Namn: Kai Nguyen
Klass: TE20B
Skola: Rudbeckianska gymnasiet
Datum: 2023-03-29
Handledare: Peter Werner

Design och kinematisk styrning av en fyrbent robot

Kai Nguyen TE20B
2023-03-29

Abstract

Some of the recent developments in robotics has been focused on mobile and agile robots for various applications such as industrial use and space exploration. Several variants of mobile robots come in different forms and has advantages and disadvantages over one another. One variant of such robot is the quadruped robot which gives more flexibility and adaptability when traveling on complex terrains compared to a traditional wheeled robot. It has therefore garnered attention and has the potential to improve efficiency in many applications. In this paper, research is performed on the history of quadruped robots and the different approaches to designing a prototype quadruped robot using low-cost material such as cardboard, creating the circuitry for the robot with different low-cost electronics and mathematically program its basic functions such as walking and balancing on uneven terrain using inverse kinematics and trigonometry on an Arduino UNO microcontroller. The low-cost and simplistic approach to designing a quadruped robot not only increases the access to a quadruped robot at a lower price point but can also be used as educational material for robotics students with limited knowledge.

Kai Nguyen TE20B
2023-03-29

Innehållsförteckning

Kai Nguyen TE20B
2023-03-29

1. Inledning

1.1 Varför detta ämne?

Utvecklingen av infrastruktur har exploderats under de senaste decennier och som resultat ökades behovet av kompetenta ingenjörer. Men inför framtiden ligger ett problem för denna utveckling, ju mer komplexa blir infrastrukturerna desto mer belastas ingenjörerna. Det finns alltså en mänsklig begränsning för utvecklingen av infrastruktur. Ett stort problem hos många industriella områden är arbete på farliga miljöer vilket ökar risken för skador för människor.

Från dessa problem utvecklades robotar i olika former anpassad till problemet. Några exempel är robotar på hjul som är bra på navigation och stationära robotar med axlar som förflyttar föremål i känsliga industrimiljöer. Under de senaste åren har intressen för en annan typ av robot ökat, bland annat robotar på ben. Robotar på ben är främst anpassad för navigation och är mycket mer kapabel än robotar på hjul i vissa miljöer. På ojämna terränger är robotar på hjul inte tillräcklig stabila, medan robotar på ben kan anpassa sina benaxlar till en högre grad av frihet för att åstadkomma stabilitet, liknande en levande organism på ben. Detta innebär att robotar på ben är en bättre tillämpning i farliga industriella miljöer med komplexa terränger som till exempel gruvor. Av denna anledning att robotar på ben har många möjligheter väckte frågor om vad som ligger till grund bakom deras mekanism och hur flexibel kan man göra robotarna med mekanismerna.

1.2 Syfte

Syftet för detta gymnasiearbete är att undersöka och utveckla mekanism hos en variant av robotar på ben, nämligen fyrbent robotar. Mekanismen innefattar robotens rörelsefunktion, självbalansering av roboten, design och elektronik. Målet är att kunna bygga en låg kostnad prototyp med fungerande programmering som kan överföras till andra liknande modeller ifall prototypen inte fungerar.

1.3 Frågeställningar

Kring arbetet ligger några frågeställningar som ligger till grund för hur roboten och dess mekanismer ska vara uppbyggd samt värdering av robotens effektivitet.

- Hur ska mekanismen för robotens rörelser vara uppbyggd? Kan rörelserna simuleras matematiskt?
- Vilken typ av motor passar till roboten med avseende på budgeten? Kommer den att vara tillräckligt snabb och stabil?
- Vilka elektronik ska roboten vara bestå av för att vara låg kostnad?
- Hur ska benen vara uppbyggd för den bästa stabilitet och effektivitet?
- Hur ska roboten känna av när dess kropp inte är parallell mot marken och hur ska dem olika benen anpassa sig efter informationen så att roboten stabiliseras?

2. Bakgrund

2.1 Historik

De tidigaste varianter på fyrbent robotar uppkom vid sent 1800-talet enligt Biswal (2019) i sin uppsats. Dessa varianter drivs på mekanisk kraft som omvandlar rotationsrörelse från en vev till en rörelse som för roboten framåt eller bakåt beroende på vevens rotationsriktning. Mekanismen kan jämföras med en cykel som omvandlar rotationsrörelse från pedalerna till en rörelse som för cykeln framåt.

Biswal visar ett exempel på en mekanisk häst som drivs på samma mekanism. Rörelsemekanism är uppbyggd av flera länkar som tillsammans utför rörelse på benen utifrån rotationsrörelsen för en kugghjul som sitter i mitten av hästen. Nackdelen med mekanismen är att den kräver att en operatör ska kontinuerlig utföra arbete genom att snurra på kugghjulet för att hästen ska gå framåt, samt att rörelsen inte är dynamisk och känner inte av hur den ska anpassa sig till omgivningen. Det innebär att hästen blir ostabil vid ojämna terränger.

Den första fyrbent robot som hade elektriska ställdon och som kunde kontrolleras av en dator uppkom i 1960-talet vid University of Southern California och döptes ”phony pony” enligt Biswal. Varje ben på roboten består av två elektriska ställdon som liknar motorer och som sätter benen i rörelse vid elektriska signaler. Ställdonens placering delar varje ben i två delar, lår och underben, precis som i levande organismer vilket gör att benen får en högre grad av rörelsefrihet. Eftersom roboten kontrolleras genom datorn kan olika program för robotens rörelse skrivas och upprepas i en loop vilket gör att roboten kan fortsätta sin färd om inte programmet avbryts. Nackdelen med denna variant är att den inte kan gå sidled eller byta sin egen riktning. Det skulle kräva att benen har en tredje ställdon som för benen i sidled riktning eftersom de två elektriska ställdon som redan finns på benen är placerade på ett visst sätt att dem endast kan gå framåt eller bakåt relativ till robotens orientering.

Vidare förklarar Biswal att numera har moderna fyrbent robotar vänt sig till användning av hydraulik och elektriska motorer för att driva robotens benaxlar. Hydrauliksystemet i en fyrbent robot kopplades till benens axlar och kunde kontrolleras genom datorn. Varje hydraulpump i systemet har sin egen funktion när hydraulikvätskan går igenom pumpen vid aktivering från en dator. Som till exempel skulle en pump kunna vara ansvarig för att föra underbenen uppåt, samtidigt som en annan pump ansvarar för att föra hela benen i sidled riktning. I kombination av dessa pumpar kan det datorkontrollerade hydrauliksystemet utföra komplexa rörelser som är nästan jämförbar med ett system som endast består av elektriska komponenter. Biswal nämner i sin uppsats en av de mest komplexa fyrbent robot som använde sig av hydraulik och utvecklades av företagen Boston Dynamics med namnet BigDog.

De senaste fyrbent robotar använder sig av elektriska motorer. Enligt York Precision Machining and Hydraulics (2019), en fördel med hydraulik över elektriska motorer är att den ger större kraft vilket förstärker benen och ger bättre stabilitet. För att producera stora mängder krafter från elektriska motorer krävs stora mängder ström och är inte effektivt. Men numera har elektriska motorer blivit tillräckligt effektiva för att kunna ersätta hydraulik i många modeller. Elektriska motorer ger bättre noggrannhet, minskar komplexiteten av

systemet och har ingen vätska som kan leda till läckage vilket kan förekomma i hydrauliska system. Samma företag som utvecklade BigDog, Boston Dynamics, har utvecklat en mindre version av BigDog som drivs på elektriska motorer och är mer kapabel är BigDog när det gäller flexibilitet och noggrannhet vid navigation i komplexa terränger och artificiell intelligens. Roboten döptes ”Spot”. Det finns möjlighet för installation av olika typer av industriella sensorer på roboten vilket gör Spot mycket användbar på industriella miljöer.

2.2 Motor

Robotens viktigaste komponent är en motor som ser till att roboten sätts i rörelse. Det finns olika typer av motorer som är olika bra vid olika tillämpningar. De vanligaste typer av motorer som går att hitta i robotar på ben är en stepper motor, brushless dc motor och i enklare och billigare robotar, servomotor.

En stepper motor och en brushless dc motor fungerar på samma princip. Inuti motorerna sitter koppartrådar runt en kärna. Kärnan är omringad med en permanentmagnet med två poler. En video från Owen J. (2020) som beskriver funktionen av en dc motor visar att när ström tillförs till koppartrådarna skapas elektromagnetiskt fält som får kärnan att snurra genom att fältet från permanentmagneten repellerar det elektromagnetiska fältet från koppartrådarna. I vissa motorer är permanentmagneten omringad av en kärna med koppartrådar.

En brushless DC motor är starkare och mer energieffektiv än en stepper motor på grund av bättre komponent samt skillnad i strukturen inuti motorn förklarar Budimir (2023, 24 mars) i sitt blogginlägg. Budimir tillägger att en stepper motor har fördelen att den är billigare och används därför i enklare system. För exakt positionering av både brushless dc motor och stepper motor krävs ofta en extern komponent som beräknar positioneringen. Positionering av motorn är viktig för roboten eftersom den kräver exakt positionering av benaxlarna för att kunna sätta benen till vissa lägen och rörelser. Att behöva köpa en extra komponent till motorn ökar kostnaden för roboten.

Servomotor till skillnad från brushless DC motor och stepper motor har en inbyggd komponent som talar för datorn vilken position motorns huvud ligger i vilket anges i antingen radianer eller grader. Servomotorn består av en DC motor som snurrar på växlar ovanpå motorn, samt en potentiometer och en mikrokontroller. I en video från Paul Evans (2022) illustreras en servomotor och dess komponenter under processen av kommunikation mellan motorn och en dator. Enligt Evans i sin video tar mikrokontrollern emot signaler från datorn som berättar för DC motorn hur mycket den behöver snurra. DC motorns mekanism är ungefär samma som en stepper motor och brushless DC motor där kärnan snurras med hjälp av magneter. DC motorns huvud är kopplad till en växel som snurrar större växlar. Den sista växel i kedjan snurrar huvudet av servomotorn. Till huvudet av en servomotor kan olika tillbehör installeras som ett horn. Hornet hjälper till att visa positioneringen av motorn i verkligheten utanför datorn.

Evans vidare förklarar att positioneringen av motorn kan beräknas med hjälp av en potentiometer. En potentiometer är en komponent som ändrar spänningen i en krets när den

snurras. I servomotorn snurras potentiometern samtidigt som huvuden snurras. Det innebär att varje position för motorn motsvarar en specifik ändring i spänning eftersom potentiometern snurrar när motorn snurrar. Evans utvecklar med att genom att datorn skickar en signal med en viss spänning som skiljer sig från potentiometerns egen spänning, kommer skillnaden mellan spänningen från signalen och potentiometern att generera rörelse för motorn. Ett exempel är att motorn sitter i 60 grader vilket motsvarar 3V och datorn ska skicka en instruktion till motorn som säger att den ska flyttas till 30 grader som motsvarar 2.6V. Skillnaden mellan potentiometerns spänning och signalens spänning blir 0.4V vilket gör att motorn snurrar ner till 30 grader. Samtidigt som motorn snurrar ner kommer potentiometern att snurra ner och minska sin egen spänning från 3V till 2.6V. När potentiometern har nått 2.6V försvinner skillnaden mellan signalen och potentiometern och motorn slutar snurrar. Då står motorn stilla vid 30 grader.

Signalen som skickas till motorn liknar pulser enligt Evans. Signalen skickas genom en metod som heter Pulsbreddsmodulering eller PWM, vilket innebär att spänningen som skickas över signalen slås på och av snabbare än vad datorn kan urskilja. Detta gör att signalerna kommer att se ut som pulser.

Evans förklarar i videon att varje puls motsvarar en signal eller spänning. Pulserna har olika bredd som talar för hur länge signalerna är på. Pulserna kan också förekomma i olika höjder som talar för hur mycket spänning som påläggs över signalerna. En puls med längre bredd som skickas till motorn kommer att snurra motorn under en längre tid. Detta innebär att för att kunna förflytta motorns huvud med till exempel 5 grader, så behöver datorn endast skicka en smal puls. Om motorn behöver förflyttas med 40 grader kommer pulsen att behöva vara tjockare. Genom att utnyttja principerna för PWM kan servomotorer enkelt programmeras.

Eftersom en servomotor kan positioneras med noggrannhet utan att den behöver extra komponenter, kommer den att vara en lämplig val för projektets egen fyrbenta robot. Nackdelarna med en servomotor är att den är inte lika kraftig och snabb, och har en mindre livslängd än en brushless DC motor och stepper motor. För projektets krav vilket är att roboten ska vara låg kostnad och är endast en prototyp, kommer en servomotor att vara tillräckligt lämplig.

2.3 Balansering av roboten

En av robotens viktigaste komponent som ansvarar för att roboten ska stå upprätt i ojämna terränger och även när roboten knuffas av en ytterkraft är en IMU sensor. IMU är en förkortning för Inertial Measurement Unit och är en elektronisk enhet som mäter i detta fall robotens acceleration och vinkelhastighet som ger robotens orientering enligt Or Barak (2022) i sitt blogginlägg. I vanligaste fall är IMU en kombination av en accelerometer som mäter accelerationen och ett gyroskop som mäter vinkelhastigheten.

IMU sensorn kan mäta accelerationen i X-, Y- och Z-led. Detta är mycket användbar som till exempel för att visa om roboten accelererar för mycket eller för litet i en viss riktning för att

sedan justera robotens hastighet eller bibehåll balans. IMU sensorn kan mäta vinkelhastigheten i X-, Y- och Z-led. Vinkelhastigheten från sensorn talar för datorn hur mycket roboten lutar relativ till utgångspunkten, vilket i många fall är marken. Den ideala orienteringen för roboten är när lutningsvinkel mellan roboten och marken är noll grader i vanlig icke-lutande mark vilket ger störst stabilitet. På ojämna terränger kan marken vara lutande och den ideala orienteringen kan i dessa fall anpassa sig beroende på hur mycket stabilitet lutningsvinkeln ger.

