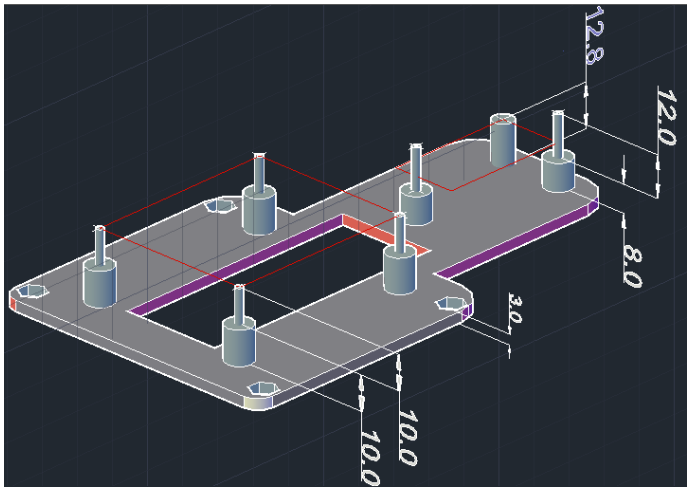
W celu zamontowania kamery, jednopłytkowego komputera Raspberry PI i sterownika Mini Maestro na maszynie kroczącej zaprojektowano w programie AutoCad dwie podstawki: dużą i małą. Następnie wydrukowano je na drukarce 3D.

Do dużej podstawki (rys. 3.4) przymocowano mikrokomputer wraz z sterownikiem serw. Cały układ zamontowano na robocie tak jak przedstawiono na rysunku 3.10. Podstawka ma sześć otworów w kształcie sześciokątów, przez które przechodzą pręty maszyny.

Dodatkowo, zaprojektowano wystające bolce w kształcie walców do nałożenia Mini Maestro i Raspberry PI 2. Głównymi założeniami przy projektowaniu podstawki były niewielkie rozmiary oraz wystarczająca przestrzeń do zamontowania płytek elektronicznych.

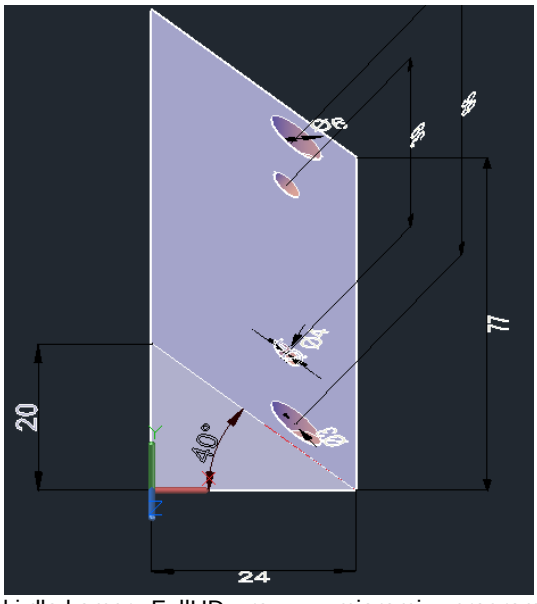


Rys.3.4 Konstrukcja podstawki dla Mini Maestro i Raspberry PI 2 wraz z wymiarami w programie AutoCad w widoku z góry. Rozmiary elementów na rysunku podane są w milimetrach.

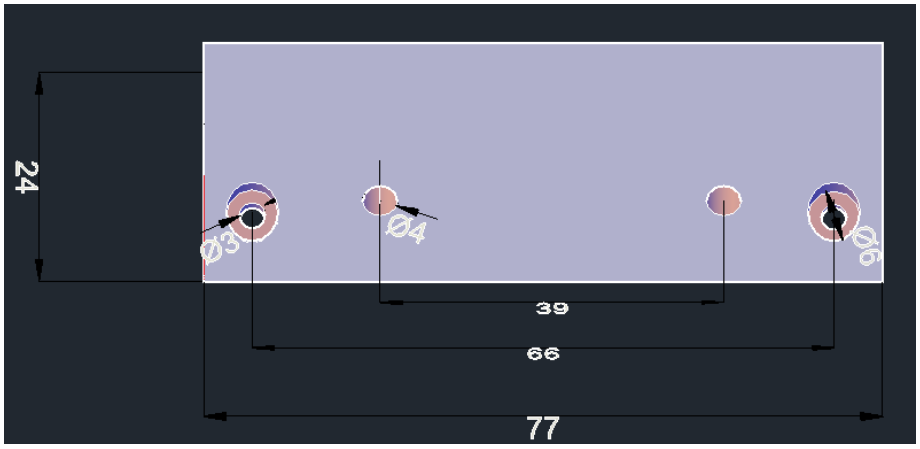


Rys. 3.5 Konstrukcja podstawki dla Mini Maestro i Raspberry PI 2 wraz z wymiarami w programie AutoCad w widoku z boku. Rozmiary elementów na rysunku podane są w milimetrach

Krótsza podstawka umieszczona jest z przodu, na górnej części robota. Do niej przymocowana jest kamera FullHD. Podstawka ma kształt graniastosłupa o podstawie trójkąta prostokątnego, w którym miary kątów przy przeciwprostokątnej wynoszą 50^0 oraz 40^0 tak jak na rysunku 3.6.

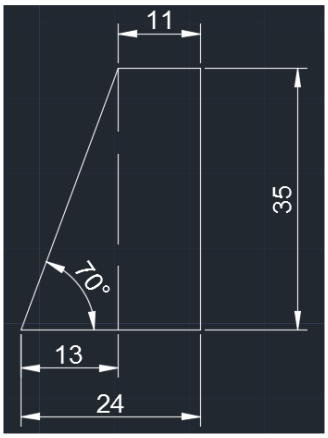


Rys. 3.6 Konstrukcja podstawki dla kamery FullHD wraz z wymiarami w programie AutoCad w widoku z boku. Rozmiary elementów na rysunku podane są w milimetrach.



Rys. 3.7 Konstrukcja podstawki dla kamery FullHD wraz z wymiarami w programie AutoCad w widoku z góry. Rozmiary elementów na rysunku podane są w milimetrach.

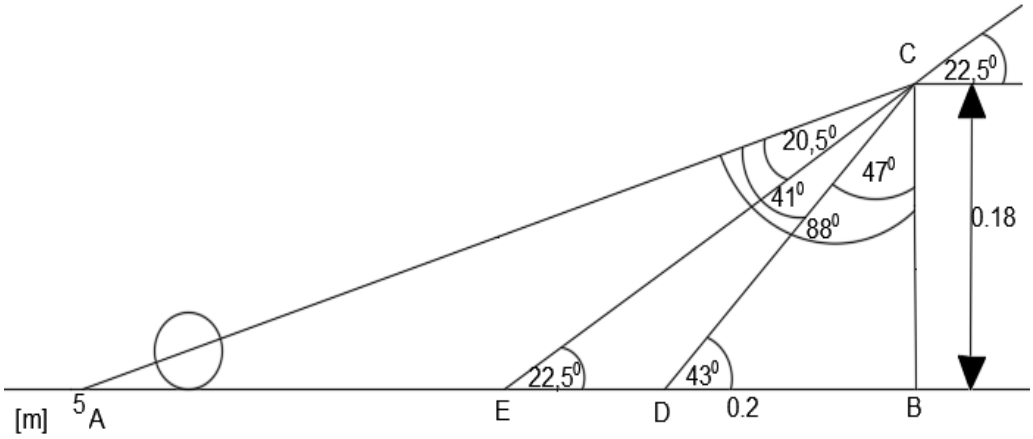
Przy projektowaniu podstawki były brane pod uwagę wymiary kamery oraz wysokość robota. Wymiary obudowy kamery zostały przedstawione na poniższym rysunku (rys. 3.8).



Rys. 3.8 Obudowa kamery wraz z wymiarami w programie AutoCad w widoku z boku. Rozmiary elementów na rysunku podane są w milimetrach.

Obiektyw jest umieszczony na wysokości 18 cm nad podłożem. Założono, że robot może najdalej widzieć piłkę z odległości 5 metrów, a najbliżej z 20 centymetrów od siebie.

Na rysunku 3.9 przedstawiono wyżej wymienione założenia oraz zaznaczono obliczone kąty, których użyto do wyznaczenia kąta nachylenia obudowy kamery do podstawy robota. Do obliczeń użyto wyżej wyliczony kąt nachylenia kamery.



Rys. 3.9 Rysunek pomocniczy do wyznaczenia nachylenia kamery do robota w programie AutoCad w widoku z boku

Wyliczone miary kątów pomocniczych:

Miara kąta ACB = $\text{atan}(5/0.18) \approx 88^0$

Miara kąta widzenia w pionie kamery : kąta ACD $\approx 41^0$

Miara kąta DCB: ACB – ACD = $88^0 - 41^0 = 47^0$

Miara kąta CDB = 90^0 – DCB = $90^0 - 47^0 = 43^0$

DB = 0.18/tan(43^0) ≈ 0.2

EC dwusieczna kąta ACD:

Miara kąta ECD= ACE = 0.5* ACD = $0.5*41^0 = 20.5^0$

Miara kąta ECB: ECD+ DCB = $20.5^0 + 47^0 = 67.5^0$

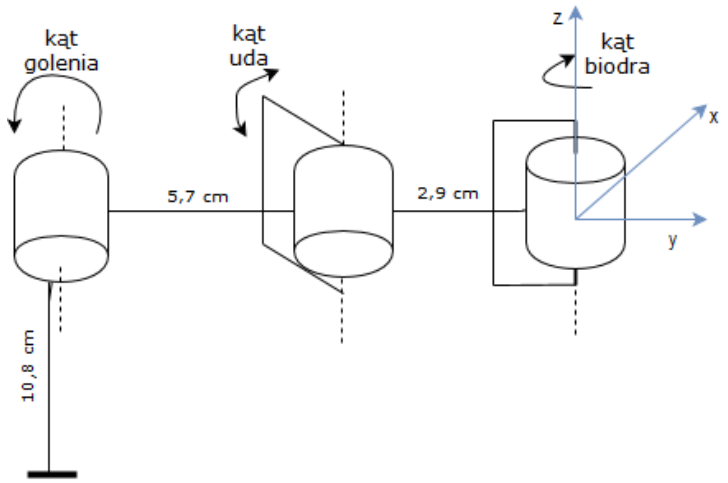
Żądana miara kąta nachylenia kamery CEB = 90 – ECB = 90 – 67.5^0= 22.5^0
Obudowa kamery kieruje obiektyw o 20^0 do góry, więc żeby otrzymać żądaną miarę kąta nachylenia kamery, podstawka powinna być nachylona o $42,5^0$ do podłoża:
Miara kąta podstawki = CEB + 20.5^0= 22.5^0 + 20.5^0 = 43^0
Wyliczony powyżej kąt nachylenia podstawki zaokrąglono do 40^0, co powoduje, że kamera patrzy nieznacznie dalej niż założono.

Rys. 3.10 Podstawki dla Raspberry PI, Mini Maestro i kamery, wydrukowane na drukarce 3D i zamontowane na robocie

3.2. Kinematyka odwrotna i prosta

3.2.1. Układ współrzędnych nogi

Do określenia ruchu nogi robota, wymagany jest opis jej końcówki w układzie współrzędnych. Do tego celu, został wybrany kartezjański, trójwymiarowy układ współrzędnych, ponieważ przedstawia on w prosty sposób ruch przegubów kończyn robota. Wykorzystanie tego układu powoduje, że w niektórych przypadkach obliczenia kinematyki odwrotnej stają się dość skomplikowane.



Rys. 3.11 Układ współrzędnych prawej nogi robota

Na powyższym schemacie blokowym (rys. 3.11) przedstawiono lokalny układ współrzędnych dla nóg prawych i lewych.Ponadto dla ułatwienia obliczeń kinematyki przyjęto, że środek lokalnego układu współrzędnych (0,0,0) w każdej nodze jest umiejscowiony w samym środku serwomechanizmu odpowiedzialnego za poruszanie stawem biodrowym robota. Założono także, że kierunki osi układu współrzędnych mają następujące zwroty:

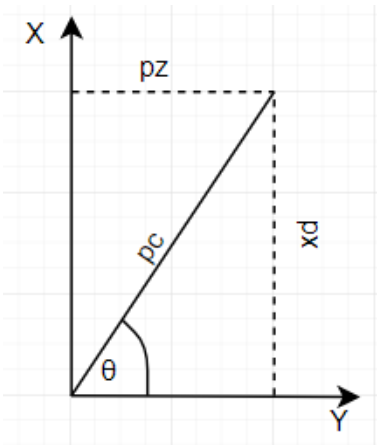
- oś X – jest to oś, która przebiega wzdłuż korpusu maszyny kroczącej. Kierunek dodatni osi wskazuje na przód robota, a ujemny na tył;
- oś Y – jest to oś prostopadła do korpusu maszyny kroczącej, która wskazuje na jej lewy i prawy bok. Kierunek ujemny osi wskazuje wnętrze robota, a dodatni na zewnątrz;
- oś Z – jest to oś, która wyznacza górę lub dół maszyny kroczącej. Kierunek ujemny wskazuje do góry, a dodatni na dół;

3.2.2. Kinematyka odwrotna

Do manipulowania (poruszania) kończyną robota tak, aby osiągnęła ona pożądane ustawienie, stosuje się obliczenia do rozwiązania zadania kinematyki odwrotnej lub kinematyki prostej oraz kartezjański układ współrzędnych. W kinematyce odwrotnej wyliczane są miary kątów poszczególnych przegubów z podanych współrzędnych końcówki nogi. Dzięki nim można precyzyjnie ustawić nogi w zadanych pozycjach oraz co najważniejsze zaplanować ich trajektorię. Wyliczone rozwiązania zadania kinematyki odwrotnej mogą być niejednoznaczne tzn. do osiągnięcia zadanej pozycji przez kończynę istnieje więcej niż jeden zestaw miar kątów. Z takich rozwiązań wybiera się jedno z najbardziej pasujących do sytuacji lub uwzględnia się ograniczenia fizyczne kończyny. W tej pracy większość niejednoznaczności została wyeliminowana poprzez wprowadzenie w programie maksymalnych oraz minimalnych wartości miar kątów, jakie dane serwomechanizm może fizycznie osiągnąć.

Do rozwiązania zadania kinematyki odwrotnej wykorzystuje się równania trygonometryczne lub notację Denavita–Hartenberga i przekształcenia macierzy. Wyliczenia rozwiązania zadania kinematyki odwrotnej w przypadku tego projektu nie były skomplikowane, dlatego została wykorzystana do tego celu trygonometria. Rozwiązaniami są miary trzech kątów umiejscowionych odpowiednio przy: biodrze – θ, udzie – μ, i goleni – Ψ.

Miary kątów zostały wyliczone przy użyciu punktów współrzędnych końcówki: p_x, p_y, p_z i długości poszczególnych członów nogi. Rysunek 3.12 przedstawia widok z góry na odnóże maszyny kroczącej w płaszczyźnie XY.



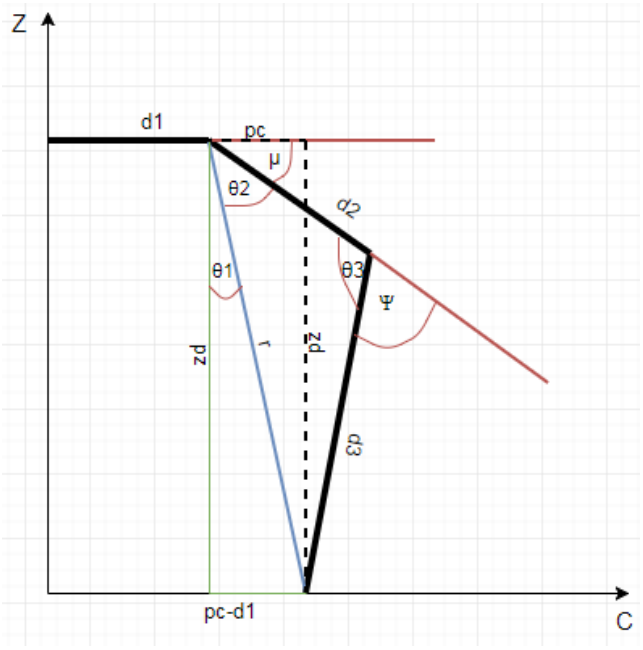
Rys. 3.12 Widok z góry na nogę maszyny kroczącej w płaszczyźnie XY

Pierwsze z trzech równań zadania kinematyki odwrotnej wylicza miarę kąta przy biodrze – θ (zaznaczony na rysunku 3.12):

$$\theta = arctg(\frac{p_x}{p_y}) \quad (1)$$

W celu wykonania dalszych obliczeń, niezbędne jest zdefiniowanie wszystkich przegubów w jednej płaszczyźnie o nazwie C. Jest ona prostopadła do płaszczyzny XY, przechodzi przez środek układu współrzędnych (0,0,0) i przecina płaszczyznę ZY pod kątem θ. Na podstawie rysunku 3.12, równanie określające długość nogi p_c na płaszczyźnie C opisane jest następująco:

$$p_c = \sqrt{p_x^2 + p_y^2} \tag{2}$$



Rys. 3.13 Noga maszyny kroczącej w płaszczyźnie ZC

Do obliczeń pozostałych miar kątów został wykorzystany rysunek 3.13, który przedstawia nogę robota widzianą z boku w płaszczyźnie ZC. Na rysunku zostały też zaznaczone długości poszczególnych członów: d1 – biodro–udo, d2 – udo–goleń, d3 – goleń–końcówka nogi, szukane miary kątów oraz kąty pomocnicze i odległość pomiędzy udem, a końcówką odnóża. Na podstawie rysunku 3.13, odległość r pomiędzy udem, a końcówką odnóża opisana jest równaniem:

$$r = \sqrt{(p_c - d1)^2 + p_z^2} \tag{3}$$

Mając wyliczoną długość r i korzystając z twierdzenia cosinusów o związku pomiędzy kątem i bokami trójkąta, można wyliczyć kąt pomocniczy $\theta 2$ przy udzie z następującego równania:

$$d3^2 = r^2 + d2^2 - 2 * r * d2 * cos(\theta 2) \tag{4}$$

Po przekształceniu powyższego wzoru, miara kąta pomocniczego $\theta 2$ wynosi:

$$d2 = arccos(\frac{r^2 - d3^2 + d2^2}{2 * r * d2^2}) \quad (5)$$

Do wyliczenia miary kąta pomocniczego $\theta 1$ można skorzystać z twierdzenia o trójkącie prostokątnym, którego długości boków przyprostokątnych wynoszą odpowiednio p_z, p_c i d1:

$$\theta 1 = arctg(\frac{p_c - d1}{p_z}) \quad (6)$$

Mając miary kątów pomocniczych $\theta 1$i $\theta 2$, można wyliczyć miarę kąta przy udzie – μ:

$$\mu = 180^0 - (90^0 + \theta 1 + \theta 2) \quad (7)$$

Po przekształceniu powyższego wzoru:

$$\mu = 90^0 - \theta 1 - \theta 2 \quad (8)$$

Wykorzystując twierdzenie cosinusów oraz długości boków trójkąta: d3, d2 i r, można wyliczyć miarę kąta pomocniczego $\theta 3$:

$$r^2 = d3^2 + d2^2 - 2 * d3 * d1 * cos(\theta 2) \quad (9)$$

Przekształcając powyższy wzór:

$$\theta 3 = arccos(\frac{-r^2 + l3^2 + l1^2}{2 * l3 * l1}) \quad (10)$$

Zatem miara kąt przy goleni Ψ wynosi*:*

$$\Psi = 180^0 - \theta 3 \quad (11)$$

Powyższe równania wchodzą w skład algorytmu do obliczenia zadania kinematyki odwrotnej. Co więcej, w celu wyeliminowania niejednoznacznych rozwiązań przyjęto, że miary kątów przy biodrze i udzie są w przedziale (-90^0,$+90^0$). W sytuacji, gdy wyliczona miara kąta jest spoza podanego przedziału to następuje obrócenie tego kąta o 180 stopni. Dodatkowo zmienna p_c przyjmuje wartość ujemną. W tablicy *tab_of_servos*, która przechowuje minimalne, środkowe oraz maksymalne pozycje poszczególnych serw, zostały przyjęte stałe ograniczenia pozycji minimalnej i maksymalnej. Dla serwa przy biodrze, minimalna osiągalna pozycja wynosi 950, a maksymalna 2500.

3.2.3. Kinematyka prosta

Rozwiązanie kinematyki prostej jest działaniem odwrotnym do obliczenia wyniku kinematyki odwrotnej. Jej celem jest uzyskanie położenia końcówki nogi przy użyciu podanych miar kątów oraz długości poszczególnych odcinków kończyny. Do rozwiązania zadania kinematyki prostej stosuje się trygonometrię lub notację Denavita–Hartenberga. W tej pracy została wykorzystana prosta trygonometria z powodu małej złożoności obliczeń.

Na podstawie rysunków 3.12 i 3.13 oraz danych z podrozdziału o kinematyce odwrotnej, obliczenia rozwiązania zadania kinematyki prostej przebiegają następująco:

- długość nogi p_c i punkt współrzędny p_z z rysunku 3.13:

$$p_c = l1 + l2 * cos(\theta) + l3 * cos(\theta + \Psi) \quad (12)$$

$$p_z = l2 * sin(\theta) + l3 * sin(\theta + \Psi) \quad (13)$$

- punkty współrzędnych p_x, p_y na podstawie rysunku 3.12 i wartości p_c:

$$p_x = p_c * sin(\theta) \quad (14)$$

$$p_y = p_c * cos(\theta) \quad (15)$$

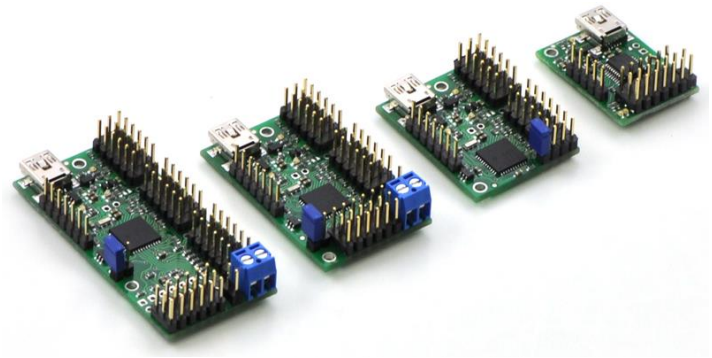
Kinematyka prosta została zaimplementowana w celu sprawdzenia poprawności działania kinematyki odwrotnej. Co więcej, na zasadzie kinematyki prostej działa funkcja *set_servos_in_leg*, która ustawia pozycje serwomechanizmów w całej nodze.

3.3. Sterowanie maszyną kroczącą

Z aktualnie dostępnych na rynku sterowników serw i komputerów jednopłytkowych, do realizacji celów tej pracy zostały wybrane: sterownik serw Mini Maestro firmy Pololu oraz jednopłytkowy komputer Raspberry PI 2. Obie płytki elektroniczne mają niewielkie rozmiary oraz uniwersalne porty zasilające typu microUSB i dodatkowo port zasilający dla sterownika. Ponadto, mikrokomputer Raspberry PI 2 oferuje wystarczającą moc obliczeniową potrzebną do realizacji tego projektu.

3.3.1. Mini Maestro

Mini Maestro firmy Pololu to kontroler serwomechanizmów i kart I/O. Występuje w wersjach 6-, 12-, 18- i 24- kanałowej (rys. 3.14). W zależności od liczby kanałów Mini Maestro ma następujące wymiary: 2.16x3 cm. (6 kanałów), 3.6x2.8 cm. (12 kanałów), 4.57x2.8 cm. (18 kanałów),5.84x2.8 cm. (24 kanałów)



Rys. 3.14 Pololu Mini Maestro w wersji 24-, 18-, 12- i 6- kanałowej. Źródło [19]

Kontroler można połączyć bezpośrednio za pomocą kabla microUSB do komputera, laptopa lub komputera jednopłytkowego np. Arduino, Raspberry PI etc. Innym sposobem komunikacji ze sterownikiem jest połączenie jednej z wyżej wymienionych płytek przez port szeregowy TTL. Kontroler przetwarza sygnały analogowe i cyfrowe. Ma wejście sterujące microUSB, 5 V port zasilający, piny: do podłączenia zasilania, sygnału sterującego oraz uziemienia dla serwomechanizmów, diody sygnalizujące pracę urządzenia oraz interfejs szeregowy UART. Najprostszą formą sprawdzenia poprawności działania sterownika jest uruchomienie programu udostępnionego na stronie internetowej producenta [19].

Znajdują się w nim podstawowe komendy napisane w niskopoziomowym języku programowania C, które umożliwiają sterowanie serwem. Można wybrać prędkość, położenie i czas reakcji serwomechanizmu. Dodatkowo użytkownik może wybrać sposób komunikacji z sterownikiem oraz szerokość pulsu sterującego do sterowania serwami. Oprogramowanie automatycznie wykrywa aktualną liczbę podłączonych serwomechanizmów.