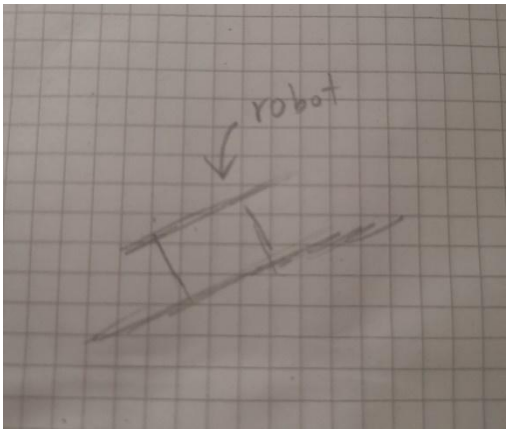

Fig. 1. Robot faller på lutande plan

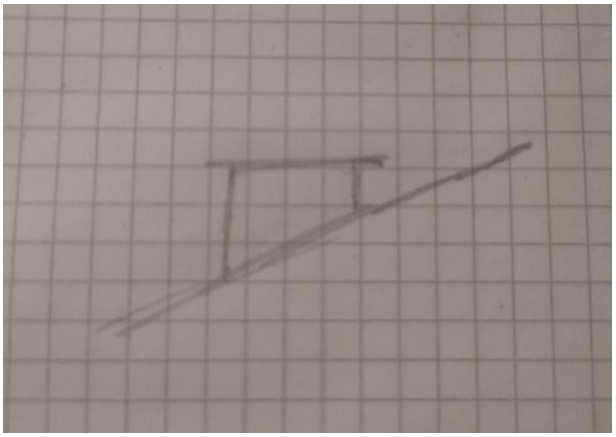

Fig. 2. Robot balanseras på lutande plan

Fig. 1. visar en fyrbent robot framifrån som är stående på ett lutande plan. I denna bild är robotens kropp vinkelrät mot normalen av planet. Detta ger inte stabilitet i alla situationer eftersom roboten lutar för mycket åt ett håll. Fig. 2. visar hur roboten kan uppnå bättre stabilitet genom att låta kroppen vara vinkelrät mot gravitationskraften som pekar rakt neråt. Roboten behöver då justera sina fyra ben till olika längder för att hålla kroppen vinkelrät mot gravitationskraften på en lutande plan.

2.4 PID

Styrkrets är en viktig funktion i en robot som behöver förhålla sig till ett idealt läge under alla situationer. Den är bland annat viktig för att kunna hålla roboten i balans med hjälp av en IMU sensor. Styrkrets är ett system som reglerar data och ser till att data förhåller sig till ett önskat värde (”PID”, 2023, 4 februari). I balans sammanhang är robotens nuvarande lutningsvinkel en data och det önskade värdet skulle kunna vara en lutningsvinkel som gör att

robotens kropp blir vinkelrät mot gravitationskraften. Så fort lutningsvinkel för roboten lämnar det önskade värdet, vilket kan hända när den till exempel faller eller lutar, så kommer styrkretsen att se till att roboten återgår tillbaka till den önskade lutningsvinkel genom att ge instruktioner till robotens ben så att roboten återigen är i balans.

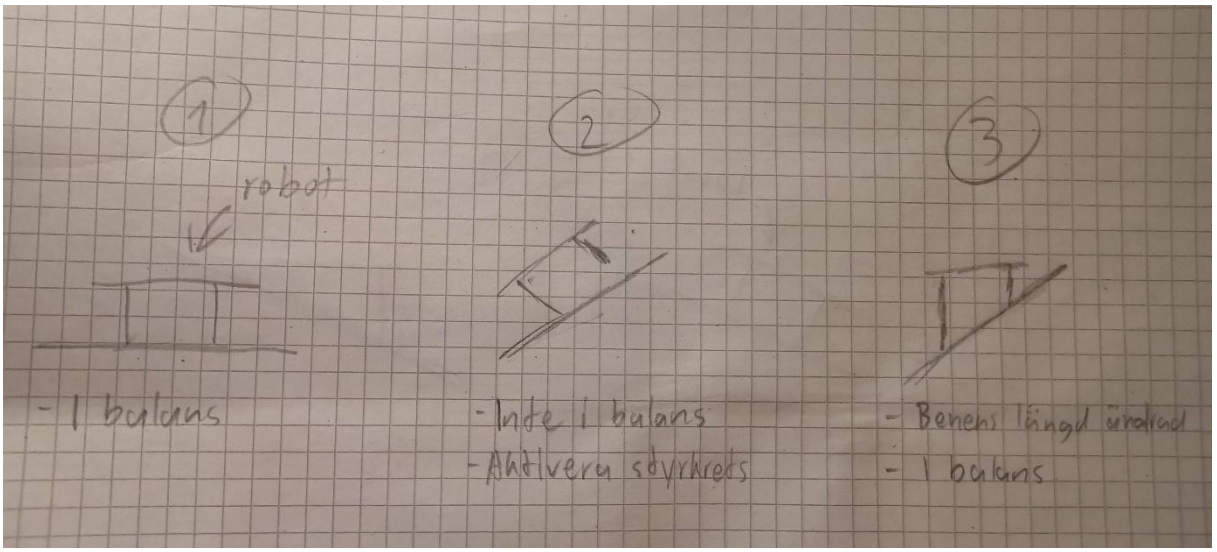

Fig. 3. Balans process för robot

Fig. 3 visar hur processen för en styrkrets i en fyrbent robot skulle kunna hjälpa till balanseringen av roboten. Vid läge 1 är roboten i balans eftersom den är vinkelrät mot gravitationskraften. Vid läge 2 är roboten inte i balans eftersom den inte är vinkelrät mot gravitationskraften och håller på att accelerera nedåt. Datorn aktiverar styrkretsen för att kontrollera benen. Vid läge 3 har styrkretsen ändrat benens längd vilket gör att kroppen återgår till att vara vinkelrät mot gravitationskraften. Styrkretsen kan jämföras med människans hjärna som ser till att kroppen inte faller genom att justera benen för att hålla kroppen upprätt.

PID är en variant av styrkrets och står för Proportional Integral Derivative. En PID styrkrets kan matematisk beräkna hur ett system ska ändras för att förhålla sig till ett önskat värde genom en formel. PID styrkretsen kommer att exekveras i en loop tills systemet har nått det önskade värdet. I en video från MATLAB av Brian Douglas (2018) beskrivs den matematiska processen för att bygga upp en PID styrkets. Den matematiska formeln som beskriver PID ses enligt formeln nedan.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Formeln består av tre termer där den första termen motsvarar den proportionella delen av ekvationen, den andra termen motsvarar integral delen av ekvationen och tredje termen motsvarar derivata delen av ekvationen. Gemensam för alla tre termer är att de innehåller funktionen $e(t)$. $e(t)$ motsvarar i första loopen av PID felet från det nuvarande värdet och

det önskade värdet i systemet. För roboten är $e(t)$ skillnaden mellan den nuvarande lutningsvinkel och den önskade lutningsvinkel som ger vinkelräthet mot gravitationskraften. Efter varje loop kommer $e(t)$ att subtrahera sig själv från det föregående felvärdet som gör att felvärdet minskar. $e(t)$ slutar minskas när felvärdet är noll.

Teorin bakom PID utgår från att genom att summera den proportionella delen av felvärdet, integralen av felvärdet och derivatan av felvärdet så kommer styrkretsen att få ett slutvärde, $u(t)$, som kan användas för att kontrollera systemet för att återgå till det önskade värdet. Ett exempel med en robot är när roboten lutar för mycket åt ett håll och håller på att falla och att robotens lutningsvinkel i detta moment är 20 grader mot gravitationskraften. Det önskade värde som ger bäst stabilitet ska vara noll grader mot gravitationskraften. Felvärdet, $e(t)$, i detta moment är 20 grader eftersom skillnaden mellan den nuvarande lutningsvinkel 20 grader och den önskade lutningsvinkel noll grader är lika med 20 grader.

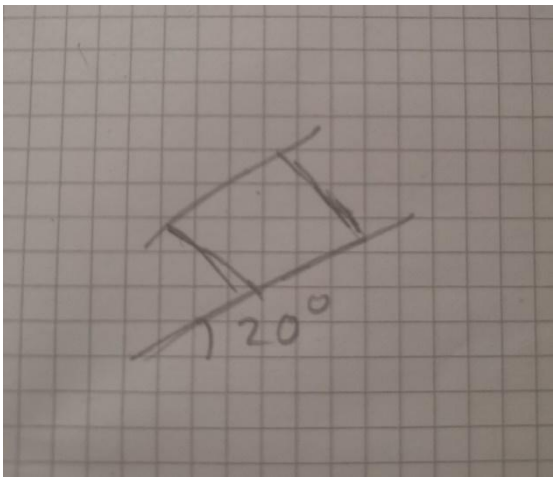

Fig. 4. Roboten på 20 graders vinkel lutande plan

Douglas i videon förklarar att den proportionella delen av PID formeln kommer att beräkna ut hur snabb ett system, eller i detta fall en robot, ska återgå till det önskade värdet per en given tidsintervall. Anta att denna tidsintervall är 50 millisekund och lutningsvinkel ska gå tillbaka till noll med en hastighet som är 1 grader per given tidsintervall som i detta fall är 50 millisekund. Det innebär att det skulle ta 1 sekund för roboten att återgå till vinkel noll. Förklaringen är att för varje 50 millisekund återgår systemet tillbaka till vinkel noll med 1 grader i taget. Detta ska ske 20 gånger för att roboten ska återgår till vinkel noll eftersom felvärdet är 20 grader. 50 millisekunder multipliceras med 20 gånger ger 1 sekund. Hur snabb vinkeln återgår tillbaka till det önskade värdet beror på konstanten K_p i formeln. Med en hastighet av 1 grader per intervall ger att $K_p = 1$. Vid större hastighet kommer denna konstant att ökas. Konstanten är också beroende av tidsintervallet som tillsammans med konstanten avgör hur snabb vinkeln ska nå det önskade värdet.

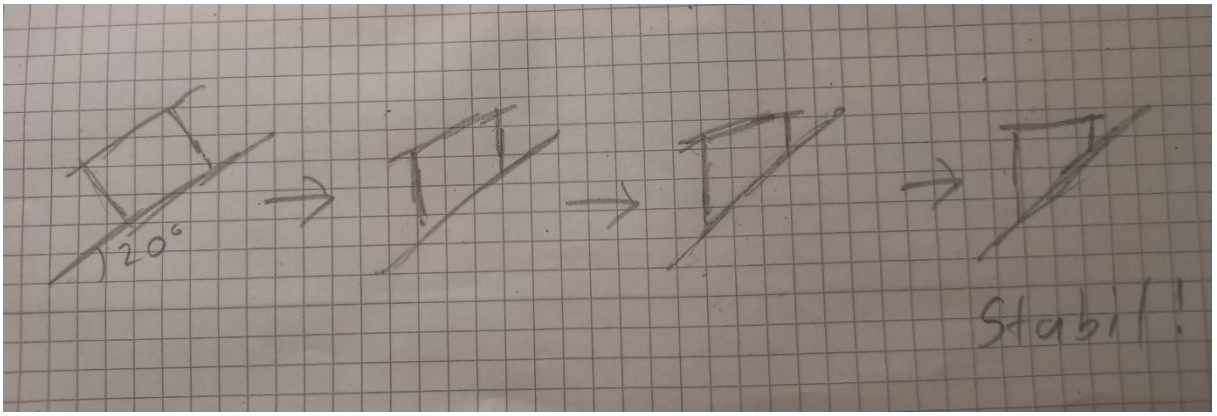

Fig. 5. Balans process för robot

Hela processen skulle kunna se ut enligt Fig. 5. Efter den första intervallen har felvärdet minskat från 20 grader till 19 grader. Processen upprepas och roboten under en sekunds tid återgår från 20 grader lutningsvinkel till noll grader som ger vinkelräthet mot gravitationskraften. Under processen har robotens olika ben ändrat längd för att nå slutresultatet. Denna längd är alltså påverkade av PID styrkretsen utifrån hur mycket roboten lutar.

Douglas tillägger att i verkligheten kommer denna typ styrkrets med endast proportionalitet delen att ha problem för felvärde som är mycket litet. Anta att felvärdet är 0,001 grader vilket innebär att roboten är nästan vinkelrät mot gravitationskraften. Eftersom felvärdet är inte noll kommer styrkretsen att fortsätta försöka få roboten till vinkelräthet. Men som resultat kommer felvärdet att inte bli noll, den kommer bara närmare noll men är aldrig noll eftersom styrkretsen rör robotens ben för långsamt vid små felvärden. Som till exempel när felvärdet ha blivit 0,00000001 kommer vinkeln att gå för långsamt med en hastighet av 0,00000001 grader per intervall för att kunna minska ner till noll. Därför används integraler i styrkretsen för att undvika detta. Integral delen i PID formeln ger summationen av alla tidigare felvärden. Så länge felet inte är noll kommer summationen att växa. Resultat blir att slutvärdet $u(t)$ kommer att få ett högre värde och därmed högre hastighet för vinkeln även om proportionalitet delen är mycket litet eftersom integralen fortsätter att växa och summerar alla tidigare fel med det nuvarande felet. Då kommer roboten att nå till noll vinkeln snabbare om istället styrkretsen endast bestod av en proportionalitet del.

I många fall kan roboten lutar sig för mycket och styrkretsen kommer att försöka återgå till noll vinkeln för snabb. Detta kan leda till att felvärdet minskar för snabb och blir negativ eller går över gränsen. Douglas förklarar i videon att med hjälp av derivata delen kan liknande problem undvikas. Derivata delen i PID formeln representerar hur snabb felvärdet minskar. Vid snabb minskning av felvärdet blir derivatan både stor och negativ. När $u(t)$ blir för stort kommer felvärdet att minska för snabbt. Det negativa värdet av derivatan kan subtraheras från proportionella och integral delen för att minska slutvärdet $u(t)$ som undvik från att göra den för stort. Resultatet blir att ju närmare och snabbare roboten kommer till noll vinkeln desto saktare minskar felvärdet.

Med kombinationen av proportionalitet, integral, och derivata kan en lämplig styrkrets skapas för balansering av roboten vid olika situationer. Därmed har PID styrkretsen varit populärt hos robotar som behöver balans.

2.5 Rörelse

PID styrkretsen är oanvändbar om inte det går att noggrant kontrollera robotens olika ben och berätta till benen vilken längd dem behöver vara. Själva roboten behöver också ett sätt för att röra benen för att till exempel röra framåt och bakåt. Rörelse är den viktigaste funktionen i en robot som är skapat för navigation. Hur rörelsefunktionen är uppbyggd kommer att vara beroende på robotens uppbyggnad vilket kan innehålla faktorer som typen av motor, antalet motorer på roboten, design av benen och även kroppsvikt. Därför måste en design först väljas för att kunna sedan utveckla rörelsefunktionen som är designspecifik. Roboten Spot av företaget Boston Dynamics är en effektiv och komplex robot som har en enkel design. Det mesta av komplexiteten kommer från mjukvaran som roboten använder. Denna design kommer att vara grunden till projektets egen fyrbenta robot.

Benstrukturen på roboten är en avgörande faktor för hur rörelse ska programmeras för roboten. För roboten Spot har varje ben tre motorer som kontrollerar tre leder på benen, en vid knän, en på toppen av låren och en bakom låren vid höften. Leden vid knän ansvarar för rörelse av underbenen, leden vid toppen av låren ansvarar för rörelse av låren och leden vid höfterna ansvarar för rörelse av hela benen sidled. Leden vid knän kan föra underbenen uppåt eller neråt och leden vid toppen av låren kan föra låren framåt eller bakåt. I kombination av alla tre leder kan roboten utföra komplexa rörelser på varje ben. Hur benen ska positionera sig vid en specifik rörelse är beroende på vilka instruktioner datorn har gett till motorn. Själva motorn som för detta projekt är en servomotor, har en enkel rörelse eftersom den behöver endast snurra. Men datorn kan berätta för motorn åt vilket håll den ska snurra, samt vilken vinkel motorn ska positionera sig i. Information om motorns vinkelposition är mycket användbar för programmeringen av rörelsen.

För att kunna ge exakta instruktioner för hur motorn ska vara positionerade för att benen ska kunna uppnå ett idealt läge används en process i robotik som kallas för inverterad kinematik. Inverterad kinematik är en matematisk process som beräknar positioneringen för motorn i till exempel en robotarm så att armens ände når en viss koordinat (MathWorks, u.å.). Ett exempel är en robotarm bestående av 5 axlar. För varje axel sitter en motor som kan snurra till en viss vinkel. Positioneringen för armens ände som innehåller koordinat (x, y) är beroende av vinkelpositionerna q1, q2, q3, q4, och θ. Genom att använda principerna för inverterad kinematik kan datorn räkna ut vilka vinklar de olika axlarna ska positionera sig för att armens ände ska nå en given punkt i koordinatsystemet. Motsatsen till inverterad kinematik är Forward Kinematics som räknar ut ändens position i koordinatsystemet utifrån axlarnas vinkelposition. Med inverterad kinematik arbetar datorn med en given ändpunkt medan Forward Kinematics arbetar med givna vinkelpositioner för axlarna för att sedan räkna ut ändpunkten. För en fyrbent robot är inverterad kinematik optimalt eftersom roboten behöver

först veta vart benen ska gå innan den kan ge instruktioner till hur motorerna ska vara positionerade.

Den matematiska processen för inverterad kinematik kommer att vara olika för olika robot beroende på hur många axlar roboten har. Principerna är tills mestadels samma för hur axelpositionen ska beräknas. De två vanligaste sättet för beräkning av axlarna är trigonometri och linjär algebra. I vissa system kommer båda att användas tillsammans medan andra kommer att använda ett av de. Beräkningarna kommer att behandlas under Metod och genomförande.

3. Metod och genomförande

3.1 Inverterad kinematik

Rörelse är den viktigaste funktionen för roboten och behandlas därför innan andra steg i genomförandet. Tillämpning av inverterad kinematik är centralt i detta avsnitt. En enkel algoritm för inverskinematik på en fyrbent robot togs fram av Nic Aqueous (2021) i en video där han dokumenterar processen för byggandet av hans egen fyrbent robot.

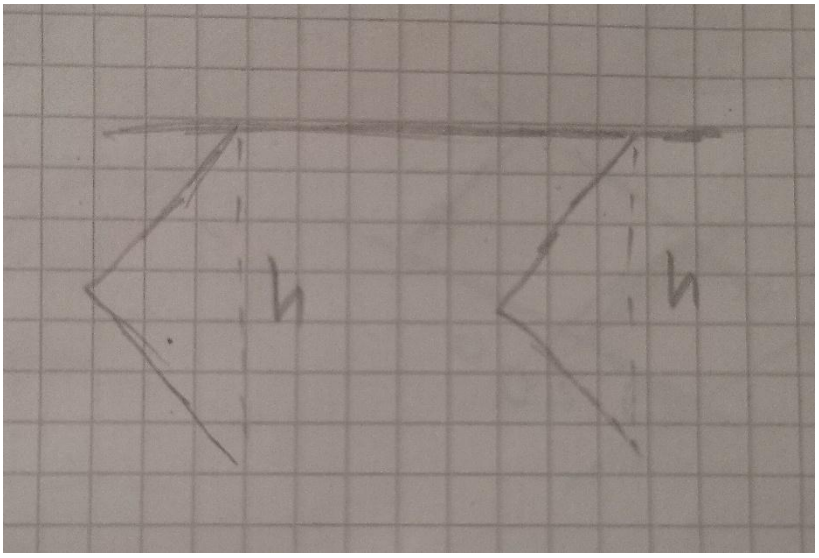

Fig. 6. Roboten från sidan

Fig. 6. visar en illustration för roboten från sidan. Aqueous förklarar att robotens ben bildar en triangel om en extra linje ritades. Denna linje är även höjden för robotens kropp över marken bortsett från kroppens dimensioner och kan kallas för h. Genom att linjen h ändras så kan benen justera sin längd och ändra både benens läge och hela kroppens läge. Om till exempel roboten ska stiga sin kropp så kommer linjen h att vara längre. Programmet för roboten ska alltså kunna mata in olika värden för linje h för att få en viss rörelse för benen som ska i sin tur motsvara en rörelse för hela roboten.

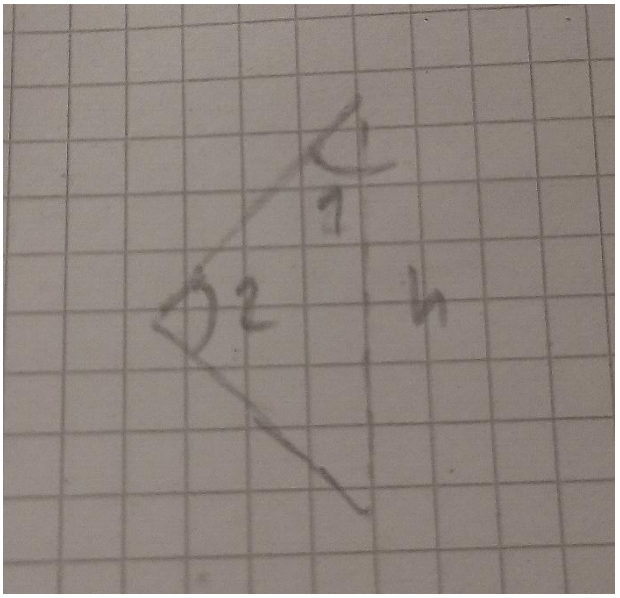

Fig. 7. Benens triangel

Triangeln bildar även vinklar som är användbara för beräkningen av rörelsen. Dessa vinklar är själva positionen för motorerna där vinkel 1 är vinkeln för motorn kopplad till låren och vinkel 2 är vinkeln för motorn kopplad till underbenen. Vinklarnas värde är beroende av storleken för linje h. För att räkna ut vinklarnas värde som är beroende av storleken för linje h används cosinussatsen enligt Aqueous. Cosinussatsen är en matematisk regel som beskriver förhållandet mellan en triangels vinklar och sidor.

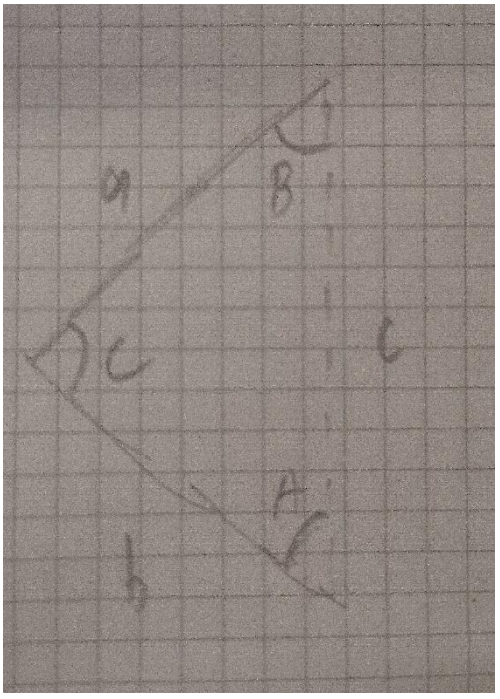

Fig. 8. Triangel

Fig. 8 visar en triangel som är lik den triangeln som är formad från robotens ben. Triangeln har tre sidor och tre vinklar. Sidan c ska motsvara linje h för roboten. Programmet för roboten

behöver utgå från sidan c för att räkna ut värdet för vinkel C och B som ska används till motorernas positionering. Enligt cosinussatsen är formeln för sambandet mellan sidan c och vinkel C:

$$c^2 = a^2 + b^2 - 2ab\,cos(C)$$

Lösningen för vinkel C ur formeln kommer att se ut på detta sätt:

$$C = \left(\frac{a^2+b^2-c^2}{2ab}\right)$$

Detta ger formeln för vinkel C. På samma sätt kan cosinussatsen används till vinkel B. Lösning för vinkel B kommer att se ut som följande:

$$B = \left(\frac{a^2+c^2-b^2}{2ac}\right)$$

Sidorna a och b är kända eftersom de utgör benens längd. Sidan c kommer vara ett bestämt tal som matas in i den inversa cosinusfunktionen för att få ut ett vinkelvärde för motorerna. Till exempel om robotens kropp ska vara 20 cm över marken så kommer sidan c att vara 20. Talet 20 används sedan i funktionen för vinklarna. Resultatet blir att motorerna har fått vissa värden, ändrat sin position från dessa värden och får till slut robotens kropp att vara 20 cm över marken.

Med formlerna för vinkel C och B kan nästa steg utföras vilket är att bearbeta enkla rörelser för roboten. För enkelhetsskull behandlar roboten endast framåt rörelser för tillfället. Processen skulle vara samma om benen hade gått bakåt.

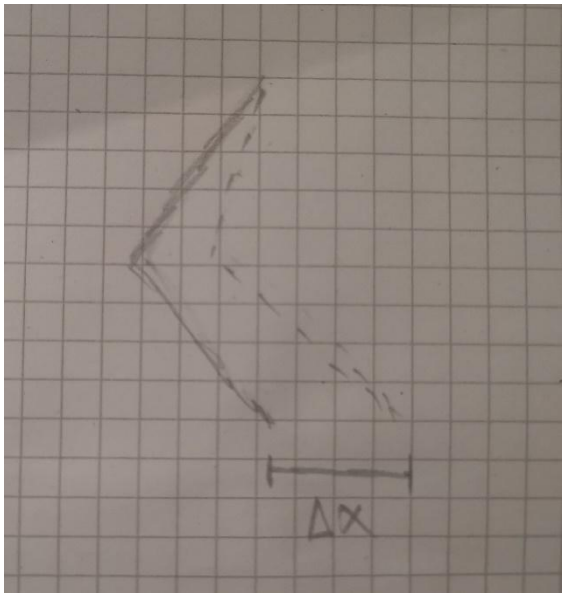

Fig. 9. Benen går framåt med delta x

För att benen ska gå framåt behöver den att flytta sig framåt med en viss steglängd. Steglängden kan kallas för x eller delta x som Fig. 9 visar. Benen vid steglängden x från startpositionen har en linje h som är längre än linje h vid startpositionen. Genom att använda denna nya linje h kan positionen för axlarna beräknas för benen vid steglängden x.

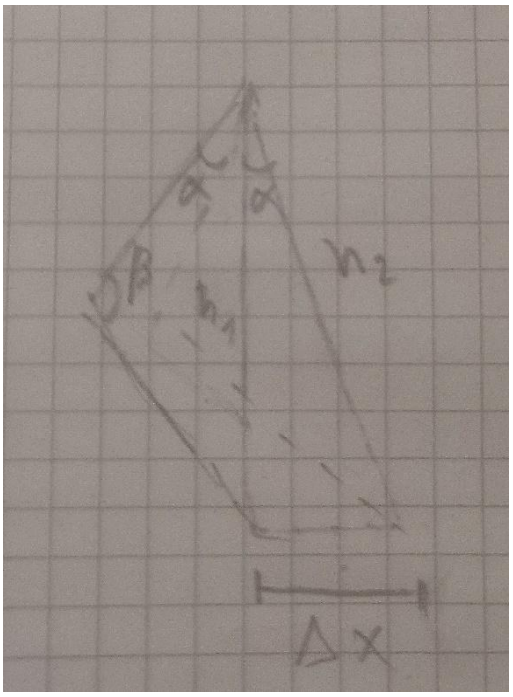

Fig. 10. Trigonometrin för rörelsen

Aqueous visar i sin video att om en rätvinklig triangel ritades så går det att hitta sambandet mellan den nya linjen h vid steglängden x och vinklarna. För tydlighet döpts den nya linjen h till h_2 och linjen h för startpositionen till h_1. Först och främst kan det konstateras att topaxeln för benen har rört sig framåt med vinkel α enligt bilden. Denna vinkel är även samma för topvinkeln för den rätvinkliga triangeln. Då kan vinkel α beräknas enligt Pythagoras sats på följande sätt.

$$\alpha = \left(\frac{\Delta x}{h_1}\right)$$

Denna vinkel är alltså vinkelpositionen för motorn som är kopplad till låren. Vinkelpositionen för motorn som är kopplad till underbenen kan kallas för vinkel β enligt bilden. Denna vinkel har blivit större vid steglängden x. Vinkel β kan enkelt räknas genom att använda cosinussatsen som i föregående exempel.

$$\beta = \left(\frac{a^2+b^2-h_2{}^2}{2ab}\right)$$

Variablerna a och b i formel är längden för låren respektive underbenen.

3.2 Cirkulär rörelse

Nu är det möjligt att kunna föra benen framåt enligt beräkningarna ovan. Ett problem med beräkningarna är att de inte tar hänsyn till att benen ska lyftas upp när den går framåt. De flesta organismer på ben utför cirkulära rörelser på fötterna när de ska till exempel gå framåt eller bakåt. Det innebär att under rörelsen har benen lyfts upp mellan startpositionen och slutpositionen. Roboten ska kunna simulera detta. Med beräkningarna ovan kan benen för roboten endast gå framåt och håller foten i kontakt med marken under hela rörelsen. En annan matematisk process måste skapas för behandling av cirkulära rörelser.

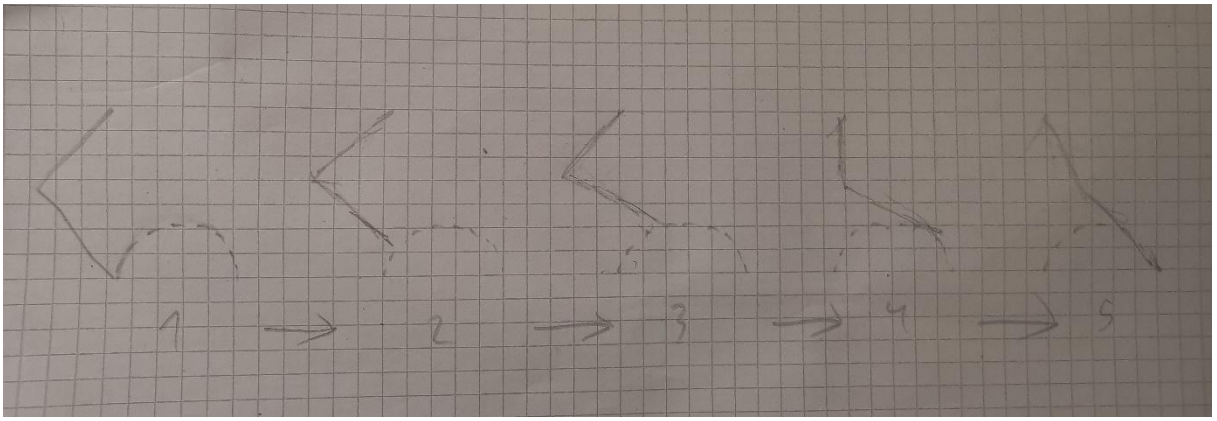

Fig. 11. Benen i cirkulär rörelse

Fig. 11. visar processen för robotens ben när den går framåt. Aqueous (2021) förklarar i en annan video att benens fot följer omkretsen på en halvcirkel under en cirkulär rörelse. Under hela processen har linjen h ändrat sig till olika läge som gör att benen befinner sig i olika längder vid varje punkt av hela rörelsen. Programmets mål är att kunna mata in hur mycket benen ska gå framåt och sedan beräkna linje h som är beroende av halvcirkelns omkrets och sist beräkna vinklarna genom att utgå från linje h från tidigare beräkningen. För att enkelt uppnå detta kan programmet använda sig av ett koordinatsystem där origon är startpositionen för rörelsen och en given punkt som kan kallas för x är steglängden för rörelsen. Programmet ska beräkna linjen h för varje punkt mellan origon och x och detta kan göras med en loop. Instruktionen för varje loop är alltså en matematisk process som beräknar linjen h längs en halvcirkel. Aqueous matematisk metod för beräkning av cirkulär rörelsen och linjen h tycktes vara svårt att förstå och därför skapades en egen metod för detta projekt.