3.3.2. Raspberry Pi 2

Raspberry PI 2 (rys. 3.15) to kolejna generacja jednopłytkowego mikrokomputera z rodziny Raspberry PI, która ma liczne zastosowanie w rozwiązaniach komercyjnych jak i amatorskich na świecie. Podstawowym założeniem twórców Raspberry było nauczanie podstaw informatyki za pomocą mikrokomputera w szkołach średnich i na uczelniach wyższych. Obecnie mikrokomputery z tej rodziny oferują na tyle dużą moc obliczeniową i dodatkowe funkcje sprzętowe jak m.in. komunikacja przez Bluetooth, aby mieć ogromne zastosowanie w takich dziedzinach jak IT (np. serwery), elektronika (np. stacje pogodowe), robotyka (np. hexapody) i rozwiązania amatorskie (np. ekspress do kawy MoccaPI, wyświetlacz dotykowy do samochodu Carputer).

Rys. 3.15 Raspbery PI 2. Źródło [30]

Pierwsza generacja Raspberry pojawiła się w roku 2012 i od tamtego czasu doczekała się już trzech wersji. Z każdą nową edycją ulepszane są lub dodawane nowe funkcje sprzętowe takie jak wbudowany moduł WiFi lub Bluetooth. Wszystkie generacje mikrokomputera zawierają zintegrowane układy scalone (SoC) z rodziny BroadCom, które są oparte na wydajnych procesorach ARM i integrują wszystkie potrzebne elementy komputera takie jak karta graficzna i CPU. Raspberry PI 2 oferuje 1 GB RAMu, 4 porty USB, port ethernetowy do 100 Mb/s i port HDMI do przesyłania obrazu.

Co więcej, płytka ma od 26 do 40 pinów I/O, co umożliwia podłączenie do niej czujników, które komunikują się za pomocą innych interfejsów niż USB np. SPI, szeregowego i I^2C. Obraz systemu oraz pamięć dla potrzeb użytkownika są przechowywane na karcie MicroSD. Dla mikrokomputerów z rodziny Rasperrby PI powstały dedykowane, wbudowane systemy operacyjne o otwartym kodzie źródłowym: Raspbian, Ubuntu, Windows 10 IOT, RISC OS, NOOBS. Z powodu coraz liczniejszych zastosowań, popyt na nowe funkcje wciąż rośnie, dlatego twórcy, by częściowo zaspokoić wymagania użytkowników, chętnie oferują odpłatne akcesoria takie jak: kamera FullHD (rys. 3.16), Gertboard, kamera na podczerwień (PI NoIR) i karta rozszerzeń HAT. Do realizacji celów tej pracy została wybrana kamera FullHD, która jest małym urządzeniem o wymiarach 2.5x2x0.9 cm.

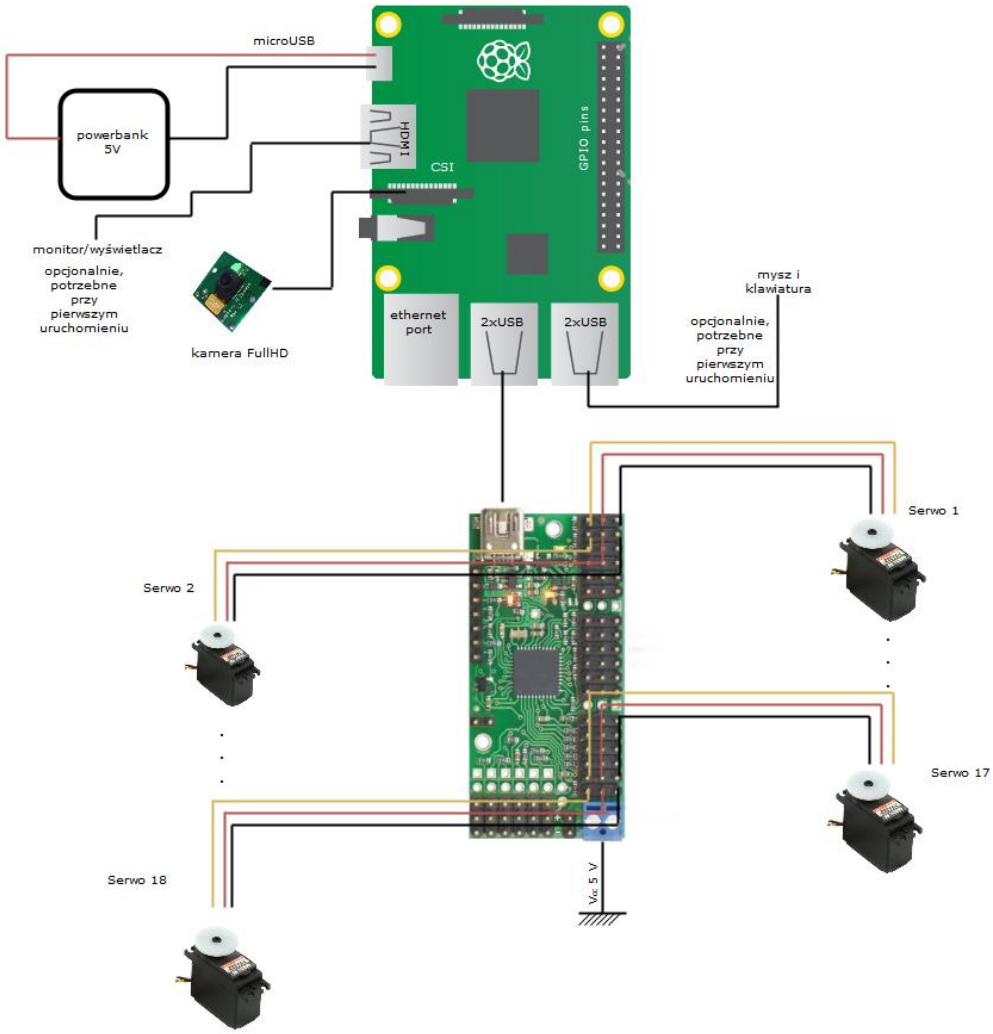
Rys. 3.16 Kamera fullHD Raspberry PI. Źródło [31]

Kamera ma płaski, elastyczny kabel taśmowy, który jest wtykany niebieskim końcem do dedykowanego interfejsu kamery CSI, znajdującego się między portami ethernetowymi oraz HDMI. Oferuje też obraz w rozdzielczości 1080p, 720p lub 640x480p. Ma 5 megapikseli. Poziomy kąt widzenia obiektywu wynosi 53.50^0, a pionowy – 41.41^0.

3.3.3. Układ sterujący maszyną kroczącą

W układzie sterowania maszyną kroczącą, sterownik serw Mini Maestro kontroluje oraz zasila 18 serwomechanizmów. Są one podłączone do pinów sterownika poprzez 3-żyłowe przewody. Czarny przewód (masa) znajduje się po zewnętrznej stronie płytki, a żółty (sygnał) po wewnętrznej stronie. Schemat połączeń serwomechanizmów i sterownika przedstawiony jest na rysunku 3.17. Sterownik zasilany jest 5 V z zewnętrznego źródła zasilania – zasilacza komputerowego. Komunikacja z mikrokomputerem odbywa się za pomocą interfejsu miniUSB. Dodatkowo, mikrokomputer za pomocą interfejsu CSI wymienia dane z kamerą.

Systemem operacyjnym Raspberry jest wbudowany system z rodziny Linux Raspbian, w którym znajduje się oprogramowanie do sterowania robotem. Mikrokomputer jest zasilany 5 V z zewnętrznego niezależnego akumulatora (powerbank), ponieważ maszyna krocząca musi mieć swobodę ruchu i nie może być ograniczona okablowaniem. Kamera FullHD jest podłączona do Raspberry PI poprzez interfejs CSI. Połączenia elementów w układzie przedstawia rysunek 3.17 :



Rys. 3.17 Schemat połączeń elektrycznych maszyny kroczącej.
Źródło [34]

4. Konfiguracja jednopłytkowego komputera Raspberry PI 2

Z poziomu Raspberry PI 2 odbywa się sterowanie procesami zachodzącymi w układzie, dlatego wymagane są odpowiedni system oraz oprogramowanie. Obecnie powszechnie stosowanym nośnikiem danych dla Raspberry PI jest karta MicroSD, dlatego system musi zajmować małą ilość pamięci oraz być na tyle niezawodnym, aby reagować na zadania krytyczne np. niekontrolowane wyłączenie systemu. Dodatkowym wymaganiem do spełnienia, jest prosty w obsłudze interfejs graficzny tak, aby każdy użytkownik o różnym stopniu umiejętności obsługi komputera, mógł z niego skorzystać. Biorąc pod uwagę powyższe kryteria, wybrano dla Raspberry PI 2 system z rodziny Linux – Raspbian, który jest oparty na stabilnej dystrybucji Debiana w wersji Jessie. Wcześniej wymienioną wersję systemu można pobrać ze strony internetowej Raspberry PI Foundation [30].

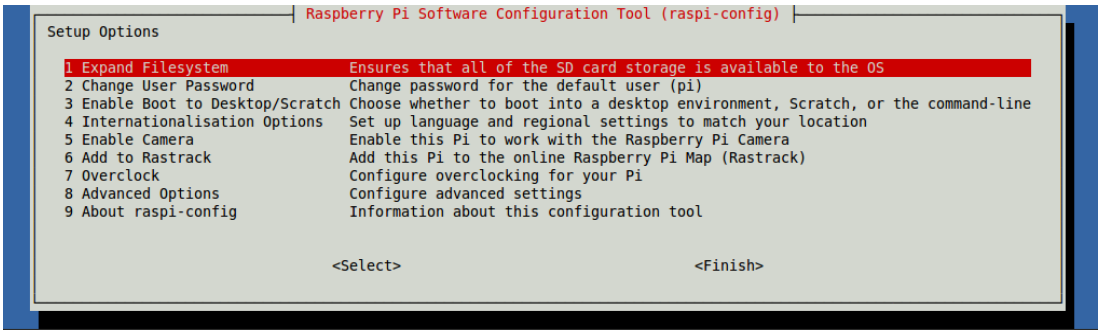
4.1. Instalacja i konfiguracja systemu Raspbian

Ze strony internetowej Raspberry Pi Foundation [30], z zakładki Downloads pobrano minimalny obraz systemu Raspbian. Następnie zapisano go na karcie karcie MicroSD, którą potem włożono do czytnika kart. Czytnik znajduje się pod spodem płytki mikrokomputera. Następnie do Raspberry PI 2 podłączono: kabel HDMI z wyświetlaczem/monitorem, klawiatura i mysz USB oraz adapter WiFi. Podłączenie urządzeń oraz zasilania do mikrokomputera przedstawiono na rysunku 3.18. Sterownik serw Mini Maestro dołączono w późniejszych etapach projektu po wstępnej konfiguracji systemu. Po zespoleniu niezbędnych urządzeń do pierwszej konfiguracji Raspbiana, mikrokomputer zasilono 5 V. Wyżej wymienione urządzenia podłączono w takiej kolejności, aby nie uszkodzić karty MicroSD. Po włączeniu zasilania, na płycie zapaliła się czerwona dioda PWR. Prawidłowe działanie systemu sygnalizuje mrugająca na zielono dioda ACT, a na ekranie pojawił się interfejs graficzny.

Przy pierwszym uruchomieniu użytkownik jest domyślnie logowany, jako „*pi*” z hasłem „*raspberry*”. W dalszych etapach projektu login i hasło nie zmieniono. W celu wykonania podstawowych konfiguracji systemu, w lewym górnym rogu interfejsu graficznego, na trzeciej pozycji od paska Menu, znajduje się ikona wiersza poleceń, w którym wpisano komendę:

```
sudo raspi -config
```

Sudo przed poleceniem *raspi-config* umożliwia modyfikację plików w systemie na prawach administratora. Użytkownik „*pi*” nie posiada takich uprawnień. Pojawiło się okno w konfiguracji systemu:



Rys. 4.1 Okno konfiguracji systemu Raspbian

W oknie konfiguracji systemu wybrano opcję 1 ”Expand FileSystem”, która przydzieliła jeszcze niewykorzystaną, wolną pamięć na potrzeby systemu. Po tej czynności system ponownie włączono, tak, aby nowa konfiguracja została zapamiętana. W tym celu wybrano opcję „Finish” i nastąpiło ponowne włączenie systemu. Do zmiany hasła i nazwy użytkownika można użyć opcji 2 „Change User Password”.

W górnym prawym rogu pulpitu znajdują się okienka do ustawienia: Wifi, zużycia zasobów obliczeniowych procesora oraz czas i data. W celu ustanowienia połączenia z internetem wybrano lokalną, bezprzewodową sieć laboratoryjną. Połączenie z nią zakończyło się sukcesem, co zostało sprawdzone przez wysłanie sygnału ping do serwera firmy Google oraz skorzystanie z wyszukiwarki tej samej firmy.

Po wyłączeniu mikrokomputera podłączono kamerę przez interfejs CSI jak na rysunku 3.17. W celu nawiązania komunikacji pomiędzy mikrokomputerem, a kamerą w oknie konfiguracji systemu (rys. 4.1) uruchomiono odbieranie sygnałów z kamery poprzez wybór opcji 5 „Enable Camera”. Następnie po wyjściu z okna konfiguracji systemu sprawdzono działanie aparatu. W wierszu poleceń wpisano następująca komendę:

```
raspistill -t 2000 -o image.jpg ,
```

która po dwóch sekundach zapisała zrobione zdjęcie w pliku o nazwie image z rozszerzeniem .jpg, który domyślnie jest utworzony w katalogu /home. Wszystkie zdjęcia i nagrania są wykonywane w najwyższej dostępnej rozdzielczości dla wybranego modelu kamery. Zmianę tego ustawienia można wprowadzić poprzez wpisanie na końcu komendy raspistill -w (ang. width – szerokość) liczba, -h (ang. height – wysokość) liczba. Przykład użycia komendy:

```
raspistill -t 2000 -o image.jpg -w 640 -h 480
```

Po otworzeniu pliku image.jpg w domyślnym programie do oglądania zdjęć w katalogu /home, obraz z kamery został zarejestrowany, co oznacza, że kamera jest w pełni funkcjonalna.

System Raspbian wykonał prawidłowo wszystkie polecenia użytkownika np. włączenie domyślnej przeglądarki internetowej Epiphany. Na tym etapie zakończono podstawową konfigurację systemu oraz kamery. Więcej informacji dotyczących dystrybucji systemów i ich instalacji, konfiguracji dodatkowych elementów, dokumentacji urządzeń oraz wsparcie ze strony innych użytkowników dostępne są na stronie internetowej Raspberry Pi Foundation [30].

4.2. Instalacja Codeblocks

Do stworzenia oprogramowania dla maszyny kroczącej wybrano zintegrowaną platformę programistyczną o otwartym kodzie źródłowym Codeblocks [39] dostępną na systemy Linux, Windows i Mac OS. Obsługuje ona wiele kompilatorów, a w szczególności dla języka C/C++. Dzięki możliwości doinstalowania dodatkowych wtyczek, środowisko to nie ogranicza się tylko do obsługi jednego języka. Platforma Codeblocks jest łatwym w użyciu środowiskiem, niewymagającym skomplikowanych nakładów obliczeniowych. Co więcej, samo środowisko po instalacji nie zajmuje dużej ilości pamięci. Instalacja Codeblocks jest łatwym procesem zarówno dla systemów Windows jak i dla systemów z rodziny Linux. Instalacja Codeblocks dla systemu Raspbian odbyła się w następujący sposób:

1. Przed pobraniem i instalacją Codeblocks zaktualizowano wszystkie pakiety oraz firmware. W wierszu poleceń terminala wpisano następujące komendy:

```
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update
```

2. Po skończonej aktualizacji pakietów i firmware, ponownie włączono system. W wierszu poleceń wpisano następujące polecenie:

```
sudo reboot
```

   Każda aktualizacja firmware Raspberry wymaga ponownego uruchomienia systemu.

3. Następnie w wierszu poleceń wydano rozkaz do pobrania i zainstalowania środowiska Codeblocks z repozytorium:

```
sudo apt-get install codeblocks
```

   Polecenie *apt-get install*, pobierze i zainstaluje zawsze najnowszą wersję oprogramowania.

Komenda nie wymaga ingerencji użytkownika przy ściągnięciu i instalacji programu.

4. Po rozpakowaniu i instalacji pakietu, środowisko Codeblocks jest już gotowe do użycia.

4.3. Instalacja OpenCV

Instalacja biblioteki OpenCV dla systemu z rodziny Linux jest dość skomplikowanym i długim procesem. W internecie można znaleźć dużo stron z przewodnikami pokazującymi niedoświadczonym użytkownikom jak przeprowadzić poprawną instalację biblioteki. Najlepszą z nich jednak okazała się strona internetowa Adriana Rosebrocka „pyimagesearch" [9]. Według niej instalacja OpenCV przebiega w następujący sposób:

1. Przed pobraniem biblioteki OpenCV zaktualizowano wszystkie pakiety oraz firmware. W wierszu poleceń wpisano następujące komendy:

```
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update
```

2. Po skończonej aktualizacji pakietów i firmware, ponowne włączono system. W wierszu poleceń wpisano komendę:

```
sudo reboot
```

3. Następnie doinstalowano paczki z narzędziami potrzebnymi do zbudowania modułów biblioteki za pomocą komendy Cmake:

```
sudo apt-get install build-essential git cmake pkg-config
```

4. W celu wykonania podstawowych działań na plikach graficznych, zainstalowano pakiety obrazów I/O za pomocą polecenia:

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev
            libpng12-dev
```

5. Instalacja pakietów wideo I/O przy użyciu poleceń:

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
            libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
```

6. Pobrano także bibliotekę GTK, aby wyświetlać zdjęcia/widea na monitorze/wyświetlaczu . Obsługuje ona moduł highgui:

```
sudo apt-get install libgtk2.0-dev
```

7. Do zoptymalizowania operacji na macierzach w OpenCV, wymagany jest pobranie następującej paczki danych:

```
sudo apt-get install libatlas-base-dev gfortran
```

8. Dodatkowe moduły umożliwiające korzystanie z metod biblioteki napisanych w języku Python w wersjach 2.7 i 3:

```
sudo apt-get install python2.7-dev python3-dev
```

W ten sposób zakończono etap konfiguracji środowiska potrzebnego do obsługi podstawowych modułów z biblioteki OpenCV.

9. Następnie ściągnięto moduły biblioteki OpenCV z repozytorium. W czasie pisania pracy magisterskiej wykorzystano najnowszą wersją biblioteki OpenCV 3.0.0. W wierszu poleceń:

```
cd ~
wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.0.0.zip
```

W przypadku innych wersji OpenCV, należy zamienić 3.0.0 na numer wersji, którą użytkownik zamierza pobrać. Rozpakowano pobraną paczkę za pomocą polecenia unzip:

```
unzip opencv.zip
```

Powyższe polecenie zakończyło etap pobrania i rozpakowania modułów biblioteki OpenCV.

10. Przed instalacją kolejnych modułów biblioteki włączono w konsoli wirtualne środowisko cv:

```
workon cv
```

11. Potem zmieniono katalog /home na katalog z rozpakowanymi plikami OpenCV:

```
cd ~/opencv-3.0.0/
```

12. Następnie utworzono folder „build”, potrzebny do zbudowania modułów::

```
mkdir build
cd build
```

13. W folderze *build* wydano następujące polecenia zbudowania modułów biblioteki:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_C_EXAMPLES=ON \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.0.0/modules \
-D BUILD_EXAMPLES=ON ..
```

14. Po prawidłowo zakończonym procesie, skompilowano istniejące w folderze pliki:

```
make -j4
```

-j4 oznacza, że do kompilacji plików używane są wszystkie cztery rdzenie mikrokomputera. W przypadku pojawienia się błędu podczas kompilacji plików, należy powtórzyć operację tylko dla jednego rdzenia:

```
make clean
make
```

15. Następnie zainstalowano bibliotekę OpenCV za pomocą następujących komend:

```
sudo make install
sudo ldconfig
```

Polecenie ldconfig tworzy połączenie i miejsce w pamięci podręcznej dla najnowszych bibliotek OpenCV zainstalowanych w systemie.

4.3.1. Weryfikacja działania biblioteki OpenCV

Po prawidłowej instalacji modułów biblioteki OpenCV, przeprowadzono weryfikację ich działania:

1. W katalogu /home utworzono nowy dokument tekstowy z rozszerzeniem.cpp o nazwie test:

```
test.cpp
```

2. Ze strony internetowej OpenCV 3.0.0 [55] skopiowano przykładowy program do pliku test.cpp, który otwiera wybrane przez użytkownika zdjęcie.
3. Ze strony internetowej wyszukiwarki Google pobrano przykładowe zdjęcie lena.jpg do katalogu /home.
4. Następnie w wierszu poleceń uruchomiono pliku test.cpp za pomocą poniższych komend:

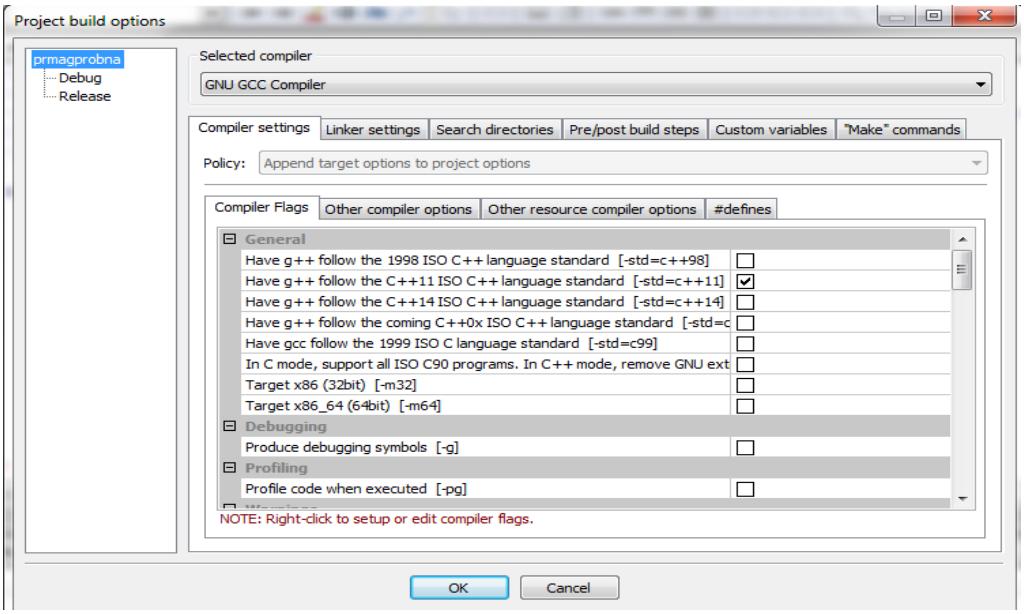
```
gcc test.cpp -o test
./test
```

5. W przypadku prawidłowej instalacji biblioteki OpenCV, na ekranie monitora/wyświetlacza pojawiło się zdjęcie lena.jpg w nowym oknie.

4.4. Konfiguracja Codeblocks, OpenCV i kamery

Jednym z trudniejszych etapów konfiguracji narzędzi potrzebnych do napisania oprogramowania do rozpoznawania piłki jest zintegrowanie modułów biblioteki OpenCV i Codeblocks. Do połączenia tych dwóch środowisk wykonano następujące czynności:

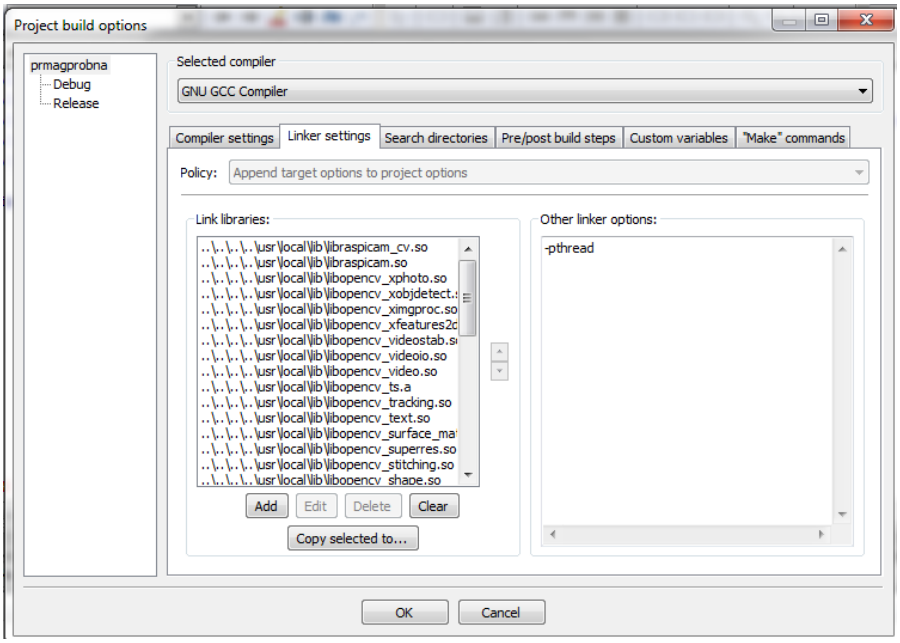
1. W Codeblocks utworzono nowy projekt. W zakładce File→New project→Console Application. W dalszych krokach dobrano parametry projektu według preferencji użytkownika.
2. Następnie w oknie z przestrzenią roboczą projektu wybrano za pomocą prawego przycisku myszy, opcję „Build options". Pojawiło się okno z ustawieniami projektu – rysunek 4.2.



Rys. 4.2 Okno z ustawieniami projektu w programie Codeblocks

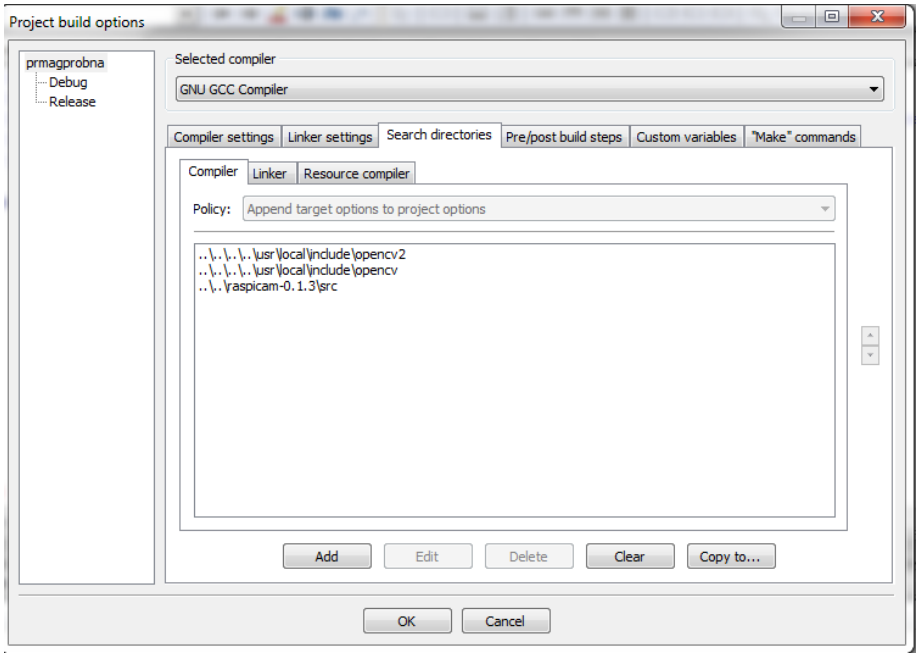
Wszystkie zmiany wykonano w ustawieniach ogólnych projektu tak, jak na rysunku 4.2 Ustawiono wersję kompilatora na C++ 11 z powodu użycia wątków (ang. threads) w projekcie.

3. Następnie w zakładce „Linker Settings" w oknie „Link Libraries" – rysunek 4.3 dodano pliki biblioteki OpenCV. Dodatkowo, w oknie „Other linker options" wpisano flagę do obsługi wątków, bez której kompilator przy sprawdzaniu programu będzie traktował wątki, jako błędy.



Rys. 4.3 Okno z ustawieniami projektu. Zakładka Linker Settings

4. Następnie w zakładce „Search directories” (rys. 4.4) dodano ścieżki do plików bibliotek OpenCV oraz „raspicam” odpowiedzialnej za obsługę sygnału z kamery FullHD. Potem zapisano zmiany w projekcie.



Rys. 4.4 Okno z ustawieniami projektu. Zakładka „Search Directories”

5. Do sprawdzenia poprawności zintegrowania środowisk Codeblocks z OpenCV użyto kodu z pliku testowego test.cpp. Po skompilowaniu i uruchomieniu programu z przykładowym kodem, pojawiło się okno ze zdjęciem lena.jpg.

4.5. Instalacja zdalnego klienta

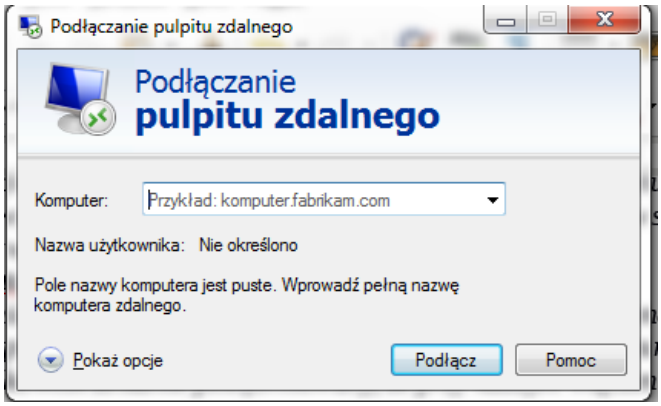
Aby maszyna krocząca była jak najbardziej mobilna, ograniczono liczbę urządzeń do niej podłączonych. Założono, ze jedynymi urządzeniami na robocie są: Raspberry Pi zasilone z małego akumulatora oraz sterownik serw Mini Maestro, do którego podłączone są serwa z zasilania zewnętrznego. W celu nadzorowania czynności wykonywanych przez maszynę, na mikrokomputerze zainstalowano serwer xdrp, który umożliwia połączenie pomiędzy dowolnym komputerem, a Raspberry PI poprzez sieci WiFi. Komunikacja pomiędzy serwerem, a klientem odbywa się z użyciem protokołu RDP (ang. remote desktop connection). Protokół pozwala na połączenie pulpitu zdalnego systemu Windows z pulpitem systemu Linux. Opcja zdalnego pulpitu jest dostępna w systemie Windows od wersji 7.

Xrdp zainstalowano w systemie Raspbian wwpisując w wierszu poleceń komendę:

```
sudo apt-get install xrdp
```

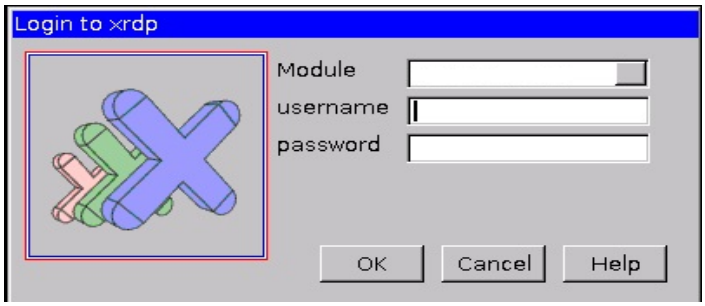
Procesy: pobierania, instalacji oraz konfiguracji wykonywane są automatycznie. Po ukończeniu powyższych czynności, w oknie wiersza poleceń pojawił się komunikat o automatycznym uruchomieniu się programu przy każdorazowym starcie systemu.

Następnie, aby nawiązać połączenie z maszyną kroczącą, uruchomiono program „podłączenie pulpitu zdalnego” z paska menu (w przypadku Windows 7) na komputerze. Na ekranie pojawiło się okno „podłączenie pulpitu zdalnego” – rysunek 4.5.



Rys. 4.5 Okno programu „podłączenie pulpitu zdalnego”.

W rozwijalnym pasku „Komputer” wpisano adres IP mikrokomputera. Pominięto dodatkowe opcje ustawienia, które są dostępne z lewej strony na dole okna.



Rys.4.6 Okno serwera xrdp. Źródło [23]

Po bezbłędnym nawiązaniu połączenia pomiędzy dwoma pulpitami, pojawiło się okno logowania do serwera – rysunek 4.6. W pasku rozwijalnym przy „Module” wybrano nazwę docelowego modułu. W pasku „username” i „password” wpisano odpowiednio nazw użytkownika „*pi*” z hasłem „*raspberry*”.

4.6. Repozytorium Git

Do rozwoju i przechowywania oprogramowania dla maszyny kroczącej, oprócz standardowego zapisu projektu w katalogu /home, dodatkowo wybrano internetowe repozytorium – Github, które jest bezpłatne i ogólnodostępne. Pozwala ono na podgląd zmian w wersjach oprogramowania, a także na powrót do poprzednich wersji. Co więcej, umożliwia pracę zespołową nad oprogramowaniem poprzez współdzielenie repozytorium. Ponadto pozwala ono na szybkie i łatwe ściągnięcie oprogramowania w skompresowanym pliku.

Na stronie internetowej Github [20] założono konto dla maszyny kroczącej razem z kontem pocztowym.

Login do konta robota w serwisie Github:

```
robot_roman@wp.pl
```

Hasło:

```
ROMAN_pw_2016
```

Po zalogowaniu się do konta, w zakładce Repositories →New, utworzono publiczny projekt o nazwie *code_for_robot*. Następnie, posługując się instrukcjami ze strony internetowej Github, w wierszu poleceń użyto następujących komend:

```
git init
git add .
```

Pierwsze polecenie utworzyło nowe, puste repozytorium. Drugie polecenie dodało wszystkie pliki z folderu do nowegorepozytorium.

Komendą *git add* można dodać także pojedynczy plik. Zamiast kropki należy podać wtedy nazwę pliku. Do przesłania plików na serwer Github służy poniższa komenda:

```
git commit -m „first commit”
```

Każdy z plików otrzymał podany przez użytkownika komentarz, aby ułatwić śledzenie wszystkch zmian dokonywanych w plikach. Poniższe polecenia dodają na główną gałąź repozytorium pliki projektu:

```
git remote add origin https://github.com/robotroman/code_for_robot .git
git push -u origin master
```

Wraz z zakończeniem dodawania plików do głównej gałęzi, projekt stał się ogólnodostępny. W razie wystąpienia błędu na którymkolwiek z powyżej opisanych etapów, do sprawdzenia przyczyny zalecane jest użycie komendy:

```
git status
```

Są dwie możliwości pobrania plików projektu. Pierwsza umożliwia pobranie ze strony internetowej projektu w formacie tar lub zip. W ramach drugiej, w konsoli wpisuje się komendę klonującą repozytorium:

```
git clone git://github.com/robotroman/code_for_robot.git
```

5. Detekcja piłki za pomocą OpenCV i kamery

W niniejszym rozdziale opisano wybrane zagadnienia związane z percepcją maszyny kroczącej. Został on podzielony na część teoretyczną i praktyczną. W pierwszych trzech podrozdziałach umieszczone są podstawowe zagadnienia związane z przetwarzaniem obrazu. Opisane są modele przestrzeni barw, obraz 3D oraz biblioteka OpenCV. W kolejnych znajduje się opis algorytmu detekcji piłki, który zaimplementowany jest w klasie *camera*. Algorytm jest podzielony na metody odpowiedzialne za: konwersję obrazu z RGB do HSV, filtrację obrazu, rozpoznawanie okrągłych obiektów i obliczenie odległości pomiędzy piłką a maszyną kroczącą.

Zadaniem oprogramowania dla maszyny kroczącej jest wykrycie małego, okrągłego obiektu o zadanym kolorze oraz wyliczenie w przybliżeniu poprawnego dystansu do niego i następnie podejście do niego i kopnięcie go. Do obsługi sygnału z kamery oraz lokalizacji obiektu, zostały wykorzystane niektóre funkcje z modułów biblioteki OpenCV. Najpierw wykonane są zdjęcia, na których zastosowany jest filtr odrzucający wszystkie niekoliste obiekty. Do znalezionego okręgu dobierane są odpowiednie poziomy saturacji i wartości koloru czerwonego, tak, aby można było jak najlepiej określić kształt piłki. W przypadku, gdy manipulacja poziomami się nie powiodła to szukany jest kolejny obiekt spełniający powyższe kryteria. Następnie na podstawie uzyskanego kształtu okręgu wyliczane są odległości pomiędzy maszyną kroczącą a piłką oraz środkiem obiektywu kamery, a piłką. Dodatkowo określana jest strona maszyny, po której znajduje się szukany obiekt. Uzyskane wyliczenia są przekazywane do metody *go_to_the_ball* w klasie *robot*, w której przeliczane są na liczbę iteracji, jaką każda noga musi wykonać, aby robot mógł podejść do piłki.

5.1. Przetwarzanie obrazu

Obraz cyfrowy to dwu- lub trójwymiarowy sygnał, który zapisany jest w formie dwu- lub trójwymiarowej macierzy pikseli. Powstaje z zamiany wartości analogowych zarejestrowanego obrazu na wartości dyskretne. Sposób ułożenia pikseli w rzędach i kolumnach tworzy mapę bitową obrazu (ang. bitmap). Z obrazem cyfrowym łączy się pojęcie rozdzielczości, które określa stosunek liczby pikseli do danej jednostki powierzchni. Najczęściej wyróżnia się cztery rodzaje obrazu cyfrowego:

- czarno-biały – to dwuwymiarowa macierz o rozmiarze M x N, w której piksele przyjmują wartość 1 – czarny lub 0 – biały;
- w odcieniach szarości (ang. grayscale) – to dwuwymiarowa macierz o rozmiarze M x N, w której jeden piksel zakodowany jest na 8 bitach, dzięki czemu może przyjmować 256 różnych odcieni szarości;
- kolorowy – to tablica trójwymiarowa o rozmiarze M x N x 3, która jest połączeniem podstawowych kolorów: czerwonego, zielonego i niebieskiego. Dzięki temu, że obraz jest zakodowany na 24 bitach, każdy z wyżej wymienionych kolorów może przyjąć 256 różnych odcieni. Maksymalnie obraz może mieć 2^{24} (16 777 216) barw;
- indeksowany – to zapis dwóch macierzy, z których pierwsza zawiera paletę kolorów, a w drugiej macierzy piksele są indeksami do barw zawartych w pierwszej. Maksymalnie obraz może mieć 256 kolorów.

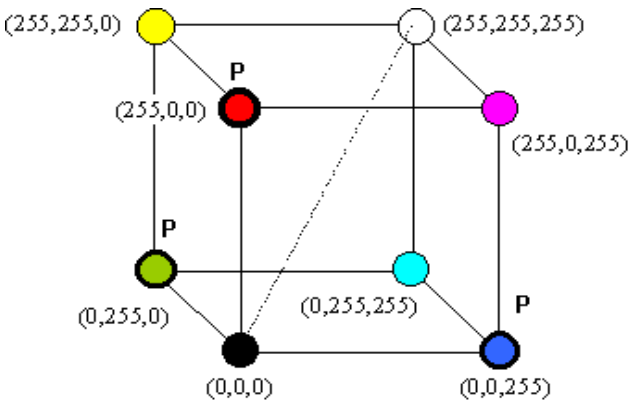
Wszystkie operacje wykonywane na obrazie określa się terminem przetwarzania lub obróbki obrazu i przebiegają one zazwyczaj w następujący sposób:

- akwizycja obrazu – to pobranie danych z kamery oraz zamiana ich na postać cyfrową czytelną dla komputera;
- przetwarzanie wstępne obrazu – to zastosowanie odpowiednich operacji w celu poprawy jakości obrazu dla dalszych operacji na nim. Do tych działań zalicza się: udoskonalanie obrazu, filtrację i przekształcenia: geometryczne, punktowe, nieliniowe, liniowe, widmowe, morfologiczne;
- segmentacja obrazu – to podział obrazu na wiele segmentów w celu ułatwienia jego dalszej analizy. Każdemu pikselowi przypisana jest etykieta tak, aby każdy z nich miał podobne cechy. Wynikiem tej operacji jest podział obrazu na homogeniczne regiony pod względem wybranych właściwości np. poziom barwy szarej. Segmentacji można dokonać metodami: klastrowymi na bazie histogramu, wykrywania krawędzi, a nawet za pomocą sieci neuronowych etc;

- opis obrazu – to wyodrębnienie pewnych cech lub funkcji, które pozwalają na odróżnienie klas obiektów;
- rozpoznawanie obiektów – to podział obiektów na podstawie ich funkcji lub cech opisanych przez ich deskryptory;
- interpretacja obrazu – to przypisanie znaczenia do grup sklasyfikowanych obiektów.

5.1.1. Modele przestrzeni barw

Do prezentacji obrazu wykorzystuje się najczęściej model przestrzeni barw RGB (ang. red – czerwony, green – zielony, blue – niebieski). Mieszając ze sobą te trzy podstawowe kolory uzyskuje się barwy pochodne jak np. żółty. Połączenie odbywa się na zasadzie syntezy addytywnej, czyli poprzez sumowanie różnych długości światła widzialnego. Zerowa intensywność wskazuje na kolor czarny, natomiast największa na kolor biały. Każda z barw RGB jest zapisana za pomocą 24 bitów. Tworzona jest przez trzy składowe, które mogą przyjmować wartości z przedziału 0–255. Kolor czarny ma wartość wszystkich składowych równą zero, a biały – 255. Barwy RGB są przedstawione na poniższym modelu (rys. 5.1).



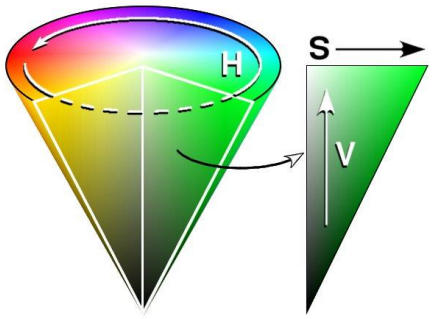
Rys. 5.1 Model RGB. Źródło [42]

Każdy z kolorów RGB można wyliczyć z następującego wzoru:

Kolor = R (kolor czerwony) *65536 + G (kolor zielony) *256 + B (kolor niebieski)

Obecnie model RGB stosuje się w urządzeniach, które rejestrują oraz wyświetlają obraz np. monitory, skanery, TV, aparaty cyfrowe etc.

Kolejnym powszechnie używanym modelem barw jest model HSV (ang. hue – barwa, saturation – nasycenie, value – wartość). W tym modelu wszystkie barwy generowane są z światła białego, którego część widma zostaje wchłonięta, a część jest odbita od przedmiotów otaczających. Model jest najczęściej prezentowany za pomocą stożka (rys. 5.2).



Rys. 5.2 Model HSV w postaci stożka. Źródło [50]

Nasycenie koloru tworzy promień podstawy figury. Przyjmuje on wartość z przedziału od 0 do 1. Barwa jest wyrażona poprzez kąt obrotu wokół wysokości stożka w zakresie od 0^0 do 360^0. Kolor czerwony występuje dla kątów 0^0 i 360^0, zielony dla 120^0, a niebieski dla 240^0. Wysokość figury określona jest przez wartość i jest w zakresie [0,1]. Od środka podstawy stożka, gdzie znajduje się kolor biały, kolor ciemnieje aż do wierzchołka, gdzie występuje kolor czarny. Model HSV można przekształcić do modelu RGB za pomocą niżej podanych wzorów:

$$C = VxS \tag{16}$$

$$H' = \frac{H}{60^0} \tag{17}$$

$$X = Cx\,(1 - |H' mod 2 - 1|) \tag{18}$$

$$m = V - C \tag{19}$$

$$(R', G', B') = \begin{cases} (C, X, 0) & ,0° \le H < 60° \\ (X, C, 0) & ,60° \le H < 120° \\ (0, C, X) & ,120° \le H < 180° \\ (0, X, C) & ,180° \le H < 240° \\ (X, 0, C) & ,240° \le H < 300° \\ (C, 0, X) & ,300° \le H < 360° \end{cases} \tag{20}$$

$$(R, G, B) = \big((R' + m)\ x\ 255, (G' + m)\ x\ 255, (B' + m)\ x\ 255\big)\,, \tag{21}$$

gdzie:

C – Chrominancja.

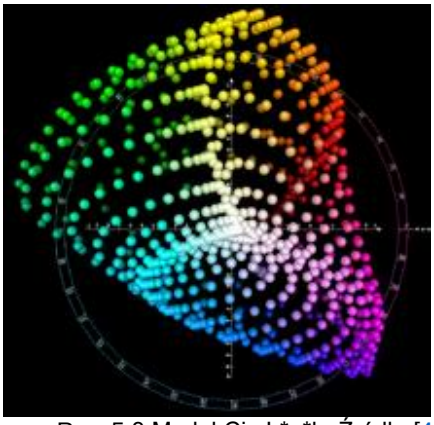
S – Saturacja.

H – Kolor.

V – Wartość.

Zapis barw przy pomocy modelu HSV jest podobny do percepcji kolorów człowieka, dlatego znalazł on zastosowanie w grafice komputerowej, detekcji obiektów etc. Umożliwia on także obliczenie stosunku liczb nasyconych pikseli do nienasyconych oraz jest on odporny na zmiany oświetlenia. Niestety pomimo jego dużych zalet, używając tego modelu można zniekształcić relacje kolorów na obrazie, zwiększając lub zmniejszając ich jasność.

Następnym modelem przestrzeni barw jest trójwymiarowy model CIE L*a*b*, który jest modyfikacją przestrzeni L, a, b. Został wprowadzony z powodu zaawansowanych badań nad rozpoznawaniem kolorów przez ludzkie oko. Przestrzeń CIE L*a*b* (rys. 5.3) tworzona jest przez trójwymiarową przestrzeń liczb rzeczywistych, które są odwzorowane na przestrzeń liczb całkowitych celem zapisu cyfrowego niezależnego od urządzeń elektronicznych. Opis koloru tworzą trzy składowe: L* – jasność (ang. lightness), a* – barwa od zielonej do magenty/jasnopurpurowej, b* – barwa od niebieskiej do żółtej. Składowe a* oraz b* mogą przyjmować wartości dodanie i ujemne. Naturalny szary kolor występuje dla wartości a* oraz b* równych zero. Dodatnia składowa a* wskazuje na kolor magenta natomiast ujemna na zielony. Ujemna składowa b* wskazuje na kolor niebieski zaś dodatnia na żółty. Wartość jasności równa zeru wskazuje na kolor czarny, a równa 100 na kolor biały.



Rys. 5.3 Model Cie L*a*b. Źródło [45]

Model ten został zaprojektowany tak, aby w jak największym stopniu odzwierciedlić to, co widzi człowiek. Używany jest w programach graficznych np. Adobe Photoshop do zachowania koloru w poligrafii etc.

5.2. OpenCV

Powszechnie stosowanymi narzędziami do wykonywania operacji na obrazie są metody z stworzonej przez firmę Intel biblioteki OpenCV o otwartym kodzie źródłowym. Dodatkowo, biblioteka ma także narzędzia do uczenia maszynowego. Początkowo zaimplementowane funkcje były tylko w języku C/C++, ale z czasem i wraz z rosnącym zapotrzebowaniem na przetwarzanie obrazów, pojawiło się wsparcie także dla takich języków jak Python, JAVA etc. Aktualnie funkcje z OpenCV można stosować na wszystkich dostępnych systemach operacyjnych. Funkcje i algorytmy zostały podzielone według ich zastosowania na tematyczne moduły tak, aby ułatwić ich używanie oraz zmniejszyć rozmiar pamięci pisanego programu. Oto niektóre moduły dostępne w OpenCV 3.0.0 [8], które zostały wykorzystane w pracy:

- core – moduł z podstawowymi funkcjami i strukturami do obróbki obrazu. Jest podstawowym modułem biblioteki OpenCV;
- highgui – prosta obsługa interfejsu użytkownika;
- imgproc – do przetwarzania obrazu. Zawiera liniowe i nieliniowe filtry, konwersje kolorów, histogramy etc;
- video – algorytmy do obróbki wideo takie jak śledzenie obiektów, odczytywanie filmu etc;
- calib3d – algorytmy do skalibrowania kamery, rekonstrukcji 3D etc;
- videoio – prosty interfejs do obsługi widea i kodeków.

Dzięki temu, że biblioteka ma licencje open source BSD i jest prosta w obsłudze oraz na bieżąco jest aktualizowana, może być stosowana zarówno w celach amatorskich, akademickich, komercyjnych jak i militarnych.

5.2.1. Filtracja

Jedną z najczęściej używanych operacji w przetwarzaniu obrazu jest filtracja. Jest to operacja matematyczna na pikselach, której wynikiem jest nowy przekształcony obraz. Może być wykorzystana do uzyskania nowych informacji z obrazu np. określenia położenia krawędzi obiektu, usunięcia szumów (np. filtr medianowy) oraz rozmycia obrazu (np. filtr blur) etc. Operacja filtracji to obliczenie wartości piksela przy uwzględnieniu wartości jego sąsiadów. Uzyskany wynik jest zapisywany w takiej samej lokalizacji w obrazie docelowym, jaką ma piksel w obrazie źródłowym, co powoduje, że oba obrazy mają taki sam rozmiar. Filtracja dzieli się na metody liniowe i nieliniowe oraz przestrzenne i częstotliwościowe. W liniowej dla każdego piksela jest wyliczana suma ważona sąsiadujących pikseli. W filtracji morfologicznej, która należy do nieliniowych, dla każdego piksela jest obliczana największa lub najmniejsza wartość pikseli sąsiadujących. Najczęściej używanymi filtrami w bibliotece OpenCV są: dylatacja (ang. dilate), erozja (ang. erode) oraz MedianBlur.

Dylatacja [54] splata obraz (A) z elementem strukturalnym (B) o dowolnym kształcie i rozmiarze. Element jest zazwyczaj okręgiem lub kwadratem i posiada jeden zdefiniowany punkt kontrolny zwany środkiem. Obiekt poddany dylatacji zwiększa swoją objętość, zamykają się w nim dziury oraz zanikają szczegóły. Operacja dylatacji jest równoważna z obliczeniem lokalnego maximum dla elementu strukturalnego.

$$dilate(x,y) = \max_{(x\prime,y\prime)\in kernel} image(x+x',y+y') \tag{22}$$

Erozja [54] jest operacją odwrotną do dylatacji. Oblicza ona lokalne minimum dla elementu strukturalnegooraz zmniejsza obiekt. Podczas, gdy dylatacja wygładza obiekty wklęsłe, erozja wygładza wypukłe. Erozja likwiduje szumy i rozszerza dziury w obiekcie.

$$erode(x,y) = \min_{(x\prime,y\prime)\in kernel} image(x+x',y+y') \tag{23}$$

Funkcja MedianBlur [54] do rozmycia obrazu wykorzystuje nieliniowy filtr medianowy. Operacja ta polega na zastąpieniu wartości piksela przez medianę jego sąsiadów w obrębie okna o ustalonym rozmiarze. Filtr stosowany jest do usuwania szumów typu „sól i pieprz”. Główną jego zaletą jest odrzucenie pikseli, których wartości są skrajne i odstają od pozostałych w zbiorze. Wadą filtru jest duża złożoność obliczeniowa spowodowana koniecznością sortowania każdego zbioru pikseli w oknie.

5.3. Detekcja i lokalizacja piłki

Jednym z zadań do zrealizowania była detekcja obiektu o ustalonym rozmiarze i kolorze przy użyciu kamery oraz metod z biblioteki OpenCV. Założono, że poszukiwana jest czerwona piłka o średnicy 51 mm. Na początku, skonfigurowano komunikację pomiędzy kamerą, a Raspberry PI przy użyciu fragmentów kodu z oprogramowania do obsługi kamery raspicam, które stworzyła grupa AVA z Uniwersytetu Cordoby. Program pobrano bezpłatnie z strony internetowej projektu z repozytorium GitHub [21]. Aby skorzystać z ściągniętego sterownika, w oprogramowaniu należało umieścić prawa autorskie grupy. W projekcie użyto fragmentów odpowiedzialnych za kalibrację kamery oraz nawiązanie z nią połączenia. Zaimplementowano je w metodach: *processCommandLine*, *camera_film* oraz w konstruktorze klasy *camera*. W konstruktorze sprawdzane jest czy zostało nawiązane połączenie z kamerą oraz wyświetlany jest jej numer. Dodatkowo, wywoływana jest funkcja *processCommandLine* do ustawienia parametrów kamery takich jak rozdzielczość wyświetlanego obrazu, poziom nasycenia etc:

```
void camera::processCommandLine ( raspicam::RaspiCam_Cv &Camera )
{
    Camera.set ( CV_CAP_PROP_FRAME_WIDTH,  800 );
    Camera.set ( CV_CAP_PROP_FRAME_HEIGHT, 600 );
    Camera.set ( CV_CAP_PROP_BRIGHTNESS,50 );
    Camera.set ( CV_CAP_PROP_CONTRAST ,50 );
    Camera.set ( CV_CAP_PROP_SATURATION, 50 );
}
```

Detekcja piłki polega na wykryciu jednolitego czerwonego obszaru w zmiennie i niejednorodnie oświetlonym otoczeniu. Do uzyskania jak najlepszych wyników, zamieniono przestrzeń RGB, w której barwy zależą w dużym stopniu od jasności, na przestrzeń HSV, która jest bardziej odporna na zmiany oświetlenia. Dodatkowo, aby dokładnie określić kształt szukanego obiektu, zdjęcie oczyszczono z zakłóceń. Do tego celu wybrano filtr medianowy (ang. medianBlur filter). Filtrację obrazu zaimplementowano w metodzie *convert_pict_to_color_hsv*, której parametrami wejściowymi są: wektor przechowujący pobrane obrazy oraz zmienne określające jego początek i koniec. Efekty konwersji zapisano w wektorach globalnych *pictures1* i *pictures2*. Metoda *convert_pict_to_color_hsv* w oprogramowaniu:

```
int camera::convert_pict_to_color_hsv(vector<Mat> img, int b, int e)
{
    Mat img2,img3;
    for(int i=b;i<=e;i++)
      {
            cvtColor(img[i],img2,COLOR_RGB2BGR);
            pictures1.push_back(img2);
            cvtColor(img2,img3,COLOR_BGR2HSV);
            medianBlur(img3,img3,9);
            pictures2.push_back(img3);
      }
}
```

Rozpoznawane były kontury obiektu wraz z jego odbiciem (rys. 5.4) od podłogi oraz cieniem (rys. 5.5). Okazało się to dość znaczącym problemem, ponieważ wpływało to niekorzystnie na obliczenia lokalizacji piłki. Źle określony kształt obiektu powodował nieprawidłowe obliczenia odległości pomiędzy nim a robotem, przez co robot nie podchodził lub omijał piłkę. W celu wyeliminowania efektu cienia i refleksu, zastosowano dodatkową binaryzację poziomów wartości i nasycenia koloru.

Rys.5.4 piłka i jej odbicie od podłoża

Rys. 5.5 Piłka i jej cień na podłożu

W OpenCV przeskalowano barwę, nasycenie i wartość z zakresu 0-360^0 na 0-255, tak, aby można je było zapisać na 1 bajcie. Po przeskalowaniu, kolor czerwony znajduje się w dwóch zakresach: 170–255 i 0–20. Dopasowanie barwy czerwonej wraz z jej nasyceniem i wartością odbywa się w metodzie *red_ball*, która jako parametry wejściowe przyjmuje obraz oryginalny i HSV. Sama operacja ustawienia barwy, nasycenia, wartości wykonywana jest przez metodę *inRange* z biblioteki OpenCV. Metoda sprawdza czy elementy tablicy wejściowej podanej, jako pierwszy parametr znajdują się w zakresie określonym przez parametry podane na drugiej i trzeciej pozycji. Uzyskane wyniki zapisywane są do tablicy wyjściowej podanej, jako czwarty parametr.

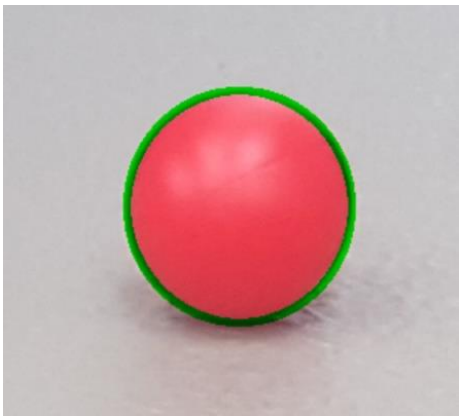
```
void camera::red_ball (Mat image_after_conversion,Mat image_org)
{
  Mat red,red2;
  if(minHuered<maxHuered)
inRange(image_after_conversion,Scalar(minHuered,minSatured,minValuered),
      Scalar(maxHuered,maxSatured,maxValuered),red);
else
{
inRange(image_after_conversion,Scalar(minHuered,minSatured,minValuered),
      Scalar(180,maxSatured,maxValuered),red);
inRange(image_after_conversion,Scalar(0,minSatured,minValuered),
      Scalar(maxHuered,maxSatured,maxValuered),red2);
red=red+red2;
}
draw_cont(red,image_org);
if (contours.size()!=0)
{
drawContours(image_org, contours, -1, CV_RGB(0,0,255),1);
}}
```

Z powodu występowania w dwóch zakresach barwy czerwonej, metodę *inRange* wywołano dwukrotnie na dwóch oddzielnie do tego stworzonych obrazach *red* i *red2*. Po przypisaniu odpowiednich wartości, zostają one scalone w jeden obraz. W celu przyśpieszenia procesu nastawy poziomów, ustawiono globalnie w klasie *camera* początkowe maksymalne i minimalne nasycenie, wartość oraz barwę:

```
int minHuered=170;
int maxHuered=10;
int minSatured=70;
int maxSatured=255;
int minValuered=100;
int maxValuered=255;
```

Wybrano takie wartości początkowe, ponieważ dla nich w warunkach umiarkowanego oświetlenia dopasowana elipsa dobrze odzwierciedla kształt piłki.

Następnie wywoływana jest metoda *draw_cont*, która rozpoznaje i obrysowuje kształt obiektu oraz oblicza jego odległość od robota. Kontur znalezionej piłki zostaje zaznaczony zielonym kolorem (rys. 5.6).



Rys.5.6 Kontur znalezionej piłki zaznaczony zielonym kolorem

Parametrami wejściowymi metody *draw_cont* są obrazy HSV i oryginalny. Znalezienie konturów przedmiotu wykonywane jest przez metodę *findContours* z biblioteki OpenCV. Metoda ta znajduje dowolne kształty na obrazie zbinaryzowanym, które następnie zapisuje w wektorze wyjściowym *contours*, z którego korzystają także pozostałe metody w klasach *robot* i *camera*. Następnie dla znalezionych obiektów liczone są obwody i pola, które są odpowiednio zapisane w zmiennych *perimeter* i *area*:

```
int camera::draw_cont(Mat& image_after_conversion, Mat& image_org)
{
 contours.clear();
 findContours(image_after_conversion, contours, RETR_EXTERNAL,
      CV_CHAIN_APPROX_SIMPLE);
       for (int i = 0; i < contours.size(); i++)
          {
            double area = contourArea(contours[i]);
            double perimeter = arcLength(contours[i], true);
```

Piłka na obrazie przedstawiona jest zawsze jako koło, dlatego do jej rozpoznania skorzystano z właściwości geometrycznych koła. Dla każdego wyszukanego konturu, obliczono jego kwadrat obwodu oraz pole, a następnie porównano ich stosunek, który jest liczbą stałą i wynosi 4π :

$$\frac{(2\pi R)^2}{\pi R^2} = 4\pi$$

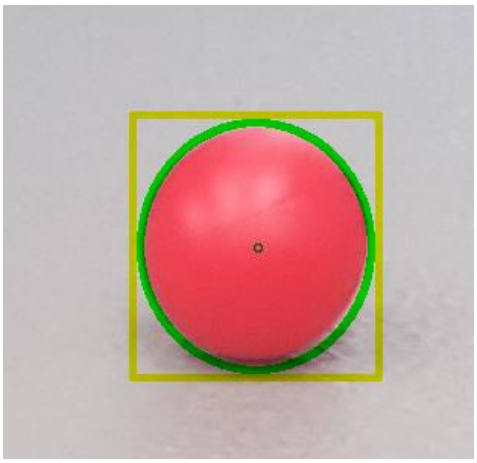
Z tej tożsamości wynika, że obwód figury ogranicza jej największą powierzchnię. Co więcej, aby obiekt był rozpoznawany z małych (20 cm) jak i dużych odległości (do kilku m) ustanowiono przedział, w którym ma się zawierać wyliczona wartość pola koła oraz wyżej pokazany stosunek. Najdłuższy dystans, z jakiego robot może rozpoznać piłkę wynosi około 3 metry. Dla dużej odległości ustalono, że pole figury nie może być mniejsze niż 250 pikseli, a dla małej nie może być większe niż 380000. Zwiększono stosunek kwadratu obwodu do pola koła, ponieważ dla mniejszych wartości nie był wykrywany cały obiekt, tylko jego cześć. W przypadku, gdy znaleziony przedmiot nie spełnia któregoś z warunków zostaje odrzucony.

```
if (area < 250 || area > 380000  || perimeter*perimeter / area>40)
{
      contours.erase(contours.begin() + i, contours.begin() + i + 1);
                  i--;
}}
```

W przypadku nie znalezienia odpowiednich konturów, nie są wykonywane obliczenia położenia piłki. Ustawiana jest wtedy zmienna globalna *probe_indicator_for_camera,* która informuje odpowiednią metodę – *walk_sidewalk* z klasy *robot*, że robot ma wykonać obrót w lewo celem znalezienia konturów piłki.

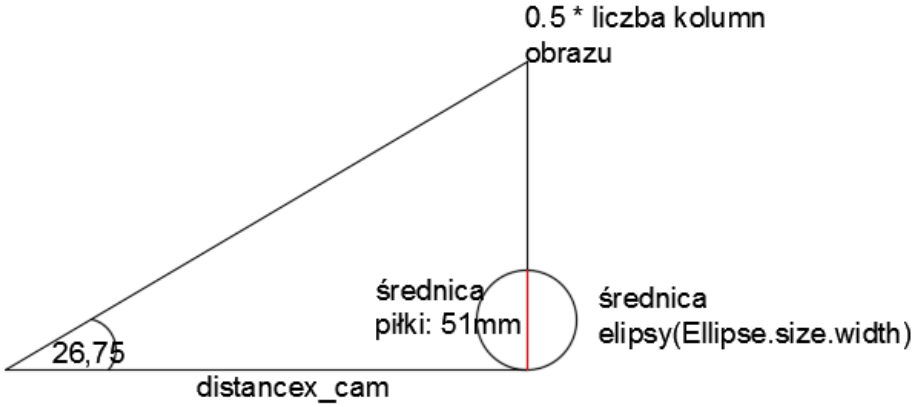
W sytuacji wykrycia szukanego obiektu dopasowywana jest do niego elipsa, która najlepiej z możliwych figur odzwierciedla piłkę na obrazie. W celu dopasowania jej optymalnego kształtu do obiektu, dobierane są jej długi i krótki promień poprzez manipulację poziomami wartości i saturacji koloru czerwonego. Przyjęto takie nastawy początkowe saturacji i wartości, że stosunek promieni elipsy wynosi 0.92, ponieważ wtedy dla zmiennych warunków oświetlenia kształtem przypomina ona piłkę. Nie jest dokładnie odwzorowywany obrys szukanego obiektu, dlatego zmieniane są promienie elipsy. Sam algorytm opiera się na stopniowym zwiększaniu najpierw nasycenia, a potem wartości. Przy każdorazowym zwiększaniu jednej z wcześniej wymienionych wielkości, ustawiany jest kolor czerwony wraz z nasyceniem i wartością za pomocą metody *inRange*. Wtedy liczony jest na bieżąco stosunek promieni elipsy, który jest pośrednio regulowany w metodzie *add_track* i porównywany jest z poprzednim wyliczonym stosunkiem w celu wyłonienia jak najlepszego rozwiązania. Im bliżej jedności stosunek tych promieni, tym bardziej elipsa odzwierciedla obiekt na obrazie. Dłuższy promień elipsy przechowywany jest w zmiennej *b*, a krótszy w zmiennej *a*. W OpenCV najpierw wyznaczony i narysowany jest obrócony prostokąt poprzez metodę *minAreaRect*, w którym zostaje wpisana elipsa poprzez metodę *fitEllipse* (rys. 5.7):

```
   for( int i = 0; i< contours.size(); i++ )
    {
      drawContours( image_org, contours, i, Scalar(0,0,255), 1, 8,     vec-
tor<Vec4i>(), 0, Point() );
       ellipse( image_org, Ellipse[i], Scalar(0,0,255), 2, 8 );
       circle( image_org, Ellipse[i].center,5, Scalar(255,0,255), 2, 8 );
       Point2f rect_points[4]; minRect[i].points( rect_points );
       for( int j = 0; j < 4; j++ )
            line(image_org, rect_points[j], rect_points[(j+1)%4],
                  Scalar(0,0,255), 1, 8 );
   }
```



Rys. 5.7 Początkowy kształt elipsy na obrazie

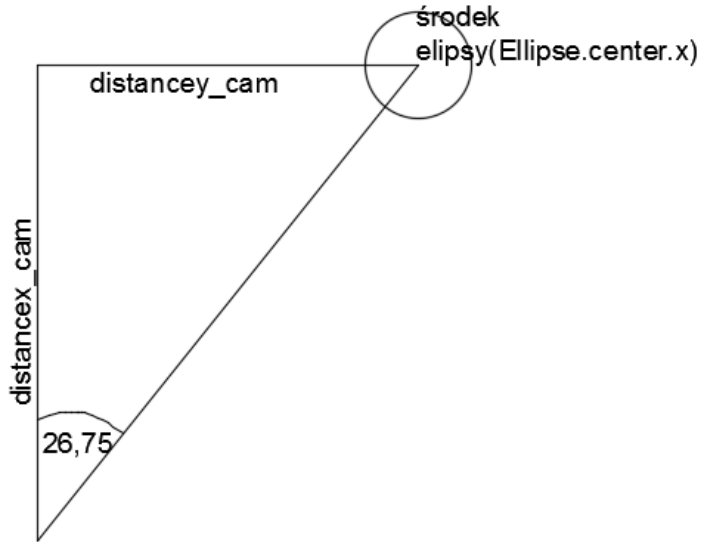
Po dopasowaniu elipsy do kształtu piłki liczone są odległości piłki od robota i od środka obiektywu kamery. Wykorzystano: średnicę piłki, która wynosi 51 mm, połowę kąta widzenia kamery w poziomie – 26.75^0, rozdzielczość obrazu oraz wymiary uzyskanej elipsy. Do wyliczenia odległości pomiędzy środkami piłki i ekranu dodatkowo uwzględniono środek elipsy. Powyższe otrzymane wyniki odpowiednio zapisano w zmiennych globalnych *distancex_cam* i *distancey_cam*. Na poniższych rysunkach 5.8 i 5.9 przedstawiono obliczenia poszczególnych odległości.



Rys. 5.8 Rysunek pomocniczy do wyliczenia odległości pomiędzy piłką a maszyną kroczącą

W kodzie:

```
distancex_cam=cos(21*deg2rad)*51*.5*image_org.cols/(Ellipse[i].size.width*tan(26.75*deg2rad));
```



Rys. 5.9 Rysunek pomocniczy do wyliczenia odległości pomiędzy środkami piłki i obiektywu kamery

W kodzie:

```
distancey_cam=(image_org.cols*0.5-Ellipse[i].center.x)*dis-
tancex_cam*tan(26.75*deg2rad)/(image_org.cols*0.5);
```

Następnie, sprawdzany jest znak wyniku zmiennej *distancey_cam* w celu określenia, po której stronie robota znajduje się szukany obiekt. Jeżeli znak jest ujemny to oznacza, że piłka znajduje się po prawej stronie robota, a jeżeli dodatni to piłka znajduje się po lewej stronie. Otrzymane wyniki są odpowiednio zapisane w wektorach *results_from_camera_x* oraz *results_from_camera_y*. Flaga określająca położenie piłki przekazywana jest do metody poruszającej maszynę w osi Y robota (*walk_sidewalk* w klasie *robot*) celem ustawienia jej do kopnięcia piłki.

Sam algorytm ustawiania wartości i saturacji zaimplementowany jest w metodzie *add_track*, której parametrami wejściowymi są obrazy HSV i oryginalny. W nim wywoływane są metody *red_ball* oraz draw_*cont*. Na początku, obraz oryginalny jest klonowany, aby nie został uszkodzony przez prowadzone na nim operacje. Następnie wywoływana jest metoda *red_ball* z ustawionymi początkowymi saturacją i wartością, które są zapisane w zmiennych globalnych przedstawionych wcześniej w tym podrozdziale. Potem sprawdzane jest czy został rozpoznany jakikolwiek okrągły obiekt przed regulacją poziomów saturacji i wartości. Najpierw zmieniana jest wartość saturacji. Za każdym razem zwiększana jest o 1, aż do maksymalnej wartości. Zmiana przechowywana jest w zmiennej *step* i domyślnie jest ustawiona na 1:

```
float precision=0.92;
Mat rgb;
rgb=img_origin.clone();
red_ball(img,rgb);
int step=1;
indicator_for_calc_of_ellipse=0;
if(contours.size()!=0)
{
do
  {
    for (minSatured=60;minSatured<180;minSatured+=step)
      {
```

Przy każdym zwiększeniu wartości nasycenia, wyliczany jest stosunek promieni elipsy i jest porównywany z największym dotychczas uzyskanym, który jest przechowywany w zmiennej globalnej *ratio_of_radii_max*. Następnie uzyskany największy stosunek promieni porównywany jest ze zmienną *precision*, która przechowuje wartość 0,92. Jeżeli aktualny stosunek promieni elipsy jest większy od 0,92 to nadpisywana jest maksymalna wartość saturacji:

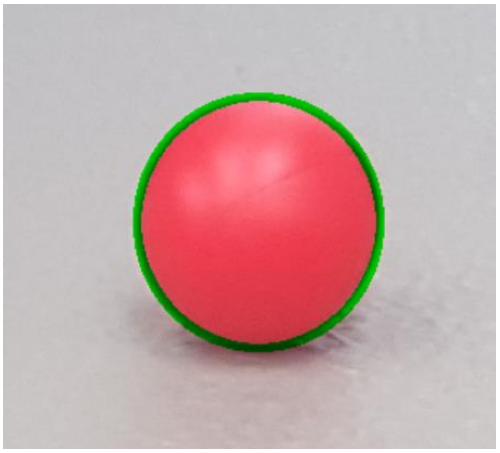
```
rgb=img_origin.clone();
red_ball(img,rgb);
if(ratio_of_radii>ratio_of_radii_max)
     {
        minSaturedab=minSatured;
        ratio_of_radii_max=ratio_of_radii;
        if(ratio_of_radii_max>precision)
           {
              indicator_for_calc_of_ellipse=1;
              break;
       }   }
```

Gdy stosunek promieni elipsy zmienia się nieznacznie tzn. o jedno lub dwa miejsca po przecinku lub nie przekroczył progu 0,92, wtedy zwiększany jest krok zmieniający wartość saturacji.