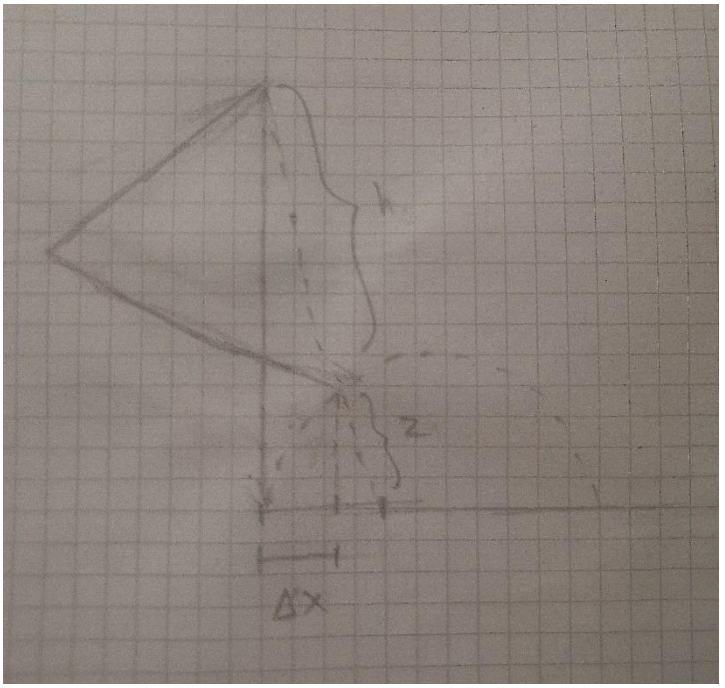

Fig. 12. Benen under en cirkulär rörelse

Fig. 12. visar läget för benen i en viss punkt under en cirkulär rörelse. Vid denna punkt har benen rört sig framåt med Δx. Början av halvcirkel ligger i origo där x är noll, där benen befinner sig i startpositionen. Linjen h är det som ska beräknas i programmet. Bilden visar att det bildar två trianglar. Den större triangel bildas utifrån linjen h som hypotenusan och en horisontell och vertikal axel. Den mindre triangel befinner sig inuti halvcirkel och har en hypotenusa som kallas för z.

Från föregående beräkningar för framåt rörelse för benen utan hänsyn till cirkulär rörelse så tolkade programmet linjen h som summan av linje h från bilden ovan och hypotenusan z från triangeln inuti halvcirkeln. Det är alltså en linje som sträcker sig från toppen av benen och ner till marken.

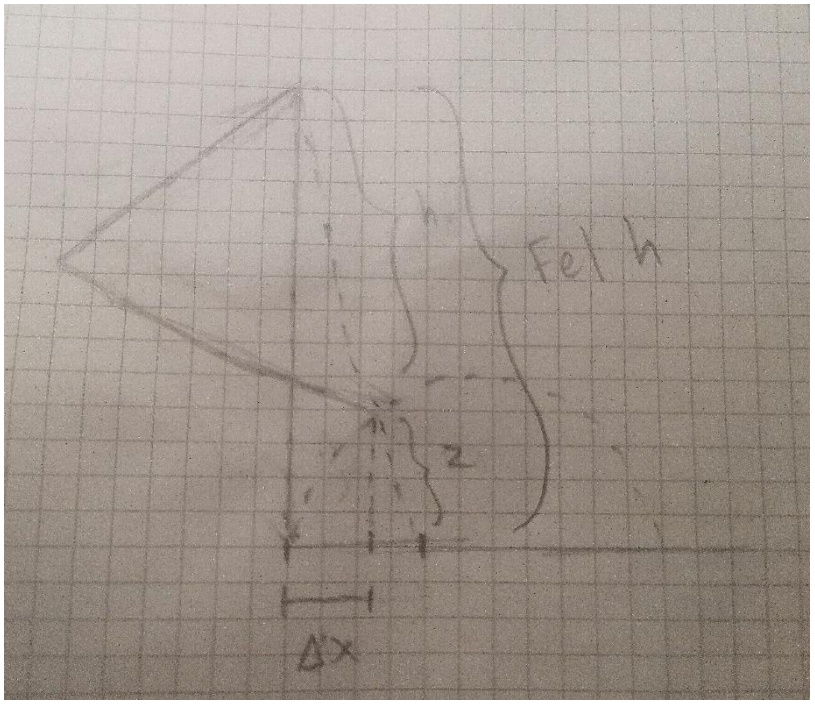

Fig. 13. En felaktig linje h bildas

För en cirkulär rörelse ska detta undvikas. Det görs genom att längden från hypotenusan z subtraheras från denna felaktiga linje h som börjar från toppen av benen och ner till marken. Slutligen får programmet den riktiga linje h för den cirkulära rörelsen som är över marken längs halvcirkeln precis som i bilden ovan.

Längden för hypotenusan z kan beräknas med hjälp av Pythagoras sats och cirkelns ekvation.

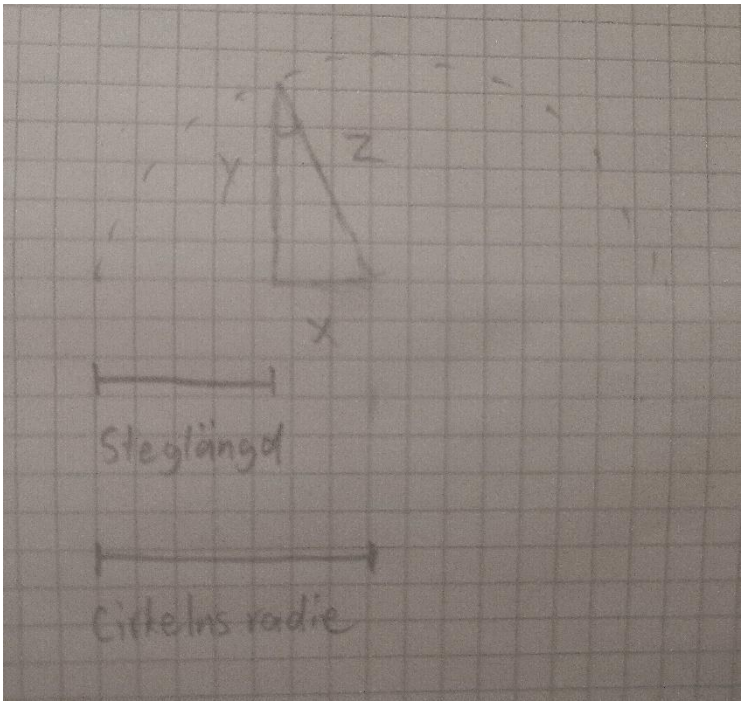

Fig. 14. Triangeln inuti halvcirkeln

Fig. 14 visar triangeln inuti halvcirkeln. De andra sidorna har döpts till y och x. Pythagoras sats ger att:

$$z = \sqrt{x^2 + y^2}$$

Längden för x kan räknas genom att ta radien av cirkeln minus den nuvarande steglängden. Steglängden kan kallas för s och cirkelns radie kan kallas för r. Uttrycket för x kommer att se ut på följande.

$$x = r - s$$

z kommer att se ur på detta sätt med det nya uttrycket:

$$z = \sqrt{(r - s)^2 + y^2}$$

Variabeln y kan lösas genom att använda cirkelns ekvation där r är radie för cirkeln.

$$r^2 = (r - s)^2 + y^2$$

$$y = \sqrt{r^2 - (r - s)^2}$$

Med y kan även uttrycket för x ändras till:

$$x = y \tan tan (a)$$

Vinkel a är toppvinkeln för triangeln inuti cirkeln som bildas av sidorna x, y och z. Slutligen beräknas hypotenusan z genom att använda uttrycken för x och y. Värdet från z subtraheras från den felaktiga h för att få den riktiga h sitter ovanpå cirkelns omkrets. Med värdet från den nya h kan vinkel för motorerna beräknas på samma sätt som i tidigare exempel med cosinussatsen.

3.3 Balans mekanism

Mekanismen för balans kan konstrueras utifrån trigonometri. Mekanismen behöver ta hänsyn till balans för roboten fram och bak, och sida till sida.

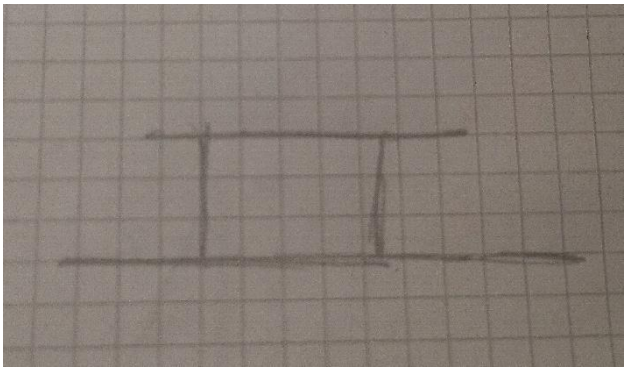

Fig. 15. Roboten framifrån

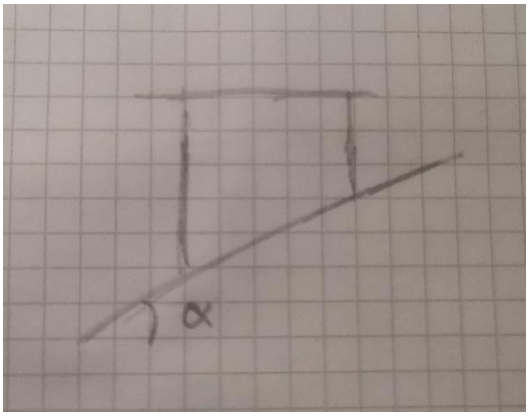

Fig. 16. Roboten balanserad

Bilderna visar hur roboten, framifrån, går från en icke-lutande plan till en lutande plan som lutar med vinkel α. Roboten har ändrat sina ben till olika längder. Programmet för roboten behöver alltså räkna ut hur mycket benens längd behöver ändra för att behålla balans utifrån lutningsvinkelns storlek.

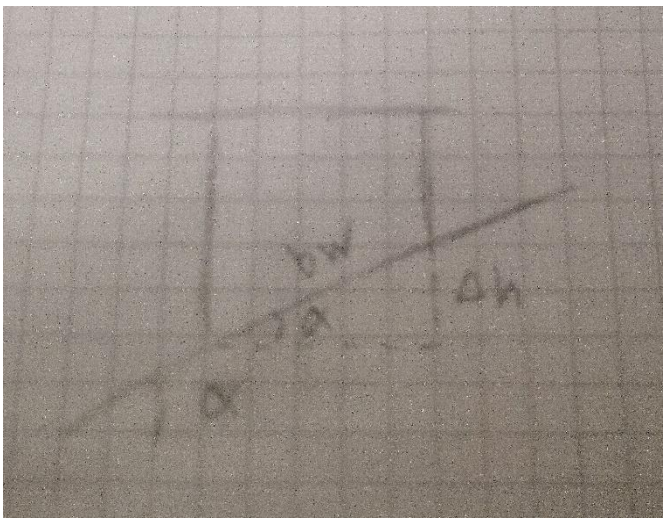

Fig. 17. Trigonometrin för balansering

Genom att rita en mindre triangel under roboten kan fler information användas. Bilden visar att robotens olika ben har ändrat sin längd med Δh från startpositionen när planet har blivit lutande. Triangelns hypotenusa är lika med kroppsbredden för roboten som döpts till bw, förkortning för bodywidth i engelska. Genom likformighet kan det konstateras att den mindre triangel har samma vinkel α som för den större triangel.

Genom att beräkna Δh som är beroende av vinkel α kan roboten anpassa benens längd för att få en grundläggande balansering. Δh räknas på följande sätt.

$$\Delta h = bw \sin sin (\alpha)$$

Värdet på Δh antingen subtraheras eller adderas till benens längd beroende på orienteringen. På bilderna ovan subtraheras Δh från den högra benen (vänt mot läsaren), och Δh adderas till vänstra benen. Hela processen är samma för balansering från sida till sida där den högra och vänstra benen kan ersättas med fram och bak benen. Beräkningarna utförs då i en PID loop med bestämda konstanter för PID ekvationens termer.

3.4 Design

För att förenkla byggandet behöver designen för själva kroppen inte vara komplicerad. Den få vara formad som en låda som ska innehålla all elektronik. Benen ska ha sin utseende lik roboten Spot men med några modifieringar. Först och främst ska benen få plats med tre motorer. Placeringen av motorerna är viktig eftersom det kommer att påverka beräkningarna, samt tröghet för roboten beroende på hur långt motorerna är från kroppens masscentrum. Detta togs upp i en video från Adam Beedle (2022) som förklarar att ju längre bort motorerna sitter från robotens masscentrum, desto större blir trögheten. Trögheten påverkar hastigheten för roboten genom att benen rör sig långsammare, samt stabilitet för roboten eftersom större tröghet ger större vridmoment för benen. Vridmoment från benen skapar krafter som kan skjuta på robotens orientering åt sidan när benen rör sig. Detta ska undvikas genom att minska trögheten från benen och motorerna. En lösning till detta är att bygga en design där alla tre motorer för benen sitter nära varandra och är nära kroppens masscentrum. Designen tog inspiration från Beedle i sin video där han går igenom en designprocess för hans egen fyrbent robot.

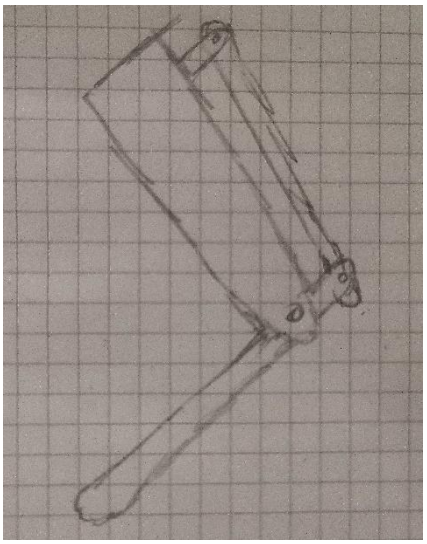

Fig. 18. Design för benen

Istället för att placera motorn som är kopplad till underbenen på knäleden så kan placeringen för denna motor förflyttas uppåt där det är närmare till både motorn för låren och robotens kropp genom att skapa en länk som går från toppen av låren ner till början av underbenet. Länkens ena ände är kopplad till motorn och den andra änden är kopplad till underbenen. På detta sätt kan motorn fortfararande röra underbenen från den nya placeringen.

Kvar återstår två motorer som behöver sin placering, motorn för låren och motorn för sidled rörelse. Beedle i sin video visade inga design för denna del av roboten och därför skapades för detta projekt en egen design för placeringen av de två motorerna.

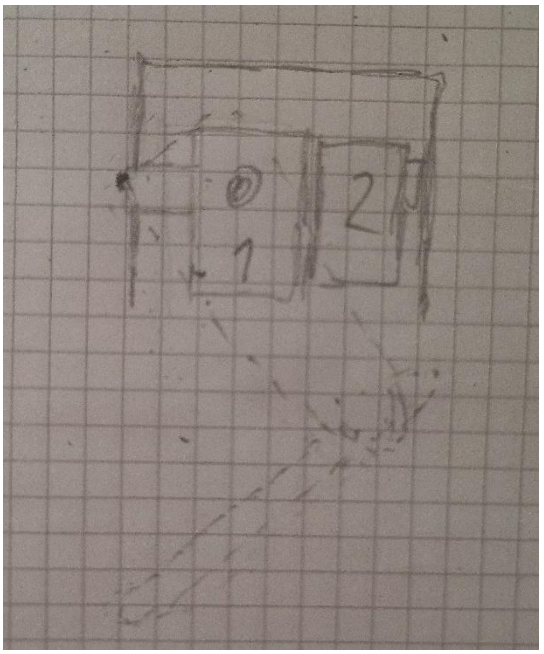

Fig. 19. Placeringen för motorerna

Fig. 19. visar hur placeringen för motorerna ska vara ungefär. Motor 1 är kopplad till låren och motor 2 är kopplad till motorn 1. Detta gör att hela benen rör sig när motor 2 är i gång. Motor 2 är orienterad så att den rör sig sidled.

För att minska kostnader används kartong och glasspinnar som material. Det hade varit bättre med en 3d-skrivare för mer noggrann konstruktion och stabilitet, samt att kostnaden för 3d-skrivarefilament per gram är mycket billig. Tillgång till en 3d-skrivare kunde inte fixas under projektets gång. Kartong och glasspinnar som hittades gratis användes istället. Eftersom detta är endast en första prototyp så räcker det med kartong och glasspinnar.

3.5 Servomotor

Roboten drivs på 12 servomotorer eftersom det är 3 motorer på 4 ben. Med låg budget i tanken valdes servomotorer av modellen SG-90. SG-90 är en billig servomotor som väger 9 gram och ger ett vridmoment mellan 1.8kg-2.4kg på 1 centimeter av vridarmen (huvuden av motorn) enligt specifikationerna från Kjell & Company (2017). Vridmoment anger hur mycket kraft motorn ger för att vrida ett föremål runt sin axel. Vridmomenten beror på motorns spänning och längden för armen som är kopplad till motorns huvud. Vid 4.8V ger motorn 1.8kg på 1 centimeter och 6V ger 2.4kg. Ett exempel är att om låren är 10cm i längd så kommer motorn för låren att skapa en tiondels vridmoment av 2.4kg vid 6V, alltså 240 gram, där låren betraktas som vridarmen för motorn. I exemplet kan motorn inte belastas med ett föremål som väger mer än 240 gram eftersom vridmomenten inte räcker till.

Tyngdkraften för roboten belastas på underbenen som är kopplad till en servomotor. Genom att beräkna vridmomenten för denna motor kan maxvikten för roboten avgöras. Om antagandet att motorn drivs på 6V och ger 2.4kg/cm (2.4kg på 1cm) vridmoment på längden för underbenen vilket är 13cm, kommer det att ge en maxvikt för benen på 2.4kg/13cm vilket är ungefär 185 gram. Benen kan alltså inte belastas med mer än 185 gram, vilket är mycket lättvikt och är svårt att uppnå när roboten ska innehålla andra elektronik som en dator och ett batteri. Men detta gäller endast för ett ben. När roboten har fler än ett ben i kontakt med marken kommer maxvikten per ben att adderas upp eftersom varje ben får stöd från andra benen som är i kontakt med marken. Om roboten har alla fyra ben i kontakt med marken så kommer alla ben att kunna lyfta upp totalt en maxvikt på 185 gram gånger 4 vilket ger 760 gram. Detta ger att roboten ska väga max 760 gram när den är stående på alla fyra ben. Maxvikten kommer att dock minska när roboten är i rörelse och har inte alltid fyra ben i kontakt med marken.

Rörelse för roboten kan betraktas som en hunds rörelse. Hunden liknar roboten och därför används den som ett exempel. I en enkel rörelse där hunden går framåt kommer den att som minst ha två av sina fötter i kontakt med marken. I vissa undantag kommer hunden att krypa och använda bara ett ben i taget för att gå framåt vid låga hastigheter. För att roboten ska kunna gå framåt är kravet att minst två av benen är tillräckligt starka för att lyfta roboten under rörelsen. Med tidigare beräkningar för en SG-90 motor på två ben kommer detta att ge en maxvikt för roboten på 185 gram gånger 2 vilket blir 370 gram.

På 370 gram är roboten mycket lättvikt. För att få mer utrymme för vikten kan motorerna för underbenen bytas till något starkare. En modell som är lite dyrare men är mycket starkare är MG995. En MG995 ger ett vridmoment på 8.5kg/cm vid 4.8V och 10kg/cm vid 6V enligt specifikationerna från DatasheetsPDF.com (u.å.). Den nya vridmomenten ger att robotens maxvikt ska vara på ungefär 1.5kg vid 6V. En nackdel med denna modell är att den väger 55 gram enligt specifikationen, nästan 6 gånger mer än SG-90. Denna nackdel bör inte vara något problem eftersom den nya maxvikten ger mycket mer utrymme. Roboten ska alltså innehålla 12 motorer där 4 stycken är av modellen MG995 och de övriga är SG-90.

3.4 Övriga elektronik

Robotens 12 motorer ska kopplas till en dator som talar för motorerna positioneringen. En lämplig dator som är både användarvänlig och liten i storlek är en Arduino UNO.
Arduino UNO är en programmerbar mikrodator eller mikrokontroller med 14 digitala ingångar där digitala kontrollerade komponenter kan anslutas till och styras från datorn enligt Arduinos produktsida (u. å.). Arduino UNO har även 6 digitala analoga ingångar som kan ta emot fler olika värden än ettor och nollor som digitala ingångar arbetar på. Arduino UNO kan drivas från en extern USB kopplad källa som en dator eller från ett batteri genom DC ingången som sitter bredvid USB ingången.

Både MG995 och SG-90 motorer går att kontrolleras digitalt och därför ansluts till de digitala ingångarna av datorn. Problemet är att motorerna kommuniceras på PWM digitala signaler och endast 6 av 14 ingångarna som är digitala kan kommuniceras med PWM signaler på en

Arduino UNO enligt Arduinos produktsida. Detta kan enkelt lösas genom att koppla en extern kort till datorn med fler PWM ingångar. En billig lösning är en servokontroller av modellen PCA9685 med 16 PWM ingångar där 16 servomotorer kan kopplas och kontrolleras samtidigt av kontrollen enligt produktsidan från Adafruit (u.å.).

Kontrollen kan sedan kommuniceras med Arduino UNO med endast två kablar som går till de analoga ingångarna på datorn. På så sätt har det sparat plats på ingångarna på datorn och det behövs endast två analoga ingångar på datorn för att kontrollera 12 servomotorer. PCA9685 kontroller kommuniceras med Arduino UNO med ett protokoll som kallas för I2C. Enligt Texas Instruments (2015) kan med detta protokoll flera enheter som använder I2C kopplas tillsammans på ett och samma datorbuss och kontrolleras av en enda enhet som kallas för master. Masterenheten kan urskilja enheterna genom att varje enhet har en specifik adress eller namn. Det går alltså teoretiskt sätt att koppla samman fler PCA9685 kontroller till samma ingång på en Arduino UNO så länge ingångarna har stöd för I2C. Arduino UNO har en I2C ingång som använder två pins, pin 4 och 5 som är analoga enligt Arduinos produktsida. Servokontrollen ska kopplas till dessa ingångar för att kunna starta kommunikation.

Motorerna har en anslutning på 3 stiftar, en för PWM signal, en för ström (+5V) och en för grund. Dessa stiftar ansluts till PCA9685 kontrollen där varje servoingång på kontrollen är färgkodad för att matcha färdkodningen på kabeln för motorn. Det finns alltså bara ett håll som man kan ansluta korrekt.

Utöver de två I2C anslutningarna på PCA9685 kopplas även en grund och +5V kabel till motsvarande ingångar på Arduino UNO för att sluta samman hela kretsen. För att få ström till motorerna krävs en extern strömkälla till PCA9685 som kopplas genom de två ingångarna på toppen av kontrollen med bemärkningarna V+ och GND. Denna strömkälla tillför ström endast till motorerna som är kopplade till kontrollen och inte Arduino UNO. En lämplig strömkälla är ett batteri med bra kapacitet. Kapaciteten för ett batteri anges ofta i mAh som står för millieampere hours eller milli-amperetimme på svenska. En SG-90 har en maxströmförbrukning på ungefär 700mA och MG995 har 1500mA enligt specifikationerna. Den totala strömförbrukningen i en värsta fall, alltså på maximal ström, blir 700mA gånger 8 plus 1500mA gånger 4 vilket ger 11600mA. En billig batterialternativ som ger stor kapacitet till ett lågt pris är en 18650 Li-on batteri. 18650 är en variant av litiumjonbatteri som ger hög kapacitet och hög urladdningshastighet till ett lågt pris och en liten formfaktor. Utöver kapaciteten är urladdningshastigeten en viktig faktor. Urladdningshastigheten anger hur mycket ett batteri kan urladdas under ett tidsintervall. DNK Power (u.å.) påpekar på sin webbsida att om batteriet urladdas med en hastighet som är större än vad den kan tålas kommer det att orsaka ostabilitet och även skada till batteriet vilket kan vara farligt. En 18650 batteri av modellen LG HG2 ger en urladdningshastighet av max 20 ampere kontinuerligt enligt specifikationerna från LG Chem (2015). Det är nästan dubbelt av motorernas strömförbrukning på 11600mAh eller 11.6 amperetimme. Det är viktigt att inte ladda ur batteriet på max urladdningshastighet även om det är inom ramen för att minska degraderingen av batteriet. Därför har modellen LG HG2 valts som batteri för robotens

motorer eftersom den är bland några av de få 18650 batteri modeller som erbjuder en hög urladdningshastighet till ett lågt pris och som är tillgänglig för att köpa.

En LG HG2 batteri har en maxspänning på 4.2V och en kapacitet på 3000mAh enligt specifikationen. Eftersom spänningen inte överstiger 4.8V gränsen för motorerna kommer den inte kunna driva motorerna. Lösningen är att ha två av batterierna i seriellkoppling för att öka spänningen enligt SparkFun Electronics (u.å.) på sin webbsida i en förklaring om batterier. Det kommer att då ge en maxspänning på 8.4V vilket är mer än den maximala spänningen som motorerna kan tålas, 6V. För att minska spänningen från batteriet till en lämplig nivå används en så kallad för buck-omvandlare som fungerar ungefär som en transformator som sänker spänningsnivån från en ingång. Den önskade spänningsnivån kan justeras genom att vrida på en skruv som sitter på modulen.

Två batterier i seriellkoppling ger större spänningsnivå med inte större kapacitet. Större kapacitet uppnås genom parallellkoppling enligt SparkFun Electronics (u.å). Spänningsnivån för batteriet är mer avgörande än kapaciteten eftersom motorerna drivs endast inom en viss ram för spänning. Kapaciteten spelar alltså mindre roll men är ändå något som bör tas hänsyn till. Med en kapacitet på 3000mAh och en urladdning på 11600mA ger en batteritid för 12 motorer på ungefär 15 minuter. Batteritiden beräknas genom att dela kapaciteten på urladdningen och gångra resultatet sedan med 60 för att omvandla till minuter. 15 minuter är en acceptabel batteritid för en billig prototyp då tidsramen är tillräckligt stor för att kunna utföra enkla tester. Batteriet går att laddas med en laddare som är kompatibel med 18650 batterier.

En annan viktig komponent som roboten behöver är en IMU sensor som kan känna av lutningen på roboten. En lämplig billig IMU sensor som går att programmeras med en Arduino är MPU6050 och valdes därför som IMU sensorn för roboten. MPU6050 liksom PCA9650 använder sig av I2C protokollet. Problemet är att de I2C ingångarna på Arduino UNO är redan använd av PCA9650. Eftersom I2C protokollet möjliggör fler enheter på samma datorbuss kan sensorn dela samma buss med PCA9650 för att lösa problemet. För att dela bussen mellan PCA9650 och MPU6050 används en breadboard.

En breadboard möjliggör kopplingar mellan olika komponenter utan lödning genom att använda bygeltrådar som kopplas mellan komponenterna. Varje hål är en anslutning på breadboarden. På kanterna av en breadboard ligger strömspår för plus och minus för att tillföra ström till kopplade komponenter. Varje hål som ligger på spåren är bunden till andra hål som ligger på samma horisontell led. Utöver strömspåren på breadboarden finns andra hål för kopplingar för att hanterar t.ex. kommunikation mellan komponenter. Varje hål är då bunden till andra hål på samma vertikal led. På andra breadboard modeller kan varje hål vara bunden till samma horisontell led istället och vice versa för hålen på strömspåren.

En video från Garage Geek Guy (2018) användes som stöd för kopplingen av både MPU6050 och PCA9685 på samma bus i en breadboard.

3.4 Byggande och kodning

Första steget är att koppla de två batterierna i serie och sedan testa om komponenterna fungerar som de ska. En batterihållare med plats för två 18650 batteri användes. Batterihållaren var inte sluten och behövdes därför slutas samman med en strömkabel.

Fig. 20. Batteri terminal

Kabeln hade en specifik storlek som kunde leda maximalt 25 ampere. Storleken gäller för ledaren i kabeln och anges i AWG (American Wire Gauge). Kabeln som användes för batterihållaren är 12 AWG vilket har en maximal ström på 25 ampere. För att seriekoppla batterierna löds en kabel mellan två terminaler på en sida av hållaren. Kabeln klipptes till rätt längd med båda änden skalad för att ta bort den isolerande materialen på ledarna. Med denna konfiguration ska den ena batteriet har sin pluspol uppåt och den andra batteriet ska ha sin pluspol nedåt när de sitter i hållaren. Mycket löd användes för att kunna hålla kabeln på plats för att undvika avbryta kretsen i fall kabeln skulle gå av.