```
  if(ratio_of_radii>precision-.01) step=2;
  else
  if(ratio_of_radii>precision-.02) step=5;
  else
  step=10;
```

Po wykonaniu wszystkich operacji, zapamiętywana jest dopasowana maksymalna wartość nasycenia, zerowana jest zmienna przechowująca obliczony stosunek promieni oraz przywracany jest krok zmiany poziomu na domyślnie ustawiony. Proces dopasowania poziomu wartości odbywa się w analogicznym sposób jak powyżej przedstawiony, z tą różnicą, że wykonywany jest już z najlepiej dobraną wartością saturacji. Po ustaleniu poziomu wartości i saturacji, wywoływany jest na ekran monitora obraz z nałożonymi konturami okręgu (rys. 5.10):

```
    red_ball(img,rgb);
    namedWindow("pic",0);
    waitKey(100);
    imshow("pic",rgb);
```



Rys. 5.10 Elipsa z najlepszymi dobranymi wartościami i saturacją

W celu usprawnienia procesu rozpoznawania piłki i przekazywania wyników obliczeń, stworzono metodę *camera_film*. Jej parametrem wejściowym jest zmienna *nCoun*t, która określa liczbę zdjęć ile ma wykonać kamera w danym momencie. Na początku aparat robi 20 zdjęć próbnych bez zapisu celem przystosowania obiektywu do otoczenia:

```
int camera::camera_film( int nCount)
{
    Mat image;
    for ( int j=0; j<20; j++ ) {
    Camera.grab();
}
```

Następnie wykonuje kolejnych 5 zdjęć, które zapisuje do globalnego wektora o nazwie *pictures* w klasie *camera* :

```
for ( int j=0; j<nCount; j++ )
   {
        Camera.grab();
        Camera.retrieve ( image );
        pictures.push_back(image);
    }
```

Na tych pobranych zdjęciach będą wykonywane operacje rozpoznania piłki i określenia odległości pomiędzy nią, a robotem. Zapisane obrazy są zamieniane z RGB na HSV oraz dobierane są odpowiednie poziomy nasycenia i wartości koloru czerwonego, tak, aby odnaleźć piłkę:

```
  convert_pict_to_color_hsv(pictures,0,nCount-1);
  add_track(pictures1[nCount-1],pictures2[nCount-1]);
```

Z uzyskanych wyników zapisanych w wektorach globalnych *results_from_camera_x* i *results_from_camera_y* wyliczane są średnie arytmetyczne i następnie przekazywane są do metod w klasie *robot* odpowiedzialnych za podejście do (metoda *walk_forwards*) i wzdłuż lokalizacji piłki (metoda *walk_sidewalk*):

```
double result_x= 0;
double result_y= 0;
for (int i=0; i<results_from_camera_x.size();i++)
{
 result_x+=results_from_camera_x[i];
 result_y+=results_from_camera_y[i];
}
distancex_cam=result_x/results_from_camera_x.size();
distancey_cam=result_y/results_from_camera_y.size();
```

6. Implementacja chodów i realizacja zadania kopnięcia piłki

W niniejszym rozdziale przedstawiono tematykę związaną z lokomocją maszyny kroczącej oraz jej fizyczną interakcją z piłką. Zostały pokazane: trajektoria trapezowa nogi oraz jej pozycja bazowa, algorytmy generacji chodów trójpodporowego i pełzającego, poruszanie się maszyny w kierunku piłki oraz jej kopnięcie.

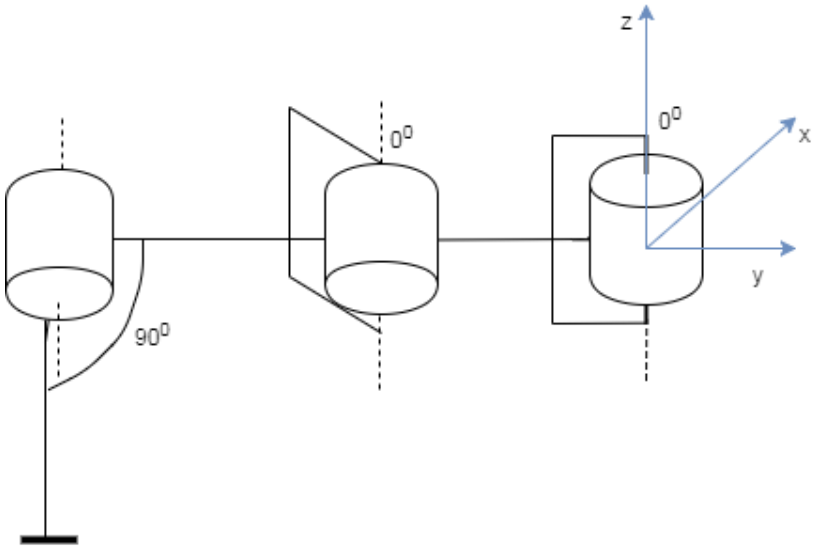
Głównym celem pracy było napisanie takiego oprogramowania, aby maszyna krocząca zlokalizowała wybrany obiekt przy pomocy systemu wizyjnego, a następnie podeszła do niego i go kopnęła. Zagadnienia związane z rozpoznawaniem obiektu i percepcją wizualną robotów przedstawiono w rozdziale 5. W projekcie zaimplementowano dwa rodzaje chodu: najwolniejszy – pełzający oraz najszybszy – trójpodporowy. Robot porusza się w kierunkach do przodu i do tyłu, w bok lewy lub prawy oraz wykonuje obrót w lewo lub w prawo. Założono, że maszyna krocząca głównie przemieszcza się chodem trójpodporowym, lecz jest także możliwość wyboru ruchu pełzającego.

Podejście do piłki wykonywane jest dwuetapowo. Na początku, robot poszukuje piłki przy pomocy kamery, którą zamontowano na przodzie robota. Jeżeli nie wykryto szukanego obiektu to maszyna wykonuje obrót o 10 stopni w lewą stronę. Po detekcji obiektu i obliczeniu odległości od niego, robot przemieszcza się w jego kierunku, ale nie podchodzi do niego całkowicie. Zatrzymuje się w odległości około 30 cm od piłki, po czym ponownie przeprowadza obliczenia jej lokalizacji, które są bardziej precyzyjne niż początkowe, gdyż dokonano ich z mniejszego dystansu. Dzięki temu, skorygowano aktualną błędną pozycję robota, która wyniknęła z zakrzywienia trajektorii podczas chodu. Po ponownej detekcji piłki, robot podchodzi do niej, a następnie przemieszcza się w lewo lub w prawo w stosunku do znalezionego obiektu, aby znalazł się on naprzeciwko wybranej nogi do kopnięcia. Noga zostaje wyprostowana w kolanie i wykonuje zamach po łuku tak, aby nadać piłce prędkość wzdłuż osi X.

6.1. Ustawienie nóg i ich trajektorie

Realizację ruchu maszyny kroczącej podzielono na poszczególne etapy. Najpierw zaprogramowano sterowanie pojedynczym serwem, a potem nogą. Następnie, zaprojektowano trajektorie dla chodu trójpodporowego i pełzającego. Na końcu zaimplemetnowano podejście do piłki.

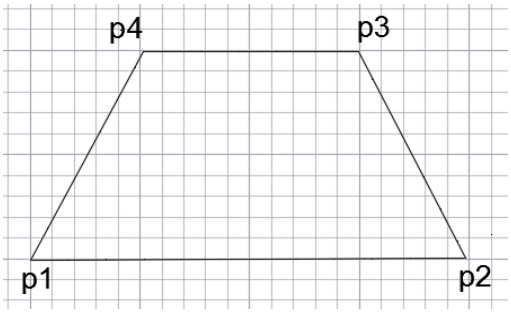
Zanim robot wykona jakikolwiek ruch, każda z nóg musi przyjąć odpowiednią pozycje początkową. Dla każdego z rodzaju chodu jest ona różna, ale żeby została prawidłowo ustawiona, każda z pozycji nóg maszyny musi być skorygowana, dlatego zaimplementowano pozycję bazową nogi. Przedstawiona jest ona poniżej na rysunku 6.1. W takim ułożeniu nogi, kąt przy biodrze wynosi 0^0, kąt przy udzie – 0^0, kąt przy goleni – 90^0. Przyjęcie postawy, która składa się z pozycji bazowych nóg, zostało nazwane pozycją bazową maszyny kroczącej.



Rys. 6.1 Pozycja bazowa nogi

Po włączeniu robota, jego nogi przyjmują pozycje bazowe. Niestety pod wpływem ciężaru robota, nogi nie osiągają w pełni zadanych pozycji. W projekcie oprócz kamery nie ma innych czujników, a wybrane serwa nie mają enkoderów, które zwróciłyby informacje o bieżących położeniach nóg, dlatego biorąc pod uwagę wszystkie powyższe czynniki, zastosowano korekcje pozycji bazowych nóg. Odbywa się ona poprzez podniesienie każdej z kończyn oraz stopniowe jej opuszczanie na podłoże do faktycznego zadanego położenia.

Każda z nóg, niezależnie od chodu, porusza się po trajektorii trapezowej, która została przedstawiona na rysunku 6.2. Została ona wybrana ze względu na łatwość jej realizacji w porównaniu z trajektorią eliptyczną oraz na możliwość jej prostego modyfikowania np. skrócenie kroku lub zmiana kształtu trajektorii na trójkątną. Co więcej, dzięki dość długiej fazie podparcia, możliwe jest poprawne zaimplementowanie ruchu trójpodporowego oraz pełzającego.

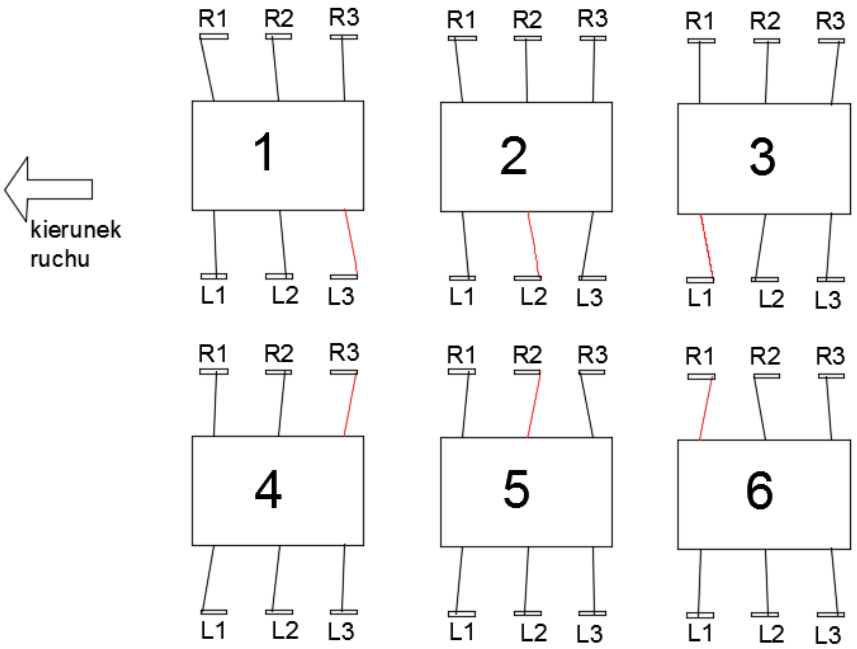


Rys. 6.2 Trajektoria trapezowa nogi

Trajektoria ma cztery punkty kontrolne nazwane p1, p2, p3, p4. Noga osiąga każdy z punktów kontrolnych poprzez przyjęcie odpowiednich wartości kątów dla poszczególnych przegubów, które są wyliczone z kinematyki odwrotnej.

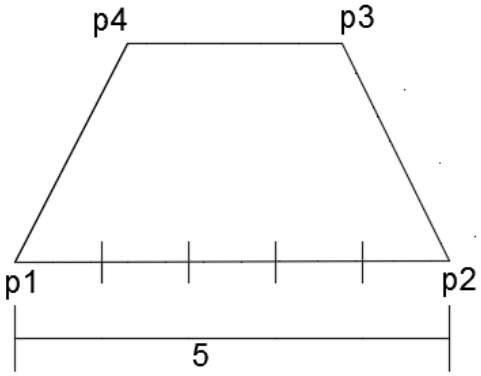
Serwo analogowe ma drobne niedoskonałości wynikające z rozstawu zębatki, przez co trajektoria końcówki nogi pomiędzy punktami jest łukiem. Najbardziej jest to widoczne w trakcie poruszania się końcówki nogi wzdłuż najdłuższego odcinka trajektorii, czyli pomiędzy punktami p1, a p2. Jednym z efektów ubocznych jest ślizganie się nóg. W celu jego redukcji, a przede wszystkim uzyskania trajektorii nogi na odcinku p1-p2 najbardziej zbliżonej do lini prostej, zastosowano podział danego odcinku na kilka mniejszych. Im mniejsza jest ich długość, tym promień łuków jest większy, co w efekcie końcowym spowodowało, że trajektoria nogi na danym odcinku jest prawie liniowa.

Kolejnym istotnym zagadnieniem związanym z wykonywaniem trajektorii jest odpowiednia synchronizacja nóg. Istotne jest, aby wybrane nogi poruszały się jednocześnie. Dzięki temu, zapewniona jest stabilność robota podczas inicjacji ruchu oraz w trakcie jego wykonywania. Dodatkowo, podczas chodu nogi nie zderzają się ze sobą. Szczególnie ważne jest to w ruchu pełzającym (rys. 6.3), ponieważ faza przenoszenia nogi w powietrzu trwa 5 razy krócej niż faza przesuwania nogi po podłożu. Dodatkowo, dużym utrudnieniem w tym chodzie jest fakt, że nogi poruszają się falowo, czyli jedna po drugiej.



Rys. 6.3. Ruch pełzający. Na czerwono zaznaczono nogę, która kończy fazę podporową, a zaczyna fazę przenoszenia. Cyfry 1,2,3,4,5,6 oznaczają kolejne fazy ruchu

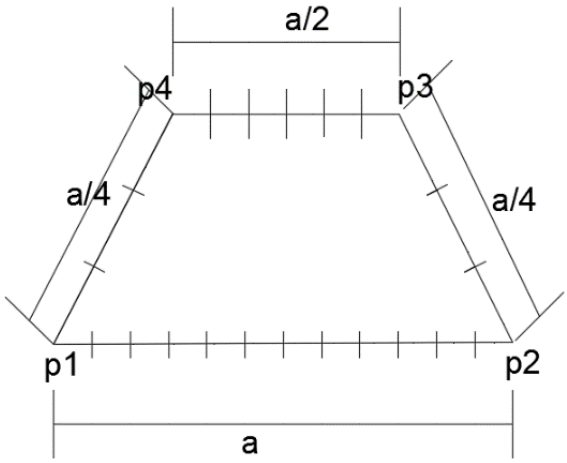
Aby noga wykonywała poprawnie całą wybraną trajektorię z zachowaniem specyfikacji chodu, każda z nóg musi przyjąć odpowiednią pozycję, z której zacznie wykonywać ruch po trajektorii. W tym celu najdłuższy odcinek trajektorii, czyli pomiędzy punktami p1 i p2, podzielono na stałą i niemodyfikowalną liczbę odcinków, równą 5 (patrz rys. 6.4). Przy wyborze strony lewej rozpoczynającej, maszyna krocząca przyjmuje pozycję nóg dla fazy ruchu numer 1, którą przedstawiono na rysunku 6.3. W przypadku prawej strony zaczynającej ruch, nogi po stronie prawej przyjmują pozycję nóg ze strony lewej z fazy numer 1 z powyższego rysunku, a lewe przyjmują pozycję prawych..



Rys. 6.4 Trajektoria trapezowa nogi dla chodu pełzającego

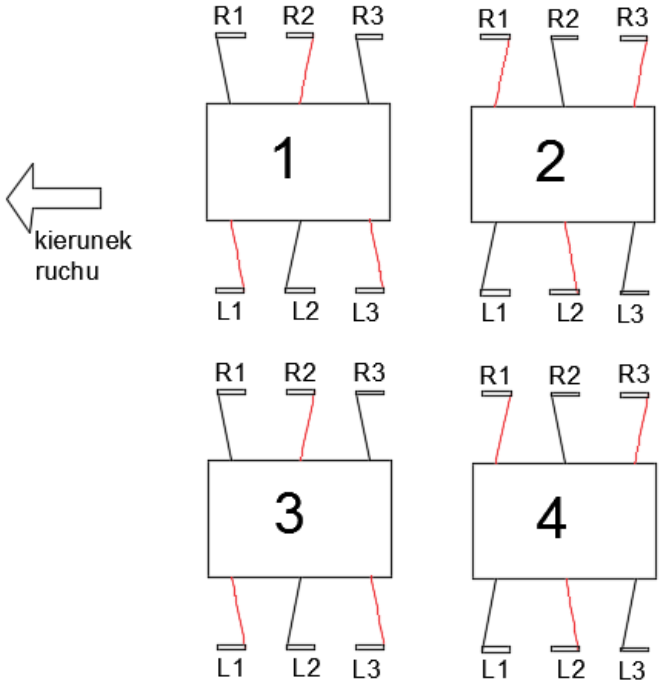
W porównaniu do ruchu pełzajacego, w którym na początku każda z nóg musi przyjąć pozycję na wyznaczonych przez odcinki punktach, w chodzie trójpodporowym nie ma takich wymagań. W początkowej fazie nogi są ustawione w punktach p1 lub p2, co umożliwiło wyznaczenie zmiennej liczby odcinków, na ile ma być podzielona odległość pomiędzy p1-p2.

Podział całej trajektorii na odcinki w ruchu trójpodporowym przedstawiono na poniższym rysunku 6.5.



Rys. 6.5 Trajektoria trapezowa nogi dla chodu trójpodporowego, a – liczba odcinków dzieląca odległość pomiędzy punktami p1 i p2

W tym chodzie występują tylko dwie fazy ruchu: 3 nogi diagonalne przemieszczają się po ziemi, podczas gdy pozostałe 3 uniesione są w powietrzu (rys. 6.6). Sprawia to, że sam ruch nie jest skomplikowany, dlatego liczba odcinków jest niestała i może być zmieniona w programie przez użytkownika.



Rys. 6.6 Ruch trójpodporowy. Na czerwono zaznaczono nogę, która kończy fazę podporową, a zaczyna fazę przenoszenia. Cyfry 1,2,3,4 oznaczają kolejne fazy ruchu

Przy dobieraniu liczby odcinków, należało uwzględnić prędkość, z jaką będzie poruszać się dana noga. Przy ich zbyt małej liczbie i dużej prędkości, noga robota nie osiąga niektórych zadanych punktów kontrolnych trajektorii. Dla maksymalnej prędkości serw, domyślnie przyjęta liczba odcinków pomiędzy punktami p1,a p2 w programie wynosi 20. Dla takich parametrów, noga wykonuje poprawnie całą trajektorię. W zależności od tego, po której stronie robota wybrana jest noga zaczynająca ruch, punkty początkowe trajektorii są w różnych miejscach. Dla strony lewej, tor ruchu kończyny zaczyna się od punktu p1, a dla prawej od punktu p2. Dla wybranej nogi zaczynającej ruch po stronie prawej, punkty początkowe są ustawione w analogiczny sposób jak dla strony lewej.

W celu podejścia do piłki, maszyna krocząca porusza się jednym z dwóch rodzajów chodu: pełzającym lub trójpodporowym. Niezależnie od wybranego chodu, algorytm sposobu przemieszczania się jest taki sam. W kierunku piłki robot rusza się krokami o pełnej długości oraz skróconymi, ponieważ odległość pomiędzy nim, a obiektem nie jest całkowicie podzielna przez długość pełnego kroku. W pracy zaproponowano dwie metody realizacji podejścia do obiektu. Pierwszy sposób to, gdy maszyna pokonuje dystans tylko za pomocą kroku o skróconej długości, który jest mniejszy od maksymalnej jego długości. Długość kroku jest dobierana tak żeby liczba kroków do wykonania, która jest ilorazem odległości pomiędzy piłką, a robotem i długości kroku była liczbą całkowitą. Maszyna wtedy dojdzie do piłki całkowitą liczbą kroków o zadanej długości. Wszystkie nogi zakończą ruch w fazie podparcia. Rozwiązanie to wymaga dość wysokiej precyzji ustawiania nóg, co w przypadku tego projektu jest nieosiągalne z powodu wykorzystania serw analogowych, które nie charakteryzują się wcześniej wymienioną właściwością. To sprawia, że noga robota nie będzie ustawiona tak dokładnie jak jest zadane w programie, co może spowodować, że robot ominie piłkę. Dodatkowo, pod uwagę trzeba wziąć niedokładność pomiaru wykonywanego przez kamerę.

Drugim sposobem jest podejście za pomocą dużych kroków na pewną odległość do piłki oraz wykonanie jednego mniejszego kroku, co zaimplementowano w programie. Długość pełnego kroku nie zmienia się i domyślnie jej wartość wynosi 10 cm. Z uzyskanej odległości z pomiarów kamery oraz z długości kroku, obliczana jest liczba wykonania pełnych kroków przez nogę. Pozostała odległość stanowi długość skróconego kroku. To rozwiązania jest łatwe w implementacji, lecz tak jak i poprzednie wyżej opisane, jest bardzo uzależnione od pomiarów z kamery.

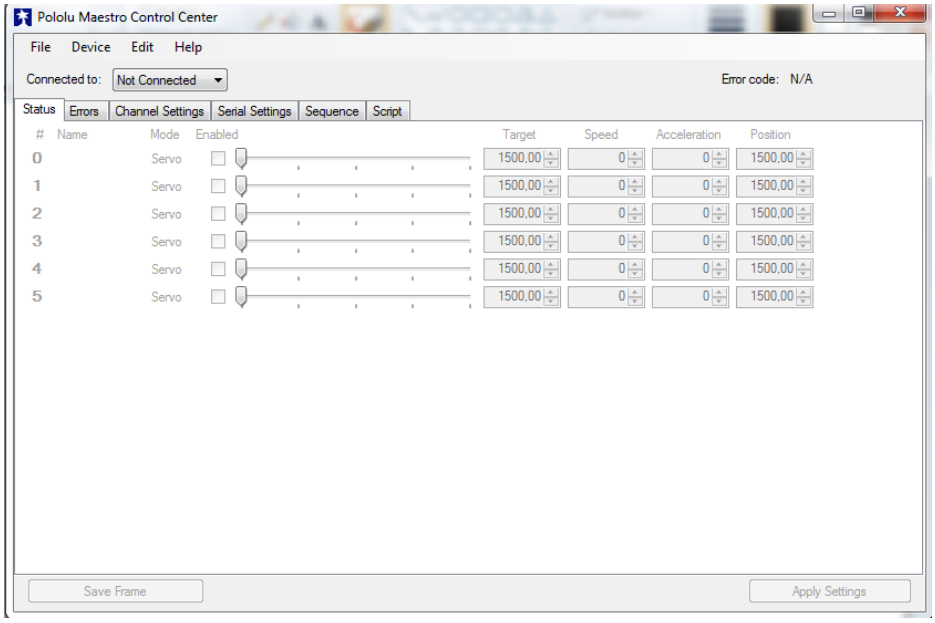
W przypadku, gdy maszyna krocząca dojdzie do piłki, wyprostowuje nogę i porusza nią po łuku, by kopnąć obiekt. Noga jest wtedy wyprostowana w kolanie i zatacza łuk o promieniu 19,4 cm (rys. 6.7). Przy udzie nachylona jest ona do podłoża pod kątem 25^0. Dzięki temu, jest duże prawdopodobieństwo, że noga trafi w piłkę, a ta nabierze wystarczająco dużej prędkości, by przebyć dość długą drogę zanim się zatrzyma.

Rys. 6.7 Ułożenie nogi do kopnięcia piłki

6.2 Sterowanie serwem maszyny kroczącej

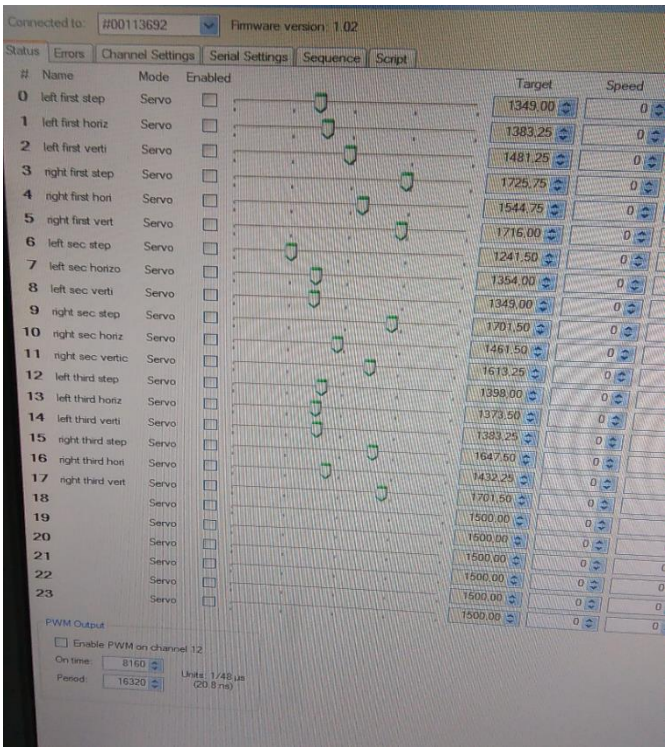
W niniejszym podrozdziale opisano proces implementacji sterowania nogą w oprogramowaniu. Pozostałe etapy przedstawiono w kolejnych podrozdziałach.

Na początku ustanowiono komunikację pomiędzy komputerem bazowym, a sterownikiem serw Maestro Pololu. Do tego celu posłużono się programem producenta sterownika Maestro Control Center [19]. Komunikacja pomiędzy dwoma urządzeniami odbyła się po kablu Micro USB. Z pomocą programu wyznaczono pozycje bazowe nóg oraz wartości minimalne i maksymalne serw.



Rys. 6.8 Okno programu Maestro Control Center po włączeniu programu

Przy każdym uruchomieniu programu, na monitorze pojawia się okno startowe z zablokowanymi serwami (rys. 6.8). Oprogramowanie automatycznie wykrywa podłączony sterownik wraz z liczbą do niego przyłączonych serw. Jeżeli sterownik został wykryty, to w pasku „Connected to" pojawia się możliwość przestawienia stanu z „Not Connected" na numer sterownika, tak jak na rysunku 6.9.



Rys. 6.9 Okno programu Maestro Control Center, gdy został wykryty sterownik wraz z przyłączonymi do niego serwami. Zdjęcie pokazuje ustawione pozycje serw przez użytkownika

W przypadku bezbłędnej komunikacji pomiędzy wszystkimi urządzeniami, użytkownik ma możliwość ustawienia parametrów serw takich jak prędkość, przyśpieszenie, opóźnienie etc. W zakładkach widocznych na rysunkach 6.8 i 6.9 można ustawić bardziej zaawansowane parametry takie jak: raportowanie błędów, interfejs komunikacji (USB, UART) oraz napisać własny skrypt do sterowania serwami.

Po wyznaczeniu wartości minimalnych i maksymalnych serw oraz pozycji bazowych nóg w programie Maestro Control Center, uzyskane wartości przeniesiono do programu w języku C++ i umieszczono w tablicy *tab_of_servos* w klasie bazowej projektu *stdfax*. Tablica ma 18 wierszy i 4 kolumny. Liczba wierszy odzwierciedla liczbę serw, a poszczególne kolumny zawierają numery serw oraz ich wartości minimalne, pozycje bazowe nóg i wartości maksymalne. Fragment tablicy *tab_of_servos* w kodzie:

```
int tab_of_servos[18][4] = { { 0, 550, 1496, 2500 },
{ 1, 550, 2095, 2500 }
```

W przypadku serw obsługujących golenie (numery serw: 5,8,11,14,17), zastosowano dodatkowe obliczenia w celu wyeliminowania wszelkich nierówności pomiędzy ułożeniem nóg.

Wynikały one z fizycznych właściwości serw oraz przyjętego układu współrzędnych, który opisano w podrozdziale 4.2.1, dlatego dodatkowo wprowadzono współczynnik skalujący o wartości 10,3. Na 1^0 obrotu serwa przypada około 10 mikrosekund trwania impulsu sterującego. Serwa obsługujące golenie przesunięto o 90^0 i ich nastawy pomnożono przez przyjętą skalę. Fragment kodu tablicy *tab_of_servos* :

```
int tab_of_servos[18][4] = {
{ 5, 550, 1123 + 90 * 10.3, 2500 },
{ 8, 550, 1933 - 90 * 10.3, 2500 },
{ 11, 550, 1176 + 90 * 10.3, 2500 },
```

Z wyżej wymienionych tablic korzystają w szczególności metody z klasy *servo*. W tej klasie odbywa się zarządzenie pojedynczym serwem. Są to metody: *maestroGetPosition*, *maestroSetTarget*, *maestroSetSpeed, poweroff_servo*, *go*. Pierwsze trzy z wyżej wymienionych metod powstały na bazie kodu pobranego ze strony producenta sterownika [19]. Dodatkowo zapewnił on także dwa protokoły do komunikacji z sterownikiem: Compact i Pololu. Do wysyłania i odbierania komend wybrano protokół Compact, ze względu na łatwość implementacji.

W celu wysyłania komend na sterownik, najpierw musi zostać otwarty port komunikacyjny pomiędzy portem USB komputera lub Raspberry Pi, a sterownikiem Pololu. Adres wirtualny portu przechowuje zmienna *fd* w klasie *stdfax*. Dla systemu Windows, ustawionym domyślnym adresem portu jest COM6 lub USBSER000, a dla systemu Linux to ttyACM0. W konstruktorze klasy *stdfax* następuje bezpośrednie jego otwarcie i ustawienie flag do ustanowienia komunikacji:

```
Linux: const char * device = "/dev/ttyACM0";
Windows: const char * device = \\\\.\\USBSER000 lub \\\\.\\COM6”
fd = open(device, O_RDWR | O_NOCTTY);
```

Zadaniem funkcji *maestroGetPosition* jest odczytanie aktualnej pozycji serwomechanizmu. Jej parametrami wejściowymi są port *fd* oraz numer serwa. W metodzie wysyłana jest komenda o numerze 0x90 oraz numer serwa do sterownika. Następnie otrzymywana jest odpowiedź, która zapisana jest w zmiennej *response*. Wadą tej funkcji jest to, że zwraca ona ostatnią ustawioną w programie pozycję serwa, a nie faktycznie osiągniętą – wynika to z tego, iż w projekcie używane są serwa analogowe, które nie umożliwiają odczytania ich rzeczywistego położenia.

Metoda *maestroSetTarget* wysyła sygnał do sterownika w celu poruszenia wybranym serwem. Jej parametrami wejściowymi są port *fd*, numer oraz zadana pozycja serwa. W tej metodzie wysyłane jest polecenie do sterownika, które zawiera numer 0x84, numer serwa oraz dwa bajty z wartością zadanej pozycji. Struktura polecenia w kodzie:

```
unsigned char command[] = {0x84, channel, target & 0x7F, target >> 7 &
0x7F};
```

Wysyłanie polecenia do sterownika odbywa się przez funkcję *write*:

```
write(fd,command,sizeof(command));
```

Dodatkowo w programie sprawdzane jest czy zadana pozycja jest osiągalna przez serwo:

```
if(target*0.25<tab_of_servos[channel][1])
 {
      cout<<"servos limit min " <<channel<<endl;
      return -1;
 }
else if(target*0.25>tab_of_servos[channel][3])
 {
      cout<<"servos limit max "<<channel<<endl;
      return -1;
 }
```

W celu poruszenia serwem, zadana nastawa przemnożona jest przez 4 tak, aby mogła ona być ustawiona przez sterownik. W tablicy *tab_of_servos* znajdują się wartości nieprzemnożone przez 4, dlatego zmienną *target* podzielono przez 4, aby móc ją porównać z ograniczeniami w tablicy.

W metodzie *maestroSetSpeed* ustawiana jest prędkość serwa. Parametrami wejściowymi są port *fd*, numer serwa oraz zadana prędkość. Do sterownika wysyłane są polecenie o numerze 0x87, numer serwa oraz w dwóch bajtach zadana szybkość.

Metoda *go* odpowiada za ustawienie serwa do wybranej pozycji. Parametrem wejściowym jest zadana wartość pozycji, która jest potem przemnażana przez zmienną *a*, czyli skalę 10,3. Następnie obliczony iloczyn jest dodawany do środkowej wartości pozycji serwa. Otrzymana suma jest jeszcze przemnażana przez 4 i wysyłana jest do sterownika przez funkcję *maestroSetTarget*. Fragment metody w programie:

```
int servo::go(int b)
{
  double c=0;
  c = center + b*a;
  int success;
  success=maestroSetTarget( fd, channel, c * 4);
```

Ostatnia funkcja o nazwie *poweroff_servo* w tej klasie jest odpowiedzialna za wyłączenie serwa. Nie przyjmuje żadnych parametrów. Korzysta z funkcji *maestroSetTarget*. Wyłączenie odbywa się poprzez wysłanie do sterownika wartości 0. Fragment metody w programie:

```
int servo::poweroff_servo()
{
      int success=maestroSetTarget(fd, channel, 0);
}
```

6.3 Sterowanie nogą maszyny kroczącej

Po napisaniu oprogramowania do sterowania serwem i sprawdzeniu bezbłędności jego działania, utworzono w programie sterowanie nogą maszyny kroczącej. Do tego celu w klasie bazowej *stdfax* zaimplementowano podział robota na stronę lewą i prawą, do których przyporządkowano odpowiednie nogi. Przypisane poszczególne nogi do stron umieszczono w tablicy *tab_of_legs* w klasie *stdfax*. Ma ona 6 wierszy oraz 4 kolumny, które odpowiadają liczbie nóg oraz numerom poszczególnych serw tworzących nogę i stronę, do której są przypisane. Fragment tablicy *tab_of_legs*:

```
int tab_of_legs[6][4] = { { 0, 1, 2,right_side },
{ 3, 4, 5, left_side },
{ 6, 7, 8,right_side }
```

W celu ruszenia nogą wymagane jest wywołanie najpierw konstruktora klasy *leg*, a następnie metody *set_servos_in_leg*. Konstruktor klasy przyjmuje, jako parametr numer nogi, która ma być uaktywniona. Samo ustawienie nogi w metodzie *set_servos_in_leg* odbywa się z wykorzystaniem metody *go* z klasy *servo*. Funkcja *set_servos_in_leg* wymaga podania wartości pozycji serw odpowiednio przy udzie, biodrze i goleni, jako argumentów wejściowych. Metoda *set_servos_in_leg* w kodzie:

```
int leg::set_servos_in_leg(double h, double m, double n)
{
   int success;
   if (side_of_robot ==right_side)
   {
      success=(servos[0]->go(-h));
      success=(servos[1]->go(-m));
      success=(servos[2]->go(n));
   }
   else
   {
       success=(servos[0]->go(h));
       success=(servos[1]->go(m));
       success=(servos[2]->go(-n));
   }
```

Ustawianie nogi wyłącznie za pomocą wartości kątów dla poszczególnych przegubów nie pozwala na łatwą rozbudowę projektu. Co więcej, bardziej przyswajalną formą dla człowieka jest podanie wartości przesunięcia końcówki nogi w centymetrach lub milimetrach, dlatego na poziomie sterowania nogą zaimplementowano obliczenia zadań kinematyki prostej i odwrotnej. Obliczenie zadania kinematyki odwrotnej znajduje się w funkcji o nazwie *move_leg_to_xyz*. Jej parametrami wejściowymi są trzy współrzędne końcówki nogi x, y, z, które musi osiągnąć

W tej metodzie odbywa się przeliczenie współrzędnych na miary kątów poszczególnych przegubów, a następnie przy pomocy funkcji *set_servos_in_leg* ustawienie ich wartości. Obliczenie zadania kinematyki odwrotnej przedstawiono w podrozdziale 3.2.2. Obliczenie zadania kinematyki prostej zaimplementowane jest w funkcji *forward_kinematics*. Jej parametrami wejściowymi są wartości pozycji serw przy udzie, biodrze i goleni. Na ich podstawie wyliczane są współrzędne końcówki nogi, które są ustawiane za pomocą funkcji *move_leg_to_xyz*. Rozwiązanie zadania kinematyki prostej przedstawiono w podrozdziale 3.2.3. Opisanie współrzędnych końcówki nogi w programie odbywa się przy pomocy zmiennych *x*, *y*, *z* z klasy *point*. Są to jedyne zmienne w tej klasie. Nie ma ona żadnych innych metod.

Ustawienie prędkości nogi odbywa się w metodzie *set_leg_speed*, a jej wyłączenie w *leg_poweroff*. Obie metody korzystają z funkcji z klasy *servo*. Pozycja bazowa nogi ustawiana jest w funkcji *canon_position*, która nie przyjmuje żadnych parametrów wejściowych. W celu ustawienia nogi, do metody *set_servos_in_leg* przekazywane są następujące argumenty: 0^0, 0^0, 90^0. Metoda *canon_position* w kodzie:

```
int leg::canon_position()
{
  int success=set_servos_in_leg(0, 0, 90);
  if(success!=0)
   {
      perror(("error in leg::canon_position"));
      return -1;
   }}
```

6.3.1 Sterowanie nogami maszyny kroczącej

Ręczne wpisywanie punktów, które ma osiągnąć noga oraz tworzenie na bieżąco trajektorii jest dość uciążliwe oraz czasochłonne, dlatego aby ominąć tę niedogodność, zaimplementowano gotowe punkty trajektorii trapezowej. Dla przyszłych zastosowań stworzono także punkty dla chodu na boki oraz kontrolę rozmiaru kroku pod postacią współczynnika beta. Tworzenie trajektorii nogi następuje w metodzie *create_default_nodes w klasie robot*. Punkty trajektorii utworzono z wykorzystaniem długości poszczególnych odcinków nogi, które zaimplementowano w klasie *stdfax*. Długości odcinków podane są w milimetrach.

Fragment kodu klasy *stdfax*:

- biodro: int hip=29;
- udo: int thigh=57;
- goleń: int tibia=108.

Dla chodu tył/przód przyjęto następujące współrzędne punktów trajektorii przedstawionej na rysunku 6.2:

- p1(50*beta, thigh + hip-30, tibia+30);
- p2(-50*beta, thigh + hip-30,tibia+30);
- p3(-20*beta, thigh +hip-20, tibia -20);
- p4(20*beta, thigh + hip-20,tibia -20).

Dla chodu na boki współrzędne punktów wyżej wymienionej trajektorii dobrano eksperymentalnie:

- p5 (0,111*beta,103),
- p6 (0,36*beta,103),
- p7 (0,36*beta,78),
- p8 (0,121*beta,73).

Następnie ustawiana jest kolejność punktów trajektorii, jaką noga będzie wykonywać dla wybranego kierunku poruszania się. Kierunek jest wyznaczony przez zmienną ze struktury o nazwie *direction* z klasy *robot*. Robot porusza sie w przód lub tył, w lewy lub prawy bok, obrócić się w lewo lub w prawo. Struktura *direction* w kodzie:

```
struct direction  {
    int forwards=0;
    int backwards=1;
    int turn_left=2;
    int turn_right=3;
    int sidewalk_left=4;
    int sidewalk_right=5;
    } rdirection;
```

Dla każdego kierunku ruchu utworzono oddzielny wektor do przechowania wygenerowanej trajektorii. Fragmenty kodu metody *create_default_nodes* z klasy *robot*:

- dla ruchu do przodu:

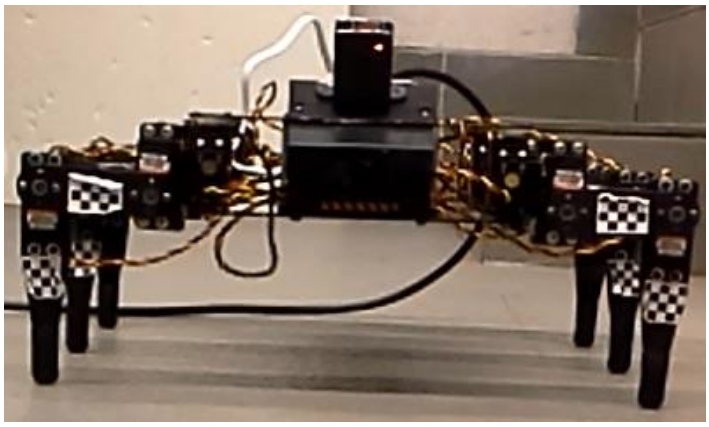
```
forward_points.push_back(p2);
forward_points.push_back(p1);
forward_points.push_back(p4);
forward_points.push_back(p3);
```

- dla ruchu w bok

```
sidewalk_left_points.push_back(p5);
sidewalk_left_points.push_back(p6);
sidewalk_left_points.push_back(p7);
sidewalk_left_points.push_back(p8);
sidewalk_right_points.push_back(p6);
sidewalk_right_points.push_back(p5);
sidewalk_right_points.push_back(p8);
sidewalk_right_points.push_back(p7);
```

Dodatkowo, w tej metodzie przechowywane są także rozmiary kroków dla ruchów w przód/tył oraz na boki odpowiednio w zmiennych *size_of_step_forwards* oraz *size_of_step_sidewalk*. Rozmiary kroków są podane w centymetrach.

Po włączeniu, zmianie kierunku lub sposobu poruszania się, nogi maszyny kroczącej przyjmują pozycje bazowe jak pokazano na rysunku 6.10. Sposób stawiania nóg oraz korekcji ich pozycji przedstawiono w rozdziale 6.1. Ustawienie nóg i ich trajektorie. W tej postawie wykonywane są także pomiary przez kamerę.



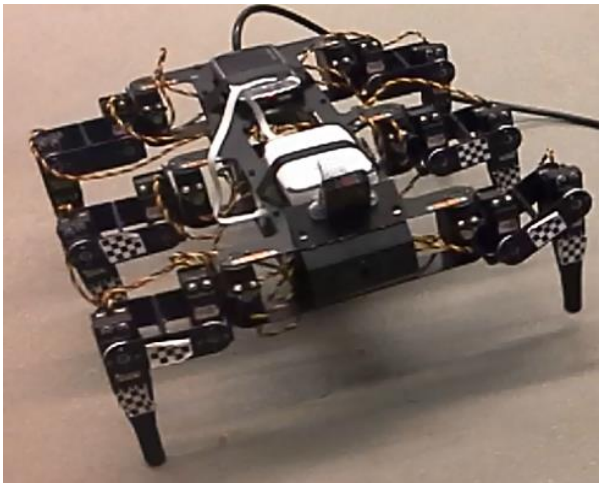
Rys. 6.10 Pozycja kanoniczna maszyny kroczącej

Pozycja kanoniczna robota ustawiana jest w metodzie *robot_canon_posit* w klasie *robot*, która korzysta z metody *canon_position* z klasy *leg.* Fragment kodu *robot_canon_posit:*

```
int robot::robot_canon_posit()
{
      legs_speed(200);
      int j;
      for (j = 0; j < 6; j++){
            legs_indicator[j]->canon_position();
      }
       while (maestroGetMovingState(fd));
```

Implementację algorytmu korekcji pozycji nóg przedstawiono poniżej (rys. 6.11). W celu ułatwienia stopniowego opuszczenia nóg wykorzystano funkcję *forward_kinematics* z klasy *leg*:

```
for (j=0;j<6;j++)
{
      legs_indicator[j]->forward_kinematics(0,-20,135);
      usleep(2e4);
      legs_indicator[j]->forward_kinematics(0,-30,121);
      usleep(2e4);
      for (int i=0;i<30;i++)
            {
                  legs_indicator[j]->forward_kinematics(0,-30+i,121-i);
                  usleep(2e4);
  }           }
}
```



Rys. 6.11 Poprawienie ułożenia lewej nogi

Ustawienie prędkości poruszania się nóg wykonywane jest w metodzie *legs_speed* w klasie *robot.* Funkcja przyjmuje tylko jeden argument wejściowy, zmienną *speed*, która określa prędkość. Przypisując zmiennej *speed* wartość 0, ustawia się w ten sposób maksymalną/domyślną prędkość serwomechanizmów. Można także nadać jej odpowiednio dużą wartość (np. 200), aby osiągnąć maksymalną prędkość.

Wyłącznie nóg odbywa się w metodzie *zero* w klasie *robot*. Funkcja nie przyjmuje żadnych parametrów wejściowych. Do wyłączenia nóg korzysta z funkcji *leg_poweroff* z klasy *leg*.

6.3.2 Generacja chodu pełzającego

Niezależnie od wykonywanego sposobu przemieszczania się, noga porusza się po trajektorii trapezowej, którą podzielono na odcinki, co opisano w rozdziale 6.1. Wyznaczenie długości odcinków odbywa się w metodzie *calc_vector_for_trajec* w klasie *leg.* Do tej metody przekazywane są dwa punkty tworzące trajektorię o współrzędnych x, y, z i liczba odcinków na ile ma być podzielona długość pomiędzy punktami. Długość wyznaczonych odcinków jest stała. Metoda *calc_vector_for_trajec* w programie:

```
vector<point3d> leg::calc_vector_for_trajec(point3d v1, point3d v2,
float a)
{
      vector      <point3d> result;
      double deltax = (v2.x - v1.x) / a;
      double deltay = (v2.y - v1.y) / a
      double deltaz = (v2.z - v1.z) / a;
      point3d tmp;
      for (int i = 0; i < a; i++)
      {
            tmp.x = v1.x + i*deltax;
            tmp.y = v1.y + i*deltay;
            tmp.z = v1.z + i*deltaz;
            result.push_back(tmp);
      }
      return result;
}
```

Metoda *calc_vector_for_trajec* nie dzieli całej trajektorii na odcinki tylko wyznacza odcinki pomiędzy wybranymi punktami tworzącymi trajektorię. W przypadku ruchu pełzającego w celu wyznaczenia odcinków dla całej trajektorii, utworzono metodę *calc_trajectory_for_crawling* w klasie *leg*. Metoda ta, wykorzystując wcześniej opisaną funkcję *calc_vector_for_trajec* dzieli całą trajektorię nogi na odcinki oraz układa je w odpowiedniej kolejności do wykonania. Przyjmuje, jako parametry wejściowe 4 punkty trajektorii oraz współczynnik kontrolujący rozmiar kroku, który domyślnie ustawiono na pełny krok. Najdłuższą część trajektorii, czyli pomiędzy punktami p1, a p2 podzielono na 5 części. Następnie do wyliczonych i zapisanych współrzędnych dodawane są pozostałe punkty trajektorii. Kod metody *calc_trajectory_for_crawling:*

```
int  leg::calc_trajectory_for_crawling(vector<point3d>nodes,float factor)
{
  traj_crawling.clear();
  traj_crawling = calc_vector_for_trajec(nodes[0],nodes[1], factor*(6 * (1+
+nodes.size()-2 )));
 for (int i = 2; i <nodes.size(); i++)
 {
            traj_crawling.push_back(nodes[i]);
 }
```

W metodzie *gen_traj* w klasie *robot* zdefiniowano ułożenie początkowe nóg, które przedstawiono, jako fazę ruchu 1 na rysunku 6.3.

Generowane jest 6 punktów, które wyznaczają końce i początki odcinków, na które podzielono trajektorię. Otrzymane punkty przechowywane są w zmiennej *initial_trajectory_position* w klasie *leg*. Kod metody *gen_traj:*

```
int robot::gen_traj( vector <point3d>nodes,int side_of_robot,float factor)
{
   for (int i = 0; i < 6; i++)
   {
      if (legs_indicator[i]->side_of_robot == side_of_robot)
      {
         legs_indicator[i]->calc_trajectory_for_crawling(nodes,factor);
         legs_indicator[i]->initial_trajectory_position = i * 3;
   }  }
```

Dopiero w metodzie *crawling* następuje ruch nóg. Jej parametrami wejściowymi są zadane punkty trajektorii, zmienna *k* określająca liczbę powtórzeń pętli oraz dwie zmienne kontrolujące rozmiar kroku nóg po lewej i po prawej stronie. Wygenerowane początkowe punkty z funkcji *gen_traj* są zapamiętane w zmiennej *current_trajectory_position* z klasy *leg*, która przechowuje aktualną pozycję nóg.Implementacja ruchu pełzającego w kodzie:

```
int robot::crawling(vector <point3d> nodes, vector <point3d> nodes2,int
k,float factor_left_leg, float factor_right_leg)
{
  gen_traj( nodes, left_side,factor_left_leg);
  gen_traj( nodes2,right_side,factor_right_leg);
  double x, y, z;
  for (int i = 0; i < 6; i++)
   {
     legs_indicator[i]->current_trajectory_position =      legs_indica-
tor[i]->initial_trajectory_position;
   }


for (int w = 0; w < k; w++){
  for (int j = 0; j < legs_indicator[0]->traj_crawling.size(); j++){
      for (int i = 0; i < 6; i++)
      {
        legs_indicator[i]->current_trajectory_position++;
        legs_indicator[i]->current_trajectory_position =
        legs_indicator[i]>current_trajectory_position%legs_indicator[i]->
                  traj_crawling.size();
        x = legs_indicator[i]->traj_crawling[legs_indicator[i]->
                  current_trajectory_position].x;
        y = legs_indicator[i]->traj_crawling[legs_indicator[i]->
                  current_trajectory_position].y;
        z = legs_indicator[i]->traj_crawling[legs_indicator[i]->
                  current_trajectory_position].z;
       legs_indicator[i]->move_leg_to_xyz(x, y, z);
      }while (maestroGetMovingState(fd));
}}
```

Metoda *maestroGetMovingState* sprawdza, czy którekolwiek z serw jest w trakcie ruchu. Nie wskazuje konkretnego serwa, które nie ukończyło ruchu. Metoda wstrzymuje wykonywanie następnych poleceń, dopóki wszystkie serwa w nogach robota nie zakończą ruchu. Jest to jeden z dwóch sposób synchronizacji nóg. Drugą możliwością jest zastosowanie metody delay, która opóźnia wykonywanie komend o tyle ile zadał użytkownik w mikrosekundach.

6.3.3 Generacja chodu trójpodporowego

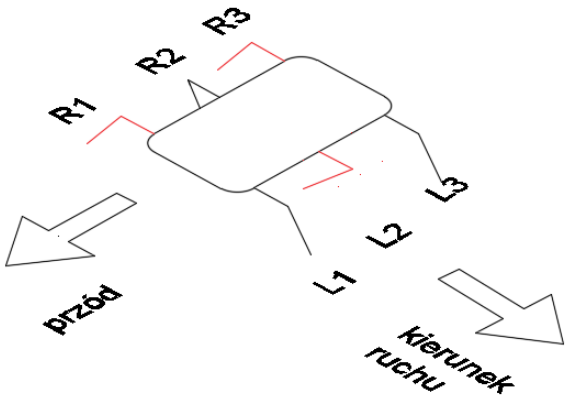
Pierwszą operacją wykonywaną w generacji ruchu trójpodporowego jest podział całej trajektorii nogi na odcinki. Odbywa się to w metodzie *calc_for_trajectory_tripod* w klasie *leg*, która tak jak w przypadku ruchu pełzającego, korzysta z metody *calc_vector_for_trajec*. Jako parametry wejściowe przyjmuje zadane punkty trajektorii, liczbę punktów na ile ma być podzielona trajektoria, wskaźnik strony robota oraz współczynnik kontrolujący rozmiar kroku, który domyślnie ustawiono na pełny krok. W tym ruchu, liczbę odcinków na ile ma być podzielona trajektoria nogi można zmienić w programie, dlatego zaimplementowano dla każdej pary punktów trajektorii osobny podział odległości pomiędzy nimi. Fragment w kodzie:

```
int leg::calc_for_trajectory_tripod(vector<point3d> nodes, int a, int pa-
rameter,float factor)
{
 std::vector      <point3d> result0,result1, result2, result3,result4;
 result0 = calc_vector_for_trajec(nodes[0],nodes[1], factor*a);
 result1 = calc_vector_for_trajec(nodes[1],nodes[2], factor*a/4);
 result2 = calc_vector_for_trajec(nodes[2],nodes[3], factor*a/2);
 result3 = calc_vector_for_trajec(nodes[3],nodes[0], factor*a/4);
 traj_tripod.clear();
```

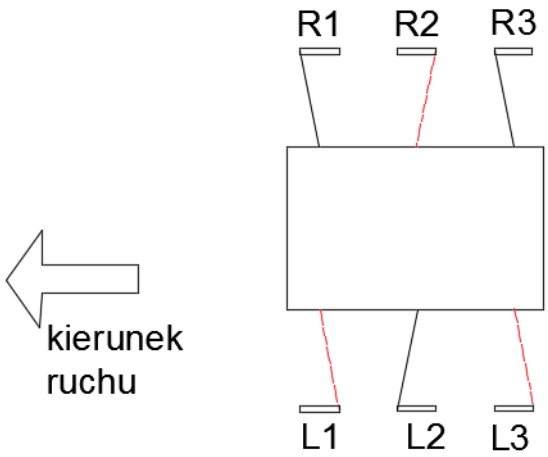
W zależności od strony robota, noga zaczyna wykonywać trajektorię z różnego punktu początkowego. W przypadku, gdy wskaźnik strony równy jest 0 wtedy ustawiana jest trajektoria dla strony lewej, a równy 1 dla strony prawej. Korzystając z rysunku z trajektorią nogi dla ruchu trójpodporowego [6.4], dla strony lewej ruch nogi zaczyna się od punktu pierwszego – p1, a dla strony prawej od punktu drugiego – p2. Fragment w kodzie:

```
int leg::calc_for_trajectory_tripod(vector<point3d> nodes, int a, int pa-
rameter,float factor)
{
 std::vector      <point3d> result0,result1, result2, result3,result4;
 result0 = calc_vector_for_trajec(nodes[0],nodes[1], factor*a);
 result1 = calc_vector_for_trajec(nodes[1],nodes[2], factor*a/4);
 result2 = calc_vector_for_trajec(nodes[2],nodes[3], factor*a/2);
 result3 = calc_vector_for_trajec(nodes[3],nodes[0], factor*a/4);
 traj_tripod.clear();
 if (parameter == 0)
 {
traj_tripod.insert(traj_tripod.end(), result0.begin(), result0.end());
traj_tripod.insert(traj_tripod.end(), result1.begin(), result1.end());
traj_tripod.insert(traj_tripod.end(), result2.begin(), result2.end());
traj_tripod.insert(traj_tripod.end(), result3.begin(), result3.end());
 }
 if (parameter !=0)
 {
traj_tripod.insert(traj_tripod.end(), result1.begin(), result1.end());
traj_tripod.insert(traj_tripod.end(), result2.begin(), result2.end());
traj_tripod.insert(traj_tripod.end(), result3.begin(), result3.end());
traj_tripod.insert(traj_tripod.end(), result0.begin(), result0.end());
 }
      return 0;
}
```

Poruszanie nogami w ruchu trójpodporowym jest znacznie łatwiejsze w implementacji niż w ruchu pełzającym. Sam ruch zaimplementowano w metodzie *tripod*, natomiast ustawienie początkowe nóg jest w metodzie *pos_for_walk*. W programie wygenerowano różne początkowe ułożenia nóg w zależności od kierunku poruszania się maszyny kroczącej, dlatego jednym z parametrów wejściowych metody *pos_for_walk* jest zmienna *probe*, która wskazuje na kierunek ruchu robota. W tej funkcji nogi są ustawiane w ten sposób, że trzy z nich (L1, L3, R2) znajdują się w końcowym punkcie fazy podparcia (p2), z którego od razu przejdą do fazy przenoszenia, podczas gdy pozostałe (R1, R3, L2) spoczywają na podłożu w punkcie p1 (końcowym punkcie fazy przenoszenia i jednocześnie początkowym punkcie fazy podparcia), aby odepchnąć ciało robota. Ustawienie nóg dla ruchu w kierunku przód/tył przedstawiono na rysunku 6.13. W przypadku poruszania się na boki, dla początkowego ułożenia nóg przyjęto eksperymentalnie następujące punkty: pierwszy punkt (0,121,73), drugi punkt (0,36,78). Pozycję początkową dla ruchu w bok lewy pokazano na rysunku 6.12.



Rys. 6.12 Ustawienie nóg dla ruchu w lewo. Na czerwono zaznaczone są nogi, które kończą fazy podporowe, a zaczynają fazy przenoszenia



Rys. 6.13 Ustawienie nóg dla ruchu w przód. Na czerwono zaznaczone są nogi, które kończą fazy podporowe, a zaczynają fazy przenoszenia

W metodzie *tripod*, przed rozpoczęciem ruchu, nogi ustawiane są do pozycji początkowej w metodzie *pos_for_walk*. W zależności od zmiennej określającej kierunek ruchu, czyli zmiennej *number*, poszczególnym nogom przypisane są odpowiednie punkty trajektorii wraz z ich kolejnością, które zapisano w zmiennych *nodes* i *nodes2*. W zmiennej *nodes* znajdują się punkty dla nogi lewej, a w *nodes2* dla nogi prawej. Dla zmiennej *number* równej 0, pobierane są punkty dla trajektorii ruchu w tył/przód, a dla *number* równej 1 dla ruchu w bok. Do wygenerowania kolejności wykonywania punktów wykorzystano funkcję *calc_for_trajectory_tripod* z klasy *leg.* Zmienna *m* określa prawą lub lewą stronę robota, a zmienne *factor_left_leg* i *factor_right_leg* kontrolują rozmiar kroku robota analogicznie po lewej i prawej stronie. Domyślnie ustawiono je na pełną długość kroku. Fragment metody *tripod* w kodzie:

```
int robot::tripod(int number, vector <point3d> nodes, vector <point3d>
nodes2, int m, int k, float factor_left_leg, float factor_right_leg)
{
      pos_for_walk(number,nodes, nodes2);
      if (number == 0)
      {
        legs_indicator[0]->calc_for_trajectory_tripod(nodes, m,
                  0,factor_left_leg);
        legs_indicator[1]->calc_for_trajectory_tripod(nodes2, m,
                        1,factor_right_leg);
        legs_indicator[2]->calc_for_trajectory_tripod(nodes2, m,
                        1,factor_right_leg);
        legs_indicator[3]->calc_for_trajectory_tripod(nodes, m,
                  0,factor_left_leg);
        legs_indicator[4]->calc_for_trajectory_tripod(nodes, m,
                  0,factor_left_leg);
        legs_indicator[5]->calc_for_trajectory_tripod(nodes2, m,
                        1,factor_right_leg);
      }
if (number == 1)
{
    legs_indicator[0]->calc_for_trajectory_tripod(nodes, m,
                  0,factor_left_leg);
    legs_indicator[1]->calc_for_trajectory_tripod(nodes2, m,
                  1,factor_right_leg);
    legs_indicator[2]->calc_for_trajectory_tripod(nodes, m,
                  1,factor_right_leg);
    legs_indicator[3]->calc_for_trajectory_tripod(nodes2, m,
                  0,factor_left_leg);
    legs_indicator[4]->calc_for_trajectory_tripod(nodes, m,
                  0,factor_left_leg);
    legs_indicator[5]->calc_for_trajectory_tripod(nodes2, m,
            1,factor_right_leg);
}
```

Do poruszania nogą po wybranej trajektorii zastosowano metodę *move_leg_to_xyz* z klasy *leg*. Zmienna *k* określa ile razy noga wykona trajektorię. Tak jak w przypadku chodu pełzającego, do zapewnienia, że każda z nóg osiągnie zadany punkt użyto metodę *maestroGetMovingState* z klasy *robot*.

Fragment w kodzie:

```
for (int g = 0; g < k; g++){
      for (int i = 0; i < legs_indicator[1]->traj_tripod.size(); i++){
                                legs_indicator[0]->move_leg_to_xyz(legs_indi-
cator[0]->traj_tripod[i].x,
legs_indicator[0]->traj_tripod[i].y, legs_indicator[0]->traj_tripod[i].z);
                         legs_indicator[1]->move_leg_to_xyz(legs_indica-
tor[1]->traj_tripod[i].x,
 legs_indicator[1]->traj_tripod[i].y, legs_indicator[1]->traj_tripod[i].z);
                  legs_indicator[2]->move_leg_to_xyz(legs_indica-
tor[2]->traj_tripod[i].x, legs_indicator[2]->traj_tripod[i].y, legs_indica-
tor[2]->traj_tripod[i].z);
                  legs_indicator[3]->move_leg_to_xyz(legs_indica-
tor[3]->traj_tripod[i].x, legs_indicator[3]->traj_tripod[i].y, legs_indica-
tor[3]->traj_tripod[i].z);
                  legs_indicator[4]->move_leg_to_xyz(legs_indica-
tor[4]->traj_tripod[i].x, legs_indicator[4]->traj_tripod[i].y, legs_indica-
tor[4]->traj_tripod[i].z);
                                legs_indicator[5]->move_leg_to_xyz(legs_indi-
cator[5]->traj_tripod[i].x, legs_indicator[5]->traj_tripod[i].y, legs_indi-
cator[5]->traj_tripod[i].z);
        while (maestroGetMovingState(fd));
}}
```

6.4 Implementacja podejścia do piłki

W metodzie *walk* z klasy *robot* ustawiany jest rodzaj i kierunek ruchu oraz prędkość, z jaką ma się poruszać maszyna krocząca. Metoda korzysta z wszystkich metod oraz gotowych trajektorii opisanych w tym rozdziale. Powstała ona w celu ułatwienia sterowania oraz skrócenia czasu wywoływania ruchu robota. Pierwszym parametrem wejściowym tej funkcji jest zmienna strukturalna o nazwie *gait* określająca rodzaj chodu. Struktura znajduje się w nagłówku klasy *robot*:

```
struct  gait{
      int tripod=0;
      int crawl=1;
      }rgait ;
```

Do wyboru jest jeden z dwóch rodzajów chodu: trójpodporowy – 0 lub pełzający – 1. Jako parametry wejściowe metoda *walk* przyjmuje kierunek ruchu robota, liczbę punktów podziału trajektorii dla ruchu trójpodporowego oraz prędkość, z jaką poruszają się nogi. Dalszymi parametrami są rozmiar kroku dla wszystkich nóg oraz rozmiar kroku dla stron lewej i prawej. Domyślnie te zmienne ustawiono na pełne długości.

W metodzie, przed wyborem kierunku i sposobu poruszania się, ustawiana jest prędkość serwomechanizmów (*legs_speed*) oraz wybierane są kierunek i odpowiednia trajektoria nóg (*create_default_nodes*):

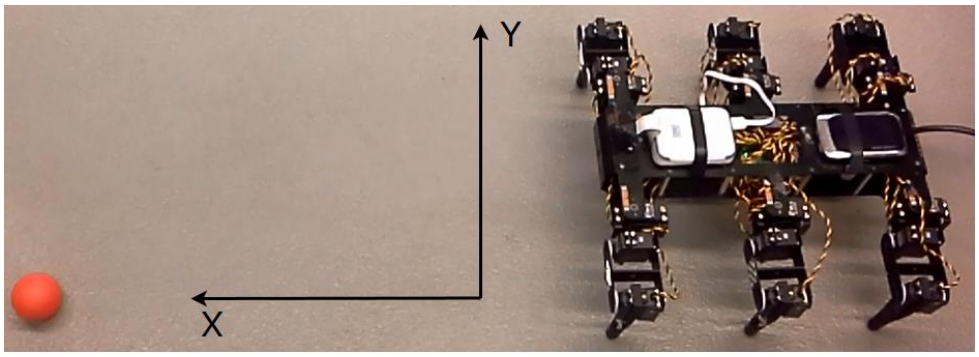
```
void robot::walk(int gait , int direction ,int samples, int iteration, int
speed,float beta,float factor_left_leg, float factor_right_leg){
    legs_speed(speed);
    create_default_nodes(direction,beta);
```

Następnie, w zależności od wybranego rodzaju chodu, wygenerowano ruch pełzający lub trójpodporowy. W przypadku chodu trójpodporowego, liczbę odcinków podziału najdłuższego odcinka trajektorii ustawiono na domyślną wartość, czyli na 20. Do tego celu zmienną *samples* ustawiono na 0. Jest to optymalna liczba próbek, ponieważ równoważy prędkość nogi z prawidłowym osiągnięciem przez nią każdego punktu trajektorii. Fragment kodu dla kierunku ruchu do przodu dla dwóch rodzajów chodu w metodzie *walk*:

```
if(gait==1)
{
     if(speed==0) legs_speed(200);
     switch (direction)
{
        case 0:
crawling(forward_points,forward_points,iteration,
factor_left_leg,factor_right_leg);
         break;
      }
}
if(gait==0)
{
      if(speed==0) legs_speed(1000);
      if(samples==0) samples=20;
      switch (direction)
{
        case 0:
  tripod(0,forward_points,forward_points,samples,iteration,
factor_left_leg,factor_right_leg);
      break;
}     }
```

Pozostałe kierunki zaimplementowano w analogiczny sposób.

Za przemieszczenie się maszyny kroczącej w kierunku piłki odpowiadają metody: *walk_forwards* oraz *walk_sidewalk* z klasy *robot*. Obie metody do wykonania ruchu wykorzystują metodę *walk*. Podejście do piłki w osi X realizuje metoda *walk_forwards.* Osie, wzdłuż których porusza się maszyna w kierunku piłki pokazano na rysunku 6.14.



Rys. 6.14 Osie, wzdłuż których porusza
się maszyna krocząca w kierunku piłki

Metoda *walk_forwards* przyjmuje, jako parametry wejściowe rodzaj chodu, obliczoną w centymetrach odległość od piłki z klasy *camera*, prędkość nóg oraz długość kroku wszystkich nóg, które ustawiono domyślnie na pełne długości. Najpierw wybierana jest trajektoria nóg dla ruchu w tył/przód oraz rodzaj chodu. Wyliczoną odległość od piłki zamieniono na liczbę cykli do zrealizowania. Jeden cykl to wykonanie przez wszystkie 6 nóg jednego kroku o zadanej długości. Do tego celu użyto długość kroku, która wynosi domyślnie 10 cm i jest zapisana w zmiennej *size_of_step_forwards* w funkcji *create_default_nodes*. Liczba cykli do wykonania przez nogi jest wynikiem dzielenia z resztą odległości z kamery przez długość kroku. Fragment w kodzie:

```
double remnant=std::fmod((dis_from_ball-5),size_of_step_forwards);
```

Zmniejszenie odległości do piłki o 5 centymetrów ma na celu zapewnienie, że piłka nie zostanie ominięta podczas chodu. Jeżeli reszta z dzielenia będzie równa 0, to znaczy, że robot może przejść całą odległość w pełnych krokach. Jeżeli jest różna od 0, to znaczy, że oprócz pełnych kroków, maszyna krocząca musi dodatkowo wykonać jeden mniejszy.

Przy określeniu liczby cykli do wykonania, użyto metody *floor*, która zwraca największą wartość całkowitą nieprzekraczającą wartości podanej liczby.Uzyskana z obliczeń liczba cykli dzielona jest przez dwa, ponieważ zarówno nogi po stronie prawej muszą wykonać taką samą liczbę kroków jak nogi po stronie lewej. Długość mniejszego kroku obliczana jest, jako iloraz z reszty uzyskanej przy wyliczeniu liczby cykli dla kroków o pełnej długości oraz rozmiaru kroku. Następnie uzyskany wynik, tak jak w przypadku liczby cykli, dzielony jest przez dwa i wpisany jest na miejscu beta, ponieważ liczba cykli mniejszych kroków wynosi jeden. Współczynnik beta zmienia długość kroku, a ponieważ zastosowano mnożenie przez betę a nie dzielenie, dlatego poprzednio uzyskaną resztę zmniejszono jeszcze o pełną długość kroku, aby uzyskać pożądane zmniejszenie długości kroku. Fragment z obliczeniem małych kroków w kodzie:

```
walk(gait,rdirection.forwards,0,floor(((dis_from_ball-5)/size_of_step_for-
wards)*0.5), speed,beta, factor_left_leg, factor_right_leg);
    usleep(1e5);
   walk(gait,rdirection.forwards,0,1,speed,
(remnant/size_of_step_forwards)*0.5, factor_left_leg, factor_right_leg);
```

Metoda *walk_sidewalk* porusza maszyną kroczącą na bok w kierunku lewo/prawo, czyli w osi Y. Jej celem jest takie ustalenie pozycji robota, aby mógł on kopnąć wyprostowaną nogą piłkę. Parametrami wejściowymi funkcji są rodzaj chodu oraz zmienna *probe*, której wartość jest pobierana z funkcji *draw_cont* w klasie *camera*. Zmienna *probe* określa, po której stronie robota znajduje się piłka. Dla *probe* równej 0, piłka jest po prawej stronie robota, a dla 1 po lewej stronie. Kolejnymi parametrami wejściowymi są wyliczona odległość pomiędzy środkami piłki i obiektywu kamery z funkcji *draw_cont,* prędkość, zmienna *prep_for_ball* określająca kierunek poruszania się robota oraz zmienne kontrolujące rozmiar kroków, które są ustawione na pełne długości. Na początku wybierana jest trajektoria nóg dla ruchu na boki oraz rodzaj chodu. Następnie wyliczana jest liczby cyklido wykonania jak w metodzie *walk_forwards*. Fragment w kodzie:

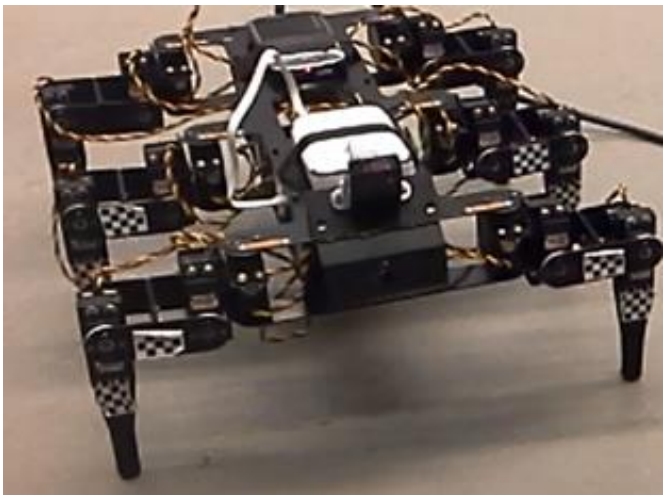
```
remnant=std::fmod(dis_from_cent, size_of_step_sidewalk);
```

Dla reszty z dzielenia równej 0, robot podchodzi do piłki pełnymi krokami. Porusza się on pełnymi krokami w lewo lub prawo w zależności od wartości zmiennej *probe*. Gdy maszyna nie może podejść tylko pełnymi krokami do obiektu, dodatkowo wyliczony jest jeden cykl mniejszych kroków. Uwzględniając wartość zmiennej *prep_for_ball* maszyna porusza się w wyznaczonym kierunku do momentu aż obiekt znajdzie się na środku obiektywu kamery. Potem, robot przesunie się w przeciwną stronę do kierunku ruchu o odległość, jaka jest pomiędzy końcówką nogi, a kamerą, aby kopnąć piłki. Fragment kodu przedstawiający ruch w prawo:

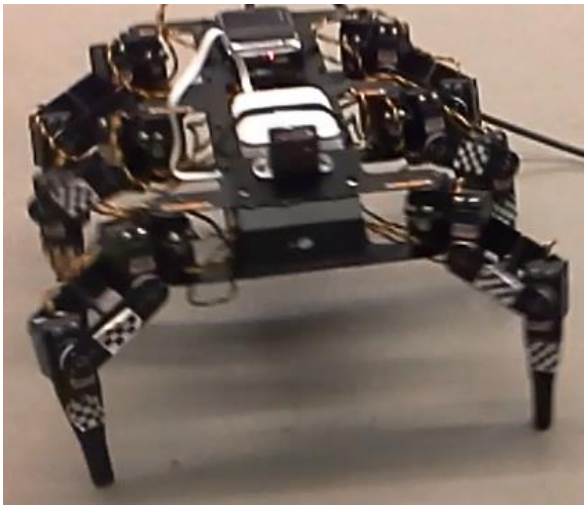
```
if (probe==0)
{
if(remnant==0)
{

walk(gait,rdirection.sidewalk_right,0,(dis_from_cent/size_of_step_side-
walk),speed,beta,factor_left_leg,factor_right_leg);
}
else
{
if(prep_for_ball==0)
{
walk(gait,rdirection.sidewalk_right,0,floor((-
dis_from_cent)/size_of_step_sidewalk),speed,beta,factor_left_leg,fac-
tor_right_leg);
usleep(1e5);
walk(gait,rdirection.sidewalk_right,0,1,speed,(remnant/size_of_step_side-
walk),factor_left_leg,factor_right_leg);
usleep(1e5);
}
   if(prep_for_ball==1)
walk(gait,rdirection.sidewalk_left,0,2,speed,0.4,factor_left_leg,fac-
tor_right_leg);
}
}
```

Dodatkowo w metodzie *walk_sidewalk* zaimplemetowano obrót robota na wypadek nieznalezienia obiektu. W tej sytuacji, zmienna *prep_for_ball* przyjmuje wartość 3. Podczas obrotu, każda z nóg wykonuje tylko pojedyncze przejście po punktach trajektorii.

W trakcie ruchu w kierunku piłki za pomocą wyżej opisanych metod, maszyna krocząca nie przemieszczała się całkowicie prosto po wyznaczonym torze z powodu ślizgania się nóg. Wynikało to z tego, że przy aktualnym ułożeniu ciała robota, jego nogi nie były przenoszone dostatecznie wysoko, przez co ich końcówki zachaczały o podłoże. Zaimplementowano metodę *up_and_down*, która w zależności od parametru *probe* obniża lub podwyższa korpus maszyny kroczącej. Podczas chodu, ciało robota jest unoszone do góry, przez co jego nogi mają większe pole manewru i ich końcówki podczas fazy przenoszenia nie dotykają podłoża. Dla zmiennej *probe* równej 0, nogi przyjmują współrzędne 0,86, 108 i robot się obniża w celu kopnięcia piłki (rys. 6.15). Ciało robota się unosi dla *probe* równej 1, nogi przyjmują współrzędne 0, 56, 138 (rys. 6.16).



Rys. 6.15 Robot z opuszczonym korpusem



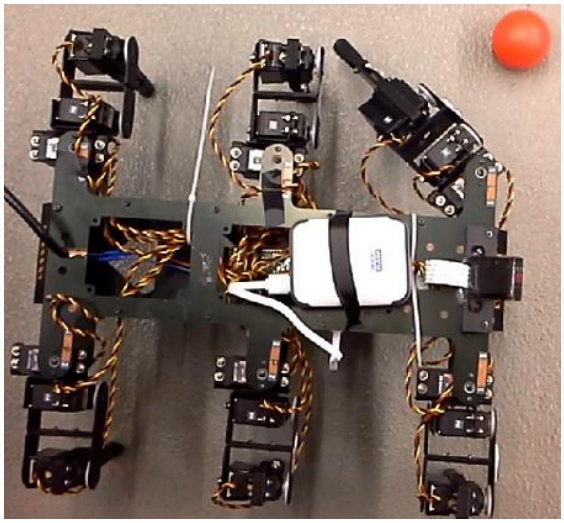
Rys. 6.16 Robot z podniesionym korpusem

6.5 Interakcja z piłką

Algorytmy podejścia do piłki i jej kopnięcia wywoływane są w metodzie *go_to_the_ball* w klasie *robot*. Nadanie piłce prędkości i kierunku odbywa się w funkcji *kick_the_ball* w klasie *robot*, której parametrem wejściowym jest zmienna *probe* określająca nogę do tej czynności. Dla *probe* równej 0 wybierana jest noga prawa, a dla 1 – lewa. Do zaimplementowania kopnięcia piłki użyto metodę *set_servos_in_leg* z klasy *leg*:

```
if (probe==0){
  legs_indicator[0]->set_servos_in_leg(0,-25,10);
  usleep(5e5);
  legs_indicator[0]->set_servos_in_leg(45,25,0);
  usleep(5e5);
  legs_indicator[0]->set_servos_in_leg(-45,25,0);
  usleep(5e5);
  legs_indicator[0]->set_servos_in_leg(0,0,90);
```

Podczas ruchu, noga jest cała wyprostowana i porusza się po łuku tak, że piłka jest kopnięta przez część goleni (rys. 6.17). Dzięki temu, piłka toczy się szybko i pokonuje długą trasę. Aby ruch pomiędzy poszczególnymi ustawieniami nogi był płynny, zastosowano opóźnienie czasowe w postaci funkcji *usleep*.



Rys. 6.17 Przygotowanie nogi do kopnięcia piłki

W metodzie *go_to_the_ball* w odpowiedniej kolejności wywoływane są metody, które poruszają maszynę w kierunku piłki oraz ją kopią i metody z klasy *camera*, które zajmują się detekcją obiektu na obrazie i obliczeniem do niego odległości. Parametrami wejściowymi funkcji *go_to_the_ball* są rodzaj chodu, prędkość nóg oraz zmienne kontrolujące długości kroków, które są domyślnie ustawione na pełne długości. Po włączeniu robota, nogi są ustawiane do pozycji bazowych, a następnie metoda *camera_film* z klasy *camera* poszukuje piłkę na uzyskanych zdjęciach z kamery.

Fragment w kodzie:

```
     while(1){
       robot_canon_posit();
      usleep(1e5);
      cam->camera_film(5);
      usleep(1e5);
```

Potem sprawdzane jest, czy wykryto kontury obiektu. Jeżeli nie znaleziono piłki, ciało robota unosi się do góry i obraca się w lewo. Następnie, korpus opuszcza się, robot przyjmuje pozycję bazową i wykonuje kolejne zdjęcia i wyszukanie obiektu:

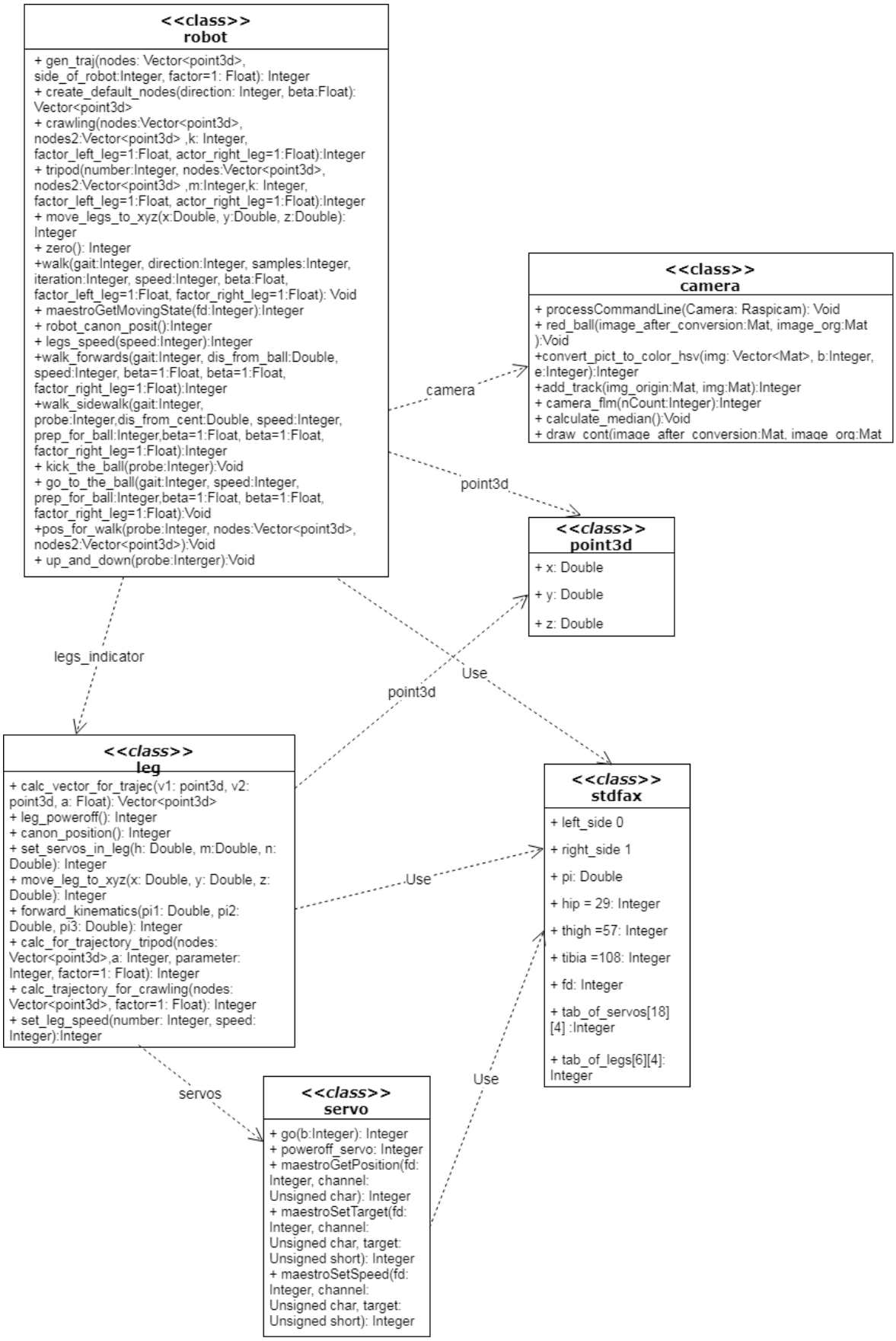
```
if (cam->contours.size()!=0)
….
{
    up_and_down(1);
    usleep(1e5);
    walk(gait,rdirection.turn_left,0,1,speed, 0.8 ,
factor_left_leg, factor_right_leg);
    usleep(1e5);
    up_and_down(0);
}
```

Dopóki nie zostaną wykryte kontury piłki, do tego czasu powyżej opisany proces będzie się powtarzał. W przypadku odnalezienia obiektu, zostają obliczone odległości dzielące robota od piłki i korzystając z metod *walk_forwards* i *walk_sidewalk* robot podchodzi do piłki po raz pierwszy na niewielki dystans – 30 cm od piłki się zatrzymuje. Pierwsza próba podejścia skutkuje dość dużym błędem wynikającym z niedokładnego oszacowania położenia piłki z dużego dystansu oraz odchylenia robota od zadanej trajektorii. Przyjęto, że po pierwszym podejściu, robot zatrzymuje się w celu korekcji swojej pozycji oraz dokładniejszego zlokalizowania piłki, aby zwiększyć możliwość jej kopnięcia. Fragment w kodzie:

```
if (cam->contours.size()!=0){
      up_and_down(1);
      usleep(1e5);
      dis_x=((cam->distancex_cam)/10)-30;
      dis_y=((cam->distancey_cam)/10);
       walk_forwards(gait,dis_x,speed,beta,factor_left_leg,
factor_right_leg);
      usleep(1e5);
      up_and_down(0);
      robot_canon_posit();
      usleep(1e5);
      walk_sidewalk(gait,cam->probe_indicator_for_camera,dis_y,speed,0);
      usleep(1e5);
      robot_canon_posit();
      cam->camera_film(5);
      if(cam->contours.size()!=0)
      {up_and_down(1);
      dis_x=((cam->distancex_cam)/10);
      dis_y=((cam->distancey_cam)/10);
      usleep(1e5);
      walk_forwards(gait,dis_x,speed,beta,factor_left_leg,
factor_right_leg);
      usleep(1e5);
      up_and_down(0);
      robot_canon_posit();
```

```
        usleep(1e5);
        walk_sidewalk(gait,cam->probe_indicator_for_camera,dis_y,speed,1);
        usleep(1e5);
        robot_canon_posit();
        usleep(1e5);
        kick_the_ball(cam->probe_indicator_for_camera);
}
```

Poniżej znajduje się diagram zależności pomiędzy klasami użytymi w projekcie (rys. 6.18).



Rys. 6.18 Diagram zależności pomiędzy klasami użytymi w projekcie

7. Podsumowanie i wnioski

W niniejszej pracy magisterskiej stworzono oprogramowanie w języku C++ do sterowania sześcionożną maszyną kroczącą. Zaprojektowano trajektorie nóg dla dwóch różnych rodzajów chodu. Zaimplementowano chód trójpodporowy i pełzający, detekcję piłki, podejście do niej i jej kopnięcie. Zrealizowano wszystkie założone cele.

Do poprawnego wysterowania ruchu nogi w chodzie trójpodporowym wymagane jest dobranie odpowiedniej liczby odcinków podziału trajektorii, w zależności od prędkości nogi. Ze względu na specyfikacje tego chodu zbyt mała liczba odcinków przy maksymalnej prędkości spowodowała zniekształcenie kształtu wykonywanej trajektorii. Problem ten nie występuje w chodzie pełzającym, ponieważ liczba odcinków podziału trajektorii jest stała. W obu tych chodach istotną kwestią była synchronizacja nóg tak, aby wybrany chód został wykonany prawidłowo oraz by robot zachował stabilność podczas przemieszczania się. Do tego celu wprowadzono różne ustawienia początkowe nóg oraz opóźniono ich ruch poprzez zaimplementowane przez producenta mechanizmy sterownika.

Pomimo długiej fazy podporowej w trajektorii trapezowej, robot do piłki nie podchodził w linii prostej. Spowodowane to było drobnymi błędami w ustawieniu początkowymi nóg i ich wysterowaniu oraz błędami pomiarowymi z kamery. W celu rozwiązania tego problemu, zaimplementowano pozycję bazową robota, która koryguje ustawienia nóg zanim rozpocznie się chód. Dodatkowo, wprowadzono początkowe ustawienie nóg dla dwóch chodów oraz podniesiono korpus robota do przemieszczania się. Po zastosowaniu wyżej wymienionych czynności, błędy odometryczne zmniejszyły się znacznie, lecz nie zniknęły całkowicie.

Zgodnie z oczekiwaniami, przestrzeń barw HSV okazała się bardziej odporna na zmienne warunki oświetlenia niż RGB. Pomimo zastosowanego szerokiego przedziału barwy do rozpoznawania koloru czerwonego, pod wpływem różnego oświetlenia, kształt piłki nie zawsze był poprawnie rozpoznawany. Oprócz tego, do detekcji kształtu były także brane pod uwagę czerwony refleks od podłoża oraz cień piłki. W celu zniwelowania wyżej wymienionych czynników, zastosowano manipulację wartościami intensywności i saturacji obrazu. Dzięki temu, udało się znacznie zminimalizować ich wpływ na rozpoznawanie obiektu oraz poprawić obliczenia położenia piłki. Nie udało się ich wyzerować do końca.

W pracy zaimplementowano dość skomplikowane w swojej złożoności, podstawowe chody maszyny kroczącej, co otwiera możliwości do zaprogramowania bardziej złożonych zachowań robota, takich jak omijanie przeszkód czy pokonywanie schodów. Jest to jednakże uwarunkowane dostępem do informacji sensorycznej na temat jego otoczenia. W tym celu wymagane jest zastosowanie innych, dodatkowych czujników oprócz kamery.

Zaletą mikrokomputera Raspberry PI 2 jest duża liczba pinów, do których można podłączyć większość czujników dostępnych na rynku. Proponowanymi sensorami są ultradźwiękowy czujnik odległości oraz czujnik nacisku na stopy. Za pomocą czujnika odległości można by oszacować odległość pomiędzy maszyną kroczącą, a przeszkodami. W oprogramowaniu można by zaimplementować metodę do omijania ich. Czujniki nacisku na stopy pozwoliłyby na optymalizację chodu maszyny oraz wspinaczkę po schodach.

Bibliografia

[1] T. Zielińska. „*Maszyny kroczące: podstawy, projektowanie, sterowanie i wzorce biologiczne*.” Wydawnictwa Komunikacji i Łączności, Warszawa, 2014.

[2] A. Rygałło. Wykłady dla mechatroników z robotyki.: „*Robotyka dla mechatroników*”, dostępne w internecie: *http://www.planrozwoju.pcz.pl/wyklady/mechatronika/Robotyka.pdf,* [Data dostępu: 27 stycznia 2017].

[3] M. García-López, E. Gorrostieta-Hurtado. „*Kinematic analysis for trajectory generation in one leg of a hexapod robot*”, Iberoamerican Conference on Electronics Engineering and Computer Science, 2012.

[4] M. Piątek. „*Problemy sterowania robotami kroczącymi – generatory chodu hexapoda”,* AGH, Kraków, 2012.

[5] J. Rebula, P. Neuhaus. “*A Controller for the LittleDog Quadruped Walking on Rough Terrain*”, Florida Institute for Human and Machine Cognition, 06.2007.

[6] B.Jahne. „*Digital Image Processing*”, Wydawnictwo Springer, 2002.

[7] T. Kornuta, M. Stefańczyk. „*Akwizycja obrazów RGB-D: metody*”. Politechnika Warszawska, Warszawa, 20.12.2013.

[8] G.Bradski, A.Kaehler. „*Learning OpenCv3 Computer Visision in C++ with the OpenCv Library*” Wydawnictwo O`Reilly, 2017.

[9] Strona internetowa Adriana Rosebrocka „Pyimagesearch”: „*How to install OpenCV 3 on Raspbian Jessie*”, dostępna w internecie: *http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/*, [Data dostępu: 20 kwietnia 2017].

[10] Strona internetowa Asimo: „*Historia robotyki do roku 1969*”, dostępna w internecie: http://www.asimo.pl/historia/robotyka_kalendarium_1969.php, [Data dostępu: 1 lipca 2017].

[11] Strona internetowa Asimo, dostępna w internecie: http://www.asimo.pl/image/historia/Past/Honda-P3-1.gif, [Data dostępu: 1 lipca 2017].

[12] Strona internetowa Ciekawe.org: „*Historia robotyki*”, dostępna w internecie: http://ciekawe.org/2016/07/11/historia-robotyki/, [Data dostępu: 1 lipca 2017].

[13] Strona internetowa Codeblocks: „Dowwnloads”, dostępna w internecie: http://www.codeblocks.org/downloads, [Data dostępu: 15 kwietnia 2018].

[14] Strona internetowa Dailymail UK: „Meet Crabster, the robtic CRAB …", dostępna w internecie: http://www.dailymail.co.uk/sciencetech/article-2590545/Meet-Crabster-giant-robotic-CRAB-revolutionise-underwater-exploration.html [Data dostępu: 15 kwietnia 2018].

[15] Strona internetowa DIM-CAD: „*Abc skanowania 3d*", dostępna w internecie: http://www.dim-cad.pl/uslugi-skanowania-3d/abc-skanowania-3d/, [Data dostępu: 28 sierpnia 2017].

[16] Strona internetowa ElectronicsTechear, dostępna w internecie: *http://www.electronicsteacher.com/robotics/robotics-technology/sensors.php*, [Data dostępu: 25 marca 2017].

[17] Strona internetowa festo: „BionicANTs", dostępne w internecie: https://www.festo.com/group/en/cms/10631.htm, https://www.festo.com/group/en/cms/10157.htm, [Data dostępu: 1 lipca 2017].

[18] Strona internetowa Boston Dynamics: „*Atlas*", dostępna w internecie: *https://www.bostondynamics.com/atlas*, oraz strona internetowa Wikipedia: „*Atlas(robot)*", dostępna w internecie: *https://en.wikipedia.org/wiki/Atlas_(robot),* [Data dostępu: 1 lipca 2017].

[19] Strona internetowa firmy Pololu, dostępna w internecie: *https://www.pololu.com/product/1353*, program Pololu Maestro Control Center dla sterownika: *https://www.pololu.com/docs/0J40/3.a*, przykładowy kod: *https://www.pololu.com/docs/0J40/5.h.2*, [Data dostępu: 6 kwietnia 2017].

[20] Strona internetowa Github, dostępna w internecie: *www.github.com*, [Data dostępu: 20 kwietnia 2017].

[21] Strona internetowa Github z projektem grupy AVA, "*AVA RaspiCam: C++ API for using Raspberry camera with/without OpenCv*", dostępna w internecie: *https://github.com/cedricve/raspicam*, [Data dostępu: 20 kwietnia 2017].

[22] Strona internetowa Government Technology: „5 robots that may rescue you from natural disasters", dostępna w internecie: http://www.govtech.com/em/safety/5-Robots-That-May-Rescue-You-From-Natural-Disasters.html, [Data dostępu: 15 kwietnia 2018].

[23] Strona internetowa Kåre Halvorsen, projekt „Oxyopus 2008", dostępna w internecie: *http://zentasrobots.com/robot-projects/*, [Data dostępu: 27 stycznia 2017].

[24] Strona internetowa Maketecheasier: „*Enabling Remote Desktop Access with xrdp on a Raspberry PI*", dostępna w internecie: *https://www.maketecheasier.com/enabling-remote-desktop-access-on-raspberry-pi/*, [Data dostępu: 20 kwietnia 2017].

[25] Strona internetowa NASA Jet Propulsion Laboratory: „*NASA goes inside a volcano, monitors activity*”, dostępna w internecie: https://www.jpl.nasa.gov/news/news.php?release=2009-117, [Data dostępu: 15 kwietnia 2018].

[26] Strona internetowa National Instruments: *„ 3D Imaging with NI LabView*”, dostępna w internecie: http://www.ni.com/white-paper/14103/en/, [Data dostępu: 28 sierpnia 2017].

[27] Strona internetowa o chodach robotów: „*Robot Gait Intro*”, dostępna w internecie: *https://oscarliang.com/quadruped-robot-gait-study/,* [Data dostępu: 27 stycznia 2017].

[28] Strona internetowa o jednonożnym robocie balansującym 3D One-Leg Hopper, dostępna w internecie: http://www.asimo.pl/modele/3d-one-leg-robot.php, [Data dostępu: 27 stycznia 2017].

[29] Strona internetowa uczelni Mit Leg Laboratory: 3D One-Leg Hopper, Spring Flamingo, dostępna w internecie: *http://www.ai.mit.edu/projects/leglab/robots/robots.html,* [Data dostępu: 27 stycznia 2017].

[30] Strona internetowa producenta Raspberry Pi Foundation, dostępna w internecie: *https://www.raspberrypi.org*, zdjęcie: *https://www.raspberrypi.org/products/raspberry-pi-2-model-b/,* [Data dostępu: 6 kwietnia 2017].

[31] Strona internetowa Raspberry PI, dostępna w internecie: *http://ras-pi.de/wp-content/uploads/2014/09/raspberry-pi-camera-board-close-800x800-300x300.jpg,* [Data dostępu: 6 kwietnia 2017].

[32] Strona internetowa RapidTables: „*HSV to RGB converter”*, dostępna w internecine: *http://www.rapidtables.com/convert/color/hsv-to-rgb.htm,* [Data dostępu: 25 marca 2017].

[33] Strona internetowa Robotplatform. Dostępna w internecie: *http://www.robotplatform.com/knowledge/sensors/types_of_robot_sensors.html*, [Data dostępu: 25 marca 2017].

[34] Strona internetowa sklepu Botland, dostępna w internecie: *https://botland.com.pl/serwa-typu-standard/1142-serwo-hitec-hs-645mg.html,* [Data dostępu: 6 kwietnia 2017].

(sterownik serw) Strona internetowa sklepu Banana Robotics, dostępna w internecie: *https://www.bananarobotics.com/shop/Pololu-Mini-Maestro-24-Channel-Servo-Controller,* [Data dostępu: 6 kwietnia 2017].

(płytka) Strona internetowa forum Raspberry PI: „*Wiring diagram software* ”, dostępna w internecie: *https://raspberrypi.stackexchange.com/questions/42735/wiring-diagram-software,* [Data dostępu: 6 kwietnia 2017].

[35] Strona internetowa sklepu Robotshop, dostępna w internecie: *http://www.robotshop.com/en/lynxmotion-bh3-hexapod-robot-kit-hardware-only.html,* [Data dostępu: 6 kwietnia 2017].

[36] Strona internetowa z RoboCup: „RoboCupSoccer – Standard Platform", dostępna w internecie: http://www.robocup.org/leagues/5, [Data dostępu: 15 kwietnia 2018].

[37] Strona internetowa sklepu robotycznego RobotShop, produkt: Lynxmotion Phoenix 3DOF Hexapod - Black (No Servos / Electronics), dostępna w internecie: *http://www.robotshop.com/en/lynxmotiophoenix-3dof-hexapod---black-no-servos---electronics.html*, [Data dostępu: 27 stycznia 2017].

[38] Strona internetowa Sourceforge, dostępna w internecie: *https://sourceforge.net/projects/win32diskimager*, [Data dostępu: 20 kwietnia 2017].

[39] Strona internetowa Technical Recipes: „Configuring Code::Blocks to use OpenCV in Linux Environments", dostępna w internecie: *http://www.technical-recipes.com/2014/using-opencv-in-codeblocks-in-linux/*, [Data dostępu: 20 kwietnia 2017].

[40] Strona internetowa theGuardian: „Meet Z-Machines, squarepusher`s new robot band", dostępna w internecie: https://www.theguardian.com/music/2014/apr/04/squarepusher-z-machines-music-for-robots, [Data dostępu: 15 kwietnia 2018].

[41] Strona internetowa uczelni Shizuoka University: „*Time-of-Flight (ToF) Method*", dostępna w internecie: *http://www.idl.rie.shizuoka.ac.jp/study/project/tof/index_e.html*, [Data dostępu: 28 sierpnia 2017].

[42] Strona internetowa uczelni VirginiaTech: „*Color and Image Processing*", dostępna w internecie: *http://courses.cs.vt.edu/~cs4624/s98/sspace/imgproc/*, [Data dostępu: 25 marca 2017].

[43] Strona internetowa użytkownika Wingman, projekt „A Quadruped Robot", dostępna w internecie: *http://letsmakerobots.com/node/42518*, [Data dostępu: 27 stycznia 2017].

[44] Strona internetowa Wikipedia: „*Accelerometer*", dostępna w internecie: *https://en.wikipedia.org/wiki/Accelerometer*, [Data dostępu: 25 marca 2017].

[45] Strona internetowa Wikipedia: „*CIELAB color space top view.png*", dostępna w internecie: *https://en.wikipedia.org/wiki/File:CIELAB_color_space_top_view.png*, [Data dostępu: 25 marca 2017].

[46] Strona internetowa Wikipedia: „*Etyka robotów*", dostępna w internecie: https://pl.wikipedia.org/wiki/Etyka_robot%C3%B3w, [Data dostępu: 1 lipca 2017].

[47] Strona internetowa Wikipedia: „*Josef Capek*", dostępna w internecie: *https://pl.wikipedia.org/wiki/Josef_%C4%8Capek*, [Data dostępu: 25 marca 2017].

[48] Strona internetowa Wikipedia: „*Gaussian Blur*", dostępna w internecie: *https://en.wikipedia.org/wiki/Gaussian_blur*, [Data dostępu: 25 marca 2017].

[49] Strona internetowa Wikipedia: „*Gyroscope*", dostępna w internecie: *https://en.wikipedia.org/wiki/Gyroscope*, [Data dostępu: 25 marca 2017].

[50] Strona internetowa Wikipedia: „*HSV(grafika)", dostępna* w internecie: *https://pl.wikipedia.org/wiki/HSV_(grafika),* [Data dostępu: 25 marca 2017].

[51] Strona internetowa Wikipedia: „*HSL and HSV*", dostępna w internecie: https://en.wikipedia.org/wiki/HSL_and_HSV, [Data dostępu: 25 marzec 2017].

[52] Strona internetowa Wikipedia: „*Median Filter*", dostępna w internecie: *https://en.wikipedia.org/wiki/Median_filter*, [Data dostępu: 25 marca 2017].

[53] Strona internetowa Wikipedia: „*RGB color model*", dostępna w internecie: *https://en.wikipedia.org/wiki/RGB_color_model,* [Data dostępu: 25 marca 2017].

[54] Strona internetowa z dokumentacją OpenCV 3.0.0. Dostępna w internecie: *http://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html*, [Data dostępu: 25 marca 2017].

[55] Strona internetowa z dokumentacją OpenCV: „*Load and Display an Image*", dostępna w internecie: *http://docs.opencv.org/3.0beta/doc/tutorials/introduction/display_image/display_image.html*, [Data dostępu: 20 kwietnia 2017].

[56] Strona internetowa z poradnikiem do budowania robotów: „*Materials for robot building: an introduction*", dostępna w internecie: *http://www.robotoid.com/howto/materials-for-robot-building-an-introduction.html,* [Data dostępu: 27 stycznia 2017].

[57] Strona internetowa WonderHowTo: „This DIY Walking Paper …", dostępna w internecie: https://paper-design.wonderhowto.com/news/diy-walking-paper-robot-shoots-rubber-bands-from-its-high-powered-gatling-gun-arms-0138944/, [Data dostępu: 15 kwietnia 2018].

[58] Strona internetowa YouTube: „Quadruped robot 4DIF", dostępna w internecie: https://www.youtube.com/watch?v=RgV6BoCdTcc, [Data dostępu: 15 kwietnia 2018].

[59] Strona internetowa sklepu EngineerDir, produkt „Carl's Electronics Gakkan Mechamo Centipede Robot", dostępna w internecie: *http://www.engineerdir.com/product/catalog/13609/index1.html,* [Data dostępu: 27 stycznia 2017].

[60] Strona internetowa Uniwersytetu de Aveiro, dostępna w internecie: http://lars.mec.ua.pt/public/LAR%20Projects/RobotActuation/2002_MarcoMelo_VascoQuinteiro/Projecto/Artigos_net/catalogo/walking_machines_katalog/node16.html, [Data dostępu: 1 lipca 2017].