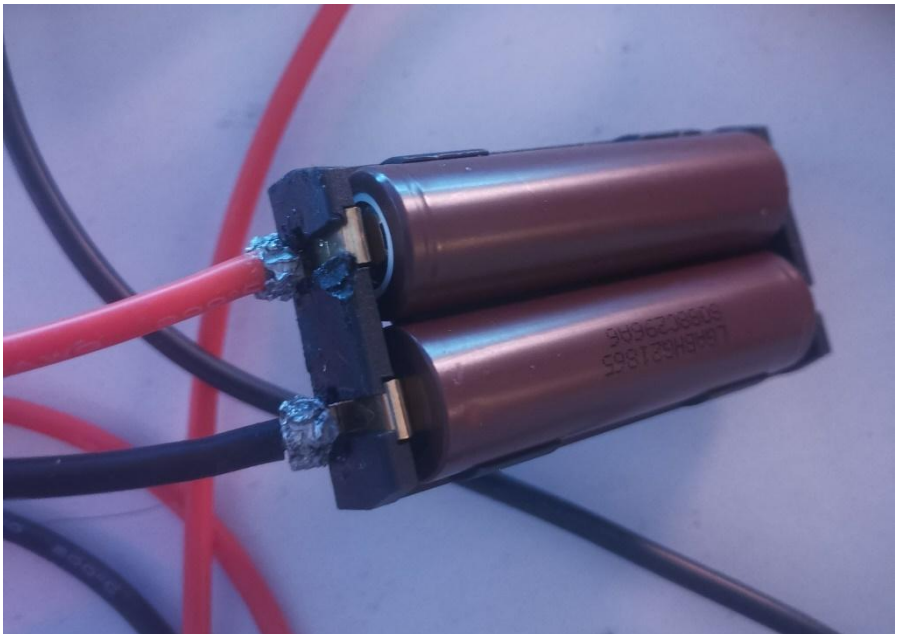

Fig. 21. Batteri strömkabel

Till de två andra terminalerna på hållaren löds två längre kablar som ska sedan anslutas till en buck-converter för att minska spänningen. Dessa kablar kan skäras till rätt längd senare i processen för att kunna få plats i roboten.

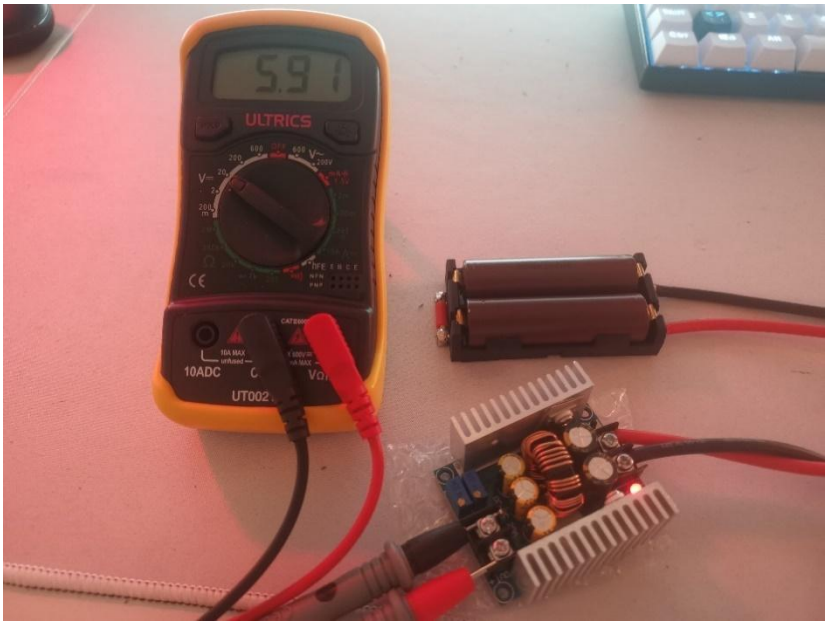

Fig. 22. Testning av buck-converter

Batterierna kopplas sedan till buck-converterns ingångar. För att kunna mäta spänningen kopplas en multimeter till buck-converterns utgångar. Spänningsnivån justeras genom att skruva på den gula skruven som sitter på en blå rektangel. Spänningsnivån i början var 7.4V eftersom batterierna hade 3.7V var för sig. Spänningen justerades ner till ungefär 5.9V men inte 6V för att undvika i fall spänningen hoppar över gränsen på något sätt och för att inte

göra motorerna för ostabila vid maximal spänning. Vid 5.9V drivs motorerna på nästan max vridmoment vilket är bra.

Fig. 23. Testning av servomotorer

För att testa motorerna kopplades batteriet från buck-convertern till PCA9685 ingångar för extern strömförsörjning. Till PCA9685 kopplades en SG-90 och MG955. PCA9685 kopplades sedan till de analoga pin 4 och 5 för I2C och +5V och GND på Arduino. Arduino kopplades sedan till datorn genom en USB kabel. För programmering av motorerna används ett program som heter Arduino IDE vilket är en utvecklingsmiljö för det språket som Arduino använder som är en variant av C++. PCA9685 har ett bibliotek för att göra programmeringen enklare.

```
// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

#define SERVOMIN  70 // This is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  450 // This is the 'maximum' pulse length count (out of 4096)
#define USMIN  600 // This is the rounded 'minimum' microsecond length based on the minimum pulse of 150
#define USMAX  2400 // This is the rounded 'maximum' microsecond length based on the maximum pulse of 600
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates
```

Fig. 24. Konstanter för PCA9685 biblioteket

I biblioteket finns några konstanter. De två första, SERVOMIN och SERVOMAX, anger minimum och maximum bredden för PWM signalerna. Bredden för PWM signalerna kan variera mellan olika servomodeller och måste därför justeras. Justeringen görs genom att skicka den lägsta PWM signalen med längden SERVOMIN och se om servomotorn hamnar exakt eller ungefär på noll grader. Samma sak görs för SERVOMAX för att se om motorn hamnar runt 180 grader vilket är maxvinkel för servomotorerna. För att göra processen enklare kan PWM signalerna omvandlas till grader. Då kan programmet specificera vilken

vinkel motorn ska vara i och skicka en motsvarande PWM signal. Detta görs med hjälp av en funktion som heter map() vilket gör om ett intervall med en min och max värde till ett annat intervall med ett annat min och max värde. I detta fall ska intervall för vinkel från 0 till 180 grader görs om till ett intervall med SERVOMIN och SERVOMAX som gränsvärdena.

```
int angletoPulse(int angle) {
  int pulse = map(angle, 0, 180, SERVOMIN, SERVOMAX);
  return pulse;
}
```

Fig. 25. Omvandling av grader till PWM

En funktion skapades med namnet angletoPulse() och tar in en vinkel som argument. Genom map() funktionen omvandlas vinkeln till en PWM signal i intervallet mellan SERVOMIN och SERVOMAX. Detta värde kallas för pulse och returneras av funktionen. För att kunna skicka PWM signaler till motorerna används en inbyggd metod i PCA9685 biblioteket som kallas för setPWM(). setPWM() tar in tre argument, vilken motor som signalen ska skickas till, vilken PCA9685 som signalen ska skickas till om det är fler PCA9685 i kretsen, och PWM signalen. Denna krets har endast ett PCA9685 och ges i argumenten med numret noll.

```
pwm.setPWM(0, 0, angleToPulse(90));
```

Fig. 26. Sändning av PWM signaler till motorn

Exemplet ovan visar hur ett PWM signal kan skickas till en servomotor på kontrollen. I detta fall är motorn den som sitter på plats noll på kontrollen (första ingången på PCA9685 från vänster), på den första och enda PCA9685 i kretsen och anges som noll i den andra argumenten. Sist kommer den tredje argumenten att vara en PWM signal. I exemplet omvandlas angleToPulse vinkel 90 till motsvarande PWM signal. Resultatet blir att motorn snurrar till 90 grader. Justeringen fick SERVOMIN och SERVOMAX till 70 respektive 450.

Nästa steg blir byggandet för benen. Benen består av låren, underbenen, en länk mellan underbenen och motorn. Först ritas upp delarna på kartong.

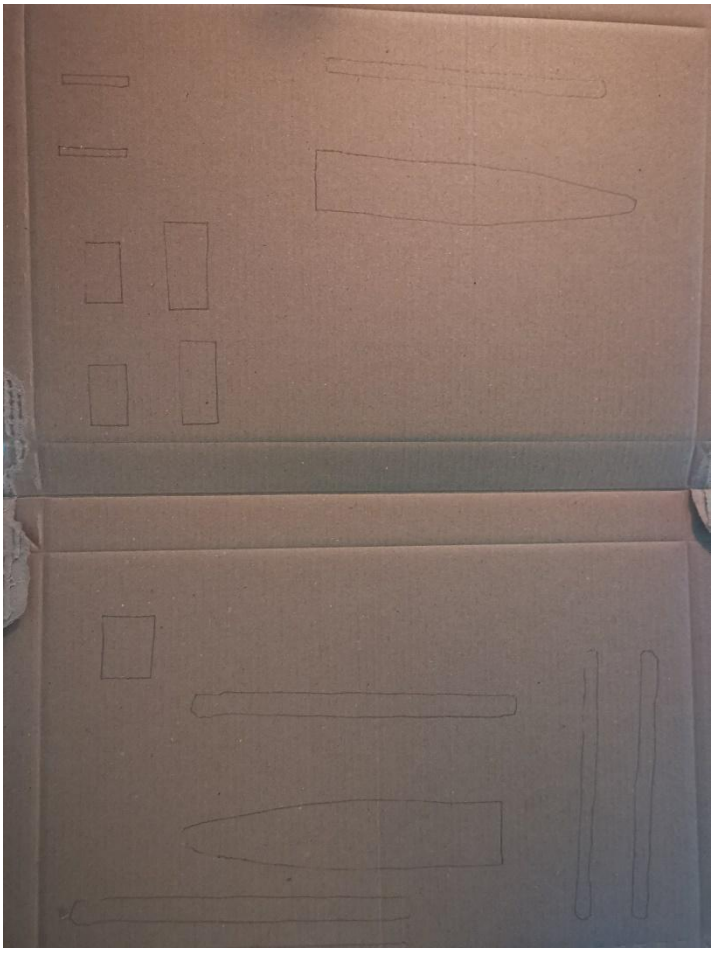

Fig. 27. Kartong

Delarna ritas upp två gånger för att kunna limmas ihop tillsammans och öka stabiliteten.
Delarna skärs ur kartongen med en kniv och limmas ihop med glasspinnar emellan delarna.

Fig. 28. Underbenen

Fig. 29. Knäleden

Knäleden består av en skruv som gör att underbenen kan vrida upp och ner.

Fig. 30. Placering av motor för låren Fig. 31.

Bilderna visar placeringen för motorn som kontrollerar underbenen. Motorn skruvas ner i några kartonghålar med två skruvar för att hållas på plats.

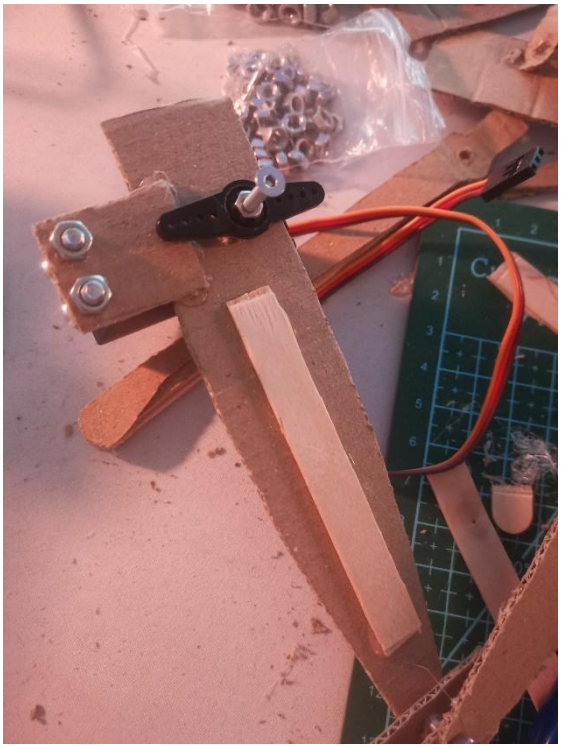

Fig. 32. Insidan av låren

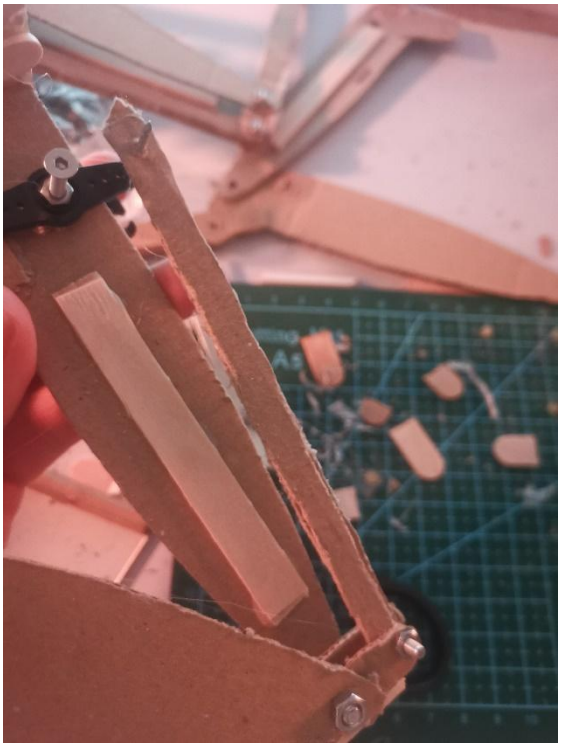

Fig. 33. Länk mellan motor och underben

Servomotor hornet skruvas ner till huvuden med en skruv. Länken mellan motorn och underbenen kopplas från hornet med en spik ner till underbenen med en mindre skruv. Extra glaspinnar limmades till benen enligt bilden för mer stabilitet.

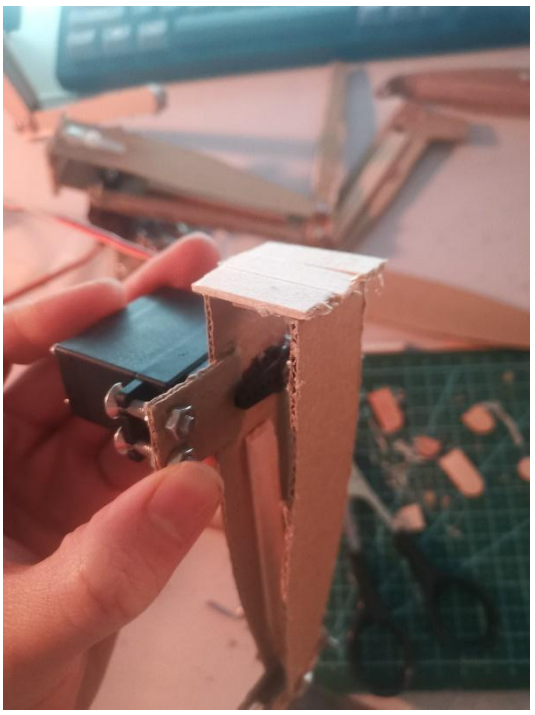

Fig. 34. Toppen av låren

Fig. 35. Färdig ben

Hela låren sattes ihop genom att limma toppen av låren med tre träbitar.

Nästa steg blir att koppla motorn för låren till benen. Denna motor ska även limmas ihop med den tredje motorn som ansvarar för sidled rörelser.

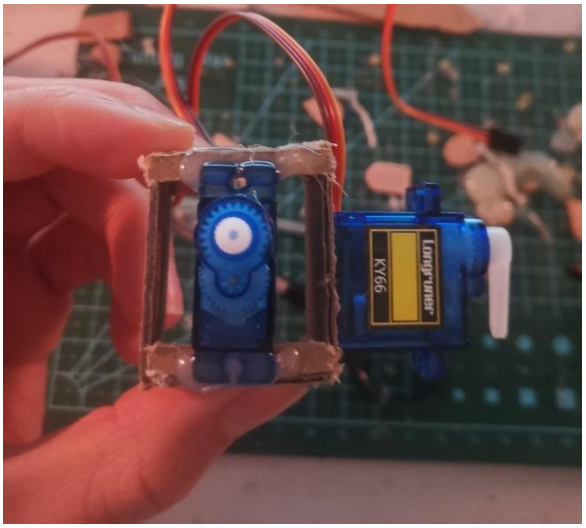

Fig. 36. Motor för låren och sidled rörelse

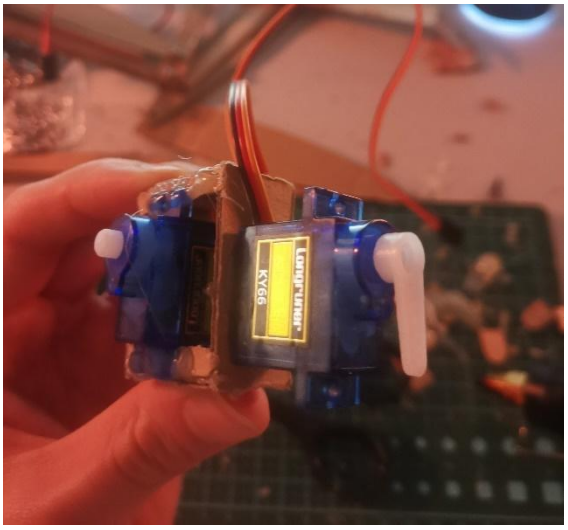

Fig. 37.

Motorn för låren limmas till en liten kartongbehållare som håller den på plats. Till sidan om behållaren limmas motorn för sidled rörelser. Motorn för låren kopplas till benen genom att koppla servohuvuden till ett servohorn som är redan limmad på benen.

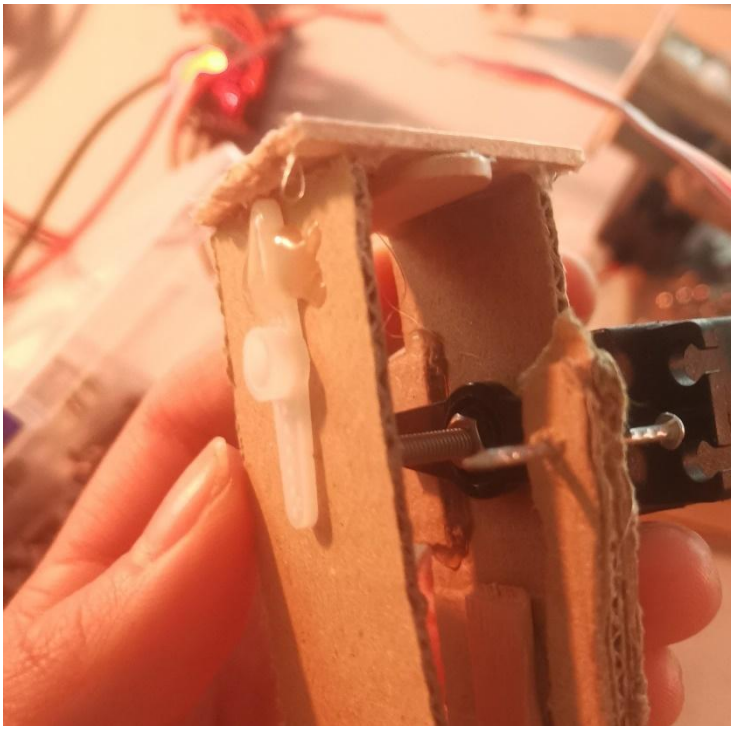

Fig. 38. Servohorn mellan låren och motorn

Innan motorn ska kopplas till benen behöver själva behållaren för motorn att sitta på höfterna på roboten.

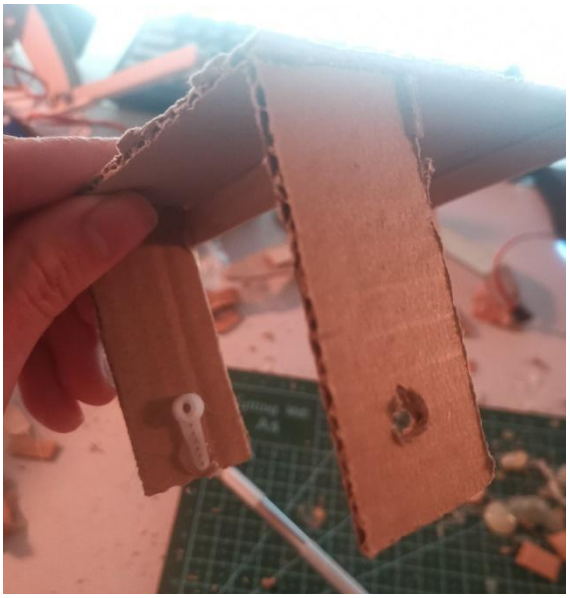

Fig. 39. Höften

Fig. 40. Färdig höft

Extra kartong användes för att bygga höfterna på roboten. De två motorerna sätts sedan mellan två kartong med ena sidan kopplad till en servohorn och andra sidan till en bit kartong som kan vrida fritt när benen ska röra sig sidled.

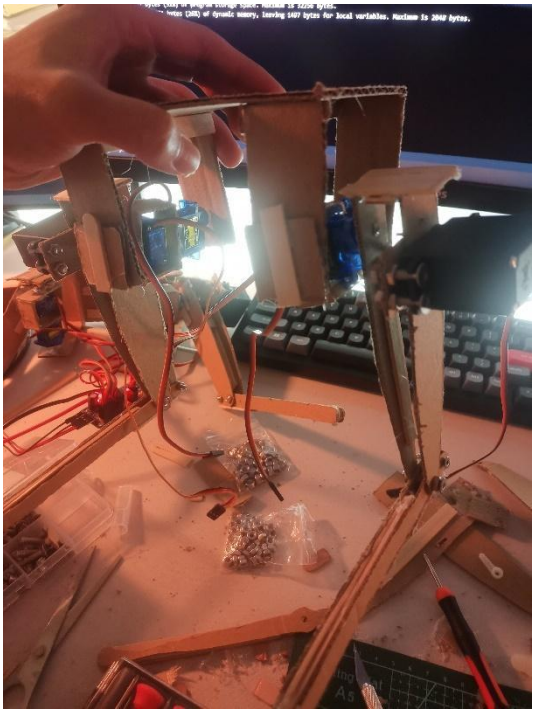

Fig. 41. Benen kopplad till höften Fig. 41.

Resterande benar och behållare för motorerna skapades. Delarna kopplades sedan ihop med en bas för kroppen. Resultatet blev enligt bilden nedan.

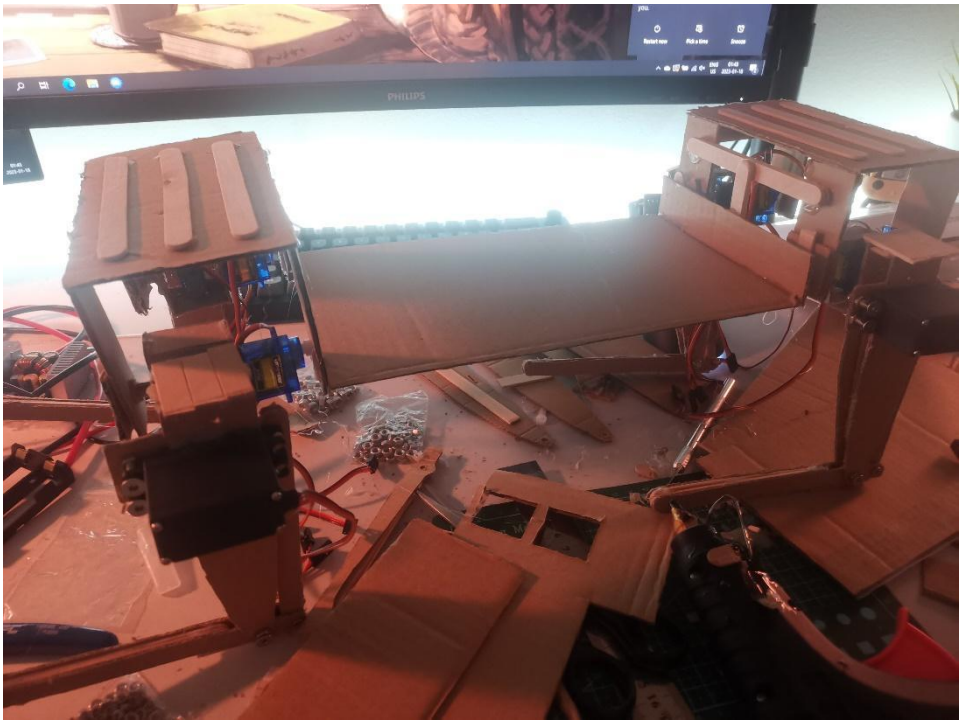

Fig. 42. Basen för kroppen

Extra kartong användes för att bygga väggarna för kroppen. Därefter sattes all elektronik som har testats hittills i kroppen.

Fig. 43. Färdig robot

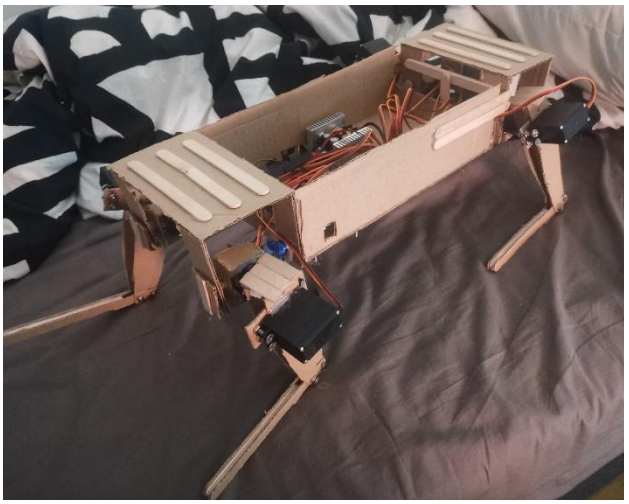

Fig. 44.

Nästa steg är programmeringen för roboten. För denna del av programmeringen har jag skapat all kod utan att hänvisa till andra källor och detta krävdes inte eftersom programmeringen är helt enkelt en omvandling av det matematiken som har behandlats från tidigare kapitel till kodspråk. Första funktionen som ska testas är ändringen i benens höjd. Detta skapades i en funktion som kallas för setLegHeight() som tar in två argument, vilken ben ska PWM signalen skickas till och till vilken längd benen ska vara.

```
void setLegHeight(Leg leg, int height) {


  leg.upperLegPos = acos((pow(leg.upperLegLength, 2) + pow(height, 2) - pow(leg.lowerLegLength, 2)) / (2 * leg.upperLegLength * height));
  leg.lowerLegPos = acos((pow(leg.upperLegLength, 2) + pow(leg.lowerLegLength, 2) - pow(height, 2)) / (2 * leg.upperLegLength * leg.lowerLegLength));

  leg.upperLegPos *= 180/PI;
  leg.lowerLegPos *= 180/PI;
```

Fig. 45. Inverskinematik för benens höjd

Fig. 45 visar koden som beräknar samma matematisk process som för den inverskinematiken som har behandlats från tidigare (se 3.1). acos() är invers cosinus och pow() är en upphöjningsfunktion. upperLegPos är vinkeln för motorn för låren och lowerLegPos är vinkeln för motorn för underbenen. Resultatet omvandlas från radianer till grader genom att

gångra med 180 delad på pi och skickas sedan som PWM signaler till de motorerna som tillhör benen. Därefter programmeras funktionerna som behandlar framåt rörelserna.

```
// Return lift off height for leg in a circular motion
int circularMotion(Leg leg, int stepLength, int radius, int height) {
  double verticalAxis = sqrt(pow(radius, 2) - pow((stepLength - radius), 2));
  double upperLegPos = acos((pow(leg.upperLegLength, 2) + pow(height, 2) - pow(leg.lowerLegLength, 2)) / (2 * leg.upperLegLength * height));
  double deltaHorizontal = verticalAxis * upperLegPos;
  double deltaHeight = sqrt(pow(deltaHorizontal, 2) + pow(verticalAxis, 2));

  return deltaHeight;

}
```

Fig. 46. Beräkning av cirkulär rörelse

Bilden ovan visar funktionen som jag har skapat som beräknar den riktiga höjden för roboten i en cirkulär rörelse med samma matematisk process som har behandlats i avsnitt 3.2. Funktionen kallas för circularMotion() med fyra argument, vilken ben, steglängden, radie för halvcirkeln och den nuvarande höjden för benen. Resultatet returneras med namnet deltaHeight som motsvarar hypotenusan z i den tidigare processen.

Funktionen som för benen framåt men håller fötterna i kontakt med marken kallas för walk().

```
void walk(Leg leg, int stepLength, bool circular) {
  balanceMode = false;

  double deltaAngle = atan((double) stepLength/leg.legHeight);
  double newHeight = stepLength / deltaAngle;
```

Fig. 47. Inverskinematik för framåt rörelse

Funktionen ger den nya linjen h när benen går framåt som har döpts till newHeight och motsvarar h_2 i den tidigare matematiska processen vid avsnitt 3.1. Genom att ange steglängden kommer funktionen att beräkna motorernas positionering med cosinussatsen genom att utgå från newHeight för att få fram benen till den önskade steglängden. Benen kommer att gå framåt men håller fötterna i kontakt med marken utan någon cirkulär rörelse.

För att få fram cirkulär rörelse kan deltaHeight från circularMotion() subtraheras från newHeight.

```
// if circular == true, calculate walk trajectory in a circular manner using circularMotion()
// else, perform non-circular walk trajectory where the legs are moved in a straight horizontal line
if(circular) {
  newHeight = newHeight - circularMotion(leg, stepLength, walkRadius, leg.legHeight);
  double newVertical = sqrt(pow(newHeight, 2) - pow(stepLength, 2));
  deltaAngle = atan((double) stepLength/newVertical);
  newHeight = stepLength / deltaAngle;
}
```

Fig. 48. Villkor för framåt cirkulär rörelse

Denna bit av koden läggs in i funktionen walk(). När benen ska utföra cirkulära rörelser kommer denna if-sats att köras. Då kommer newHeight att subtraheras från deltaHeight från circularMotion(). Variabeln deltaAngle är beräkningen på med hur stor vinkel motor för låren ska åka framåt. newHeight kan sedan definieras genom att dividera steglängden och deltaAngle enligt Pythagoras sats.

En hunds ben har som minst två av sina ben i kontakt med marken när den går framåt. För varje steg åker två av benen på samma sida av hunden framåt samtidigt. Detta fungerar i praktiken på en hund men är svårare för en robot att uppnå. Om två av benen på samma sida lyfts upp kommer roboten att bli ostabil och luta sig för mycket åt ett hål. Lösningen är att för varje steg framåt så åker ett ben och ett annat ben som sitter diagonalt mot den första benen framåt. Ett exempel är att om den vänstra frambenen går framåt så ska den högra bakbenen också gå framåt. Samtidigt som dessa ben går framåt kommer de två andra benen att gå bakåt utan att behöva utföra cirkulär rörelse. När benen går bakåt så behöver de att hålla kontakt med marken för att kunna putta roboten framåt. Därför har if-satsen ett villkor som säger om benen behöver gå cirkulärt, annars så utförs beräkningar utan hänsyn till cirkulär rörelse.

För att kunna få två av benen att gå framåt samtidigt som de två andra går bakåt måste någon form av multiprocessing används där datorn behandlar flera olika uppgifter samtidigt i olika trådar. Detta sätter dock krav på att processorn för datorn har fler än en tråd på sin arkitektur. Ju fler trådar en processor har, desto fler uppgifter den kan behandla samtidigt. Arduino UNO har inte stöd för multiprocessing och programmet som körs på den går inte att köras tillsammans med ett annat program. Det finns ett sätt för att köra flera olika program samtidigt utan att använda sig av multiprocessing genom att använda tidsintervaller där för varje tidsintervall ska flera olika program körs samtidigt.

```
void walkMotion() {
  unsigned long currentTime = millis();

  if(currentTime - prevTime > interval) {
    if(addition1) {
      if(!initial) {
        counter2 -= 1;
        walk(frontRightLeg, counter2, false);
        walk(backLeftLeg, counter2, false);
      }
      counter1 += 1;

      walk(frontLeftLeg, counter1, true);
      walk(backRightLeg, counter1, true);
    }
    else if(addition2) {
      counter1 -= 1;
      counter2 += 1;

      walk(frontLeftLeg, counter1, false);
      walk(backRightLeg, counter1, false);

      walk(frontRightLeg, counter2, true);
      walk(backLeftLeg, counter2, true);
    }

    if(counter1 == 10) {
      addition1 = false;
      addition2 = true;
    }

    if(counter2 == 10) {
      addition1 = true;
      addition2 = false;
      initial = false;
    }

    prevTime = currentTime;
  }

}
```

Fig. 49. Benens rörelsesekvens

Bilden visar funktionen walkMotion() som jag har skapat för att behandlar de olika rörelserna för benen samtidigt. En tidsfunktion körs i början av funktionen och lägger i minnet den nuvarande tiden i millisekunder. Om denna tid har passerat från det föregående tidstagandet med ett tidsavstånd som är definierad av tidsintervallet, så kommer all kod nedan att köras. Intervallet för programmen har definierats till 60 millisekunder vilket innebär att all kod i funktionen körs varje 60 millisekund. För varje tidsintervall kollar funktionen på två villkor, addition1 och addition2. Addition1 säger att den vänstra frambenen och den högra bakbenen ska åka fram med en steglängd som kallas för counter1, samt att den högra frambenen och vänstra bakbenen åker bakåt med samma steglängd counter1. Då två ben åker bakåt sätts argumentet circular i funktionen walk() till false och vice versa om benen åker framåt.

Addition2 fungerar på samma princip som addition1 men för motsatta ben och har en steglängd counter2. Funktionen i bilden är definierad för steglängden 10. De två sista if-satster kollar om antingen counter1 eller counter2 har nått värdet 10. Om en av de når värdet 10 ska motsvarande additionen stängas av. Om till exempel counter1 har nått 10 vilket innebär att den vänstra frambenen och den högra bakbenen har gått 10 centimeter i steglängd, så ska addition1 avslutas och sättas till false och addition2 ska slås på. Då addition2 är på kommer benen att gå bakåt. I addition1 finns en extra villkor, initial, som kollar om intervallet för funktionen är den första intervallet i hela rörelsen. Om initial är sant så ska endast två av benen åka framåt utan att de två andra åker bakåt. I början av en rörelse är det alltid bara ett par ben som rör sig framåt och den andra paren står stilla tills en senare intervall.

För balansering av roboten behöver programmet att läsa av MPU6050 sensorn.

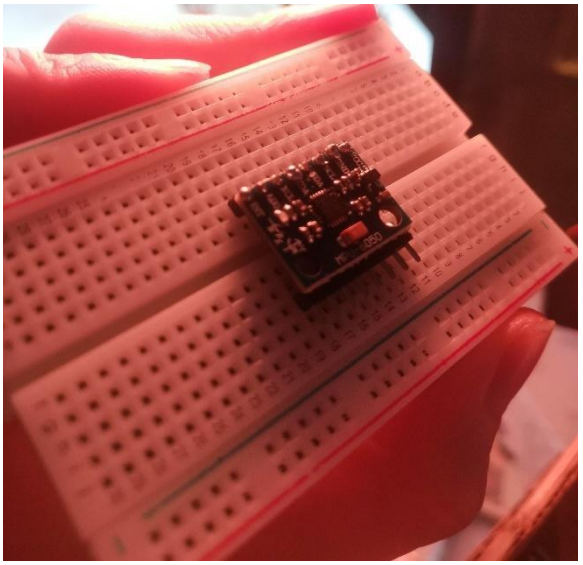

Fig. 50. MPU6050

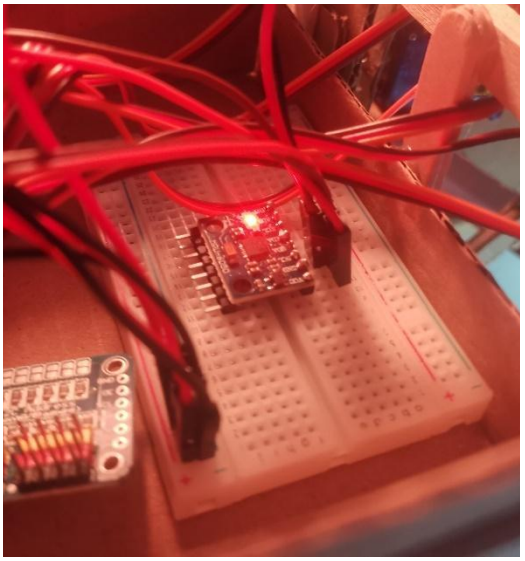

Fig. 51. Breadboard koppling

Bilden visar MPU6050 sensorn som kopplas till en breadboard. De I2C stiftarna av både sensorn och PCA9685 kontrollern kopplas till en buss på breadboarden enligt bilden till höger. +5V och GND av båda komponenterna kopplas till samma terminal på breadboarden. Riktningen för sensorn som placeras på breadboarden kommer att påverka nollställen för orienteringen. Det finns bemärkningar på sensorns krets som talar för riktningen för gyroskopens axlar vilket hjälper till med placeringen av sensorn.

```
// Balance without PID. Only translates the legs according to body orientation, not true balance
void staticBalance() {
  balanceMode = true;

  mpu.update();

  float ax = mpu.getAngleX();
  float ay = mpu.getAngleY();

  int heightLimit = 25;

  if(true) {
    double deltaHeightX = bodyWidth * sin(ax * (PI/180));
    double deltaHeightY = bodyLength * sin(ay * (PI/180));

    int leftLegHeight = frontLeftLeg.legHeight - (int) deltaHeightX;
    int rightLegHeight = frontRightLeg.legHeight + (int) deltaHeightX;
```

Fig. 52. Ändring av benens höjd beroende på lutningen från sensorn

För att testa om matematiken för balanseringen är korrekt skapade jag en funktion som kallas för staticBalance(). Denna funktion har inga PID krets utan ändrar bara benens längd proportionellt till storleken av robotens lutning. Biblioteken MPU6050_light.h användes för att kunna enkelt läsa värdena från sensorn. Sensorn betecknas med mpu i koden och har en adress som koden använder för att identifiera sensorn. Värdena för lutningen i sensorn läses i x och y led med metoderna getAngleX() och getAngleY(). Ändringen i benens längd ges i deltaHeightX och deltaHeightY. deltaHeightX ger ändringen i längden för benen om roboten lutar till sidorna. deltaHeightY ger ändringen i längden för benen om roboten lutar fram och bak. I många fall kommer roboten att luta diagonalt båda till sidorna och fram eller bak. Då ska programmet kunna kombinera deltaHeightX och deltaHeightY ihop för att ta hänsyn till ändringarna i båda lägen.

I koden ovan så har programmet endast behandlat ändringen för deltaHeightX. För att kombinera deltaHeightX med deltaHeightY kan de helt enkelt adderas eller subtraheras från varandra. För de två benen som ligger framåt ska deltaHeightX adderas med deltaHeightY och vice versa för bakbenen. Koden för ändringen av deltaHeightY kommer att se ut enligt bilden nedan.

```
if((leftLegHeight + (int) deltaHeightY) < heightLimit && (rightLegHeight + (int) deltaHeightY) < heightLimit) {
  setLegHeight(frontLeftLeg, leftLegHeight + (int) deltaHeightY);
  setLegHeight(backLeftLeg, leftLegHeight - (int) deltaHeightY);
  setLegHeight(frontRightLeg, rightLegHeight + (int) deltaHeightY);
  setLegHeight(backRightLeg, rightLegHeight - (int) deltaHeightY);
}
```

Fig. 53. Inkludering av deltaHeightY i balanseringen

Ett villkor har lagts till programmet för att se om den slutliga ändringen överstiger max höjden för benen som benämns med heightLimit i programmet. Värdet för heightLimit är 25 vilket innebär att benens längd inte får överstiga 25 centimeter. Detta är på grund av att själva konstruktionen av benen inte tillåter denna längd.
Med staticBalance() så kommer roboten att göra linjära ändringar i benens längd som är proportionellt mot lutningens storlek. Ett exempel är att om roboten lutar 20 grader till sidan så kommer deltaHeightY att vara 20 (värdet är bara ett exempel). Benen kommer att hålla denna nya längd så länge robotens lutning inte går tillbaka till noll vilket innebär att roboten måste alltid vara lutande för att en ändring i robotens ben ska kunna hållas på plats. Detta var avsikten med funktionen, att testa endast om matematiken fungerar. För att kunna få en riktig balansering måste en PID krets användas.

Sista steget är att bygga upp en funktion för balansering med PID. Funktionen som jag har skapat ser ut enligt bilden nedan.

```
void PID_balance() {
  balanceMode = true;

  if(millis() > timer + period) {
    timer = millis();

    mpu.update();
    float errorX = mpu.getAngleX();

    PID_p = kp * errorX;

    float errorDifference = errorX - previousErrorX;
    PID_d = kd * (errorDifference/period);

    PID_i += ki * errorX;

    float outputX = PID_p + PID_i + PID_d;

    // Calculate leg height according to output
    double deltaHeightX = bodyWidth * sin(outputX * (PI/180));

    int heightLimit = 25;  // Height limit for each leg to avoid trigonometric error for impossible heights in triangles

    if((10 <= frontLeftLeg.legHeight - (int) deltaHeightX && frontLeftLeg.legHeight - (int) deltaHeightX <= heightLimit) && (10 <= frontRightLeg.legHeight + (
      frontLeftLeg.legHeight -= (int) deltaHeightX;
      backLeftLeg.legHeight -= (int) deltaHeightX;
      frontRightLeg.legHeight += (int) deltaHeightX;
      backRightLeg.legHeight += (int) deltaHeightX;
```

Fig. 54. Balansering med PID

Koden följer bland annat ekvationen för PID och körs upprepande i en intervall av 50 millisekunder. Felvärdet för ekvationen är bland annat skillnaden i sensorns lutning från noll grader. Notera att balanseringen i bilden gäller endast för lutningar i sidled och inte framåt och bakåt. PID_p ger den proportionella delen av ekvationen genom att ta det nuvarande felvärdet, errorX, och gångra det med en konstant som kallas för kp. PID_i ger integralen av felvärdet genom att addera felvärdet gånger en konstant, ki, till sig själv vid varje intervall. Slutligen ger PID_d derivatan i ekvationen genom att ta skillnanden mellan det nuvarande felvärdet, errorX, och felvärdet från föregående intervall, previousErrorX, delad på intervallens storlek som är 50 millisekunde och gånger det hela med en konstant, kd. Genom att summera PID_p, PID_i och PID_d med varandra får funktionen ett slutvärde, outputX, som ges i radianer. outputX omvandlas till grader och används för att beräkna ändringen på

benens längd, deltaHeightX. Det går att lägga deltaHeightY till PID funktionen på samma sätt som i funktionen staticBalance().

Slutligen skickas PWM signalerna till benen. Vid slutet av intervallen sätts variabeln previousErrorX till errorX.

```
  if(true) {
    setLegHeight(frontLeftLeg, frontLeftLeg.legHeight);
    setLegHeight(backLeftLeg, backLeftLeg.legHeight);
    setLegHeight(frontRightLeg, frontRightLeg.legHeight);
    setLegHeight(backRightLeg, backRightLeg.legHeight);
  }

  previousErrorX = errorX;
}
```

Fig. 55.

Konstanterna kp, ki och kd har fåtts fram genom testning. kp testades först med värdet 1 vilket ger att för varje 50 millisekunder så återgår roboten till balans med 1 grader. Genom att utgå från kp = 1 och sedan testa fram olika värden för ki och kd fick till slut PID funktionen en acceptabel effekt. Ett bra sätt för att få en bild för vad ki och kd ska vara är att kd kommer att påverka hastigheten för ändringen på felvärdet och ki ska generellt vara liten för att slutvärdet inte ska överstiga det önskade värdet. Värdena för konstanterna i programmet ses i bilden nedan.

```
int kp = 1;
int ki = 0.2;
int kd = 100;
```

Fig. 56. PID konstanter

4. Resultat

Matematiken verkar stämma och får roboten att utföra planerade rörelser och balans. För cirkulär rörelsen är rörelsen ungefärligt men inte ändå 100%. Rörelserna är även lite skakiga vid låga hastigheter.

Kroppen med all elektronik och batteri vägde tillsammans ungefär 800 gram. Även om den inte översteg maxvikten vilket var ungefär 1.7kg så hade roboten fortfarande svårighet för att kunna stå på sina ben. All testning med roboten kunde därför inte utföras i ett mer verkligt scenario där roboten står på marken. Istället byggdes en plattform för att hålla roboten upp i luften så att roboten inte behöver stå utan svävar sina ben i luften.

Fig. 57. Roboten på en platform

Benen hade också ett problem med att den åkte in i roboten om roboten observerades framifrån.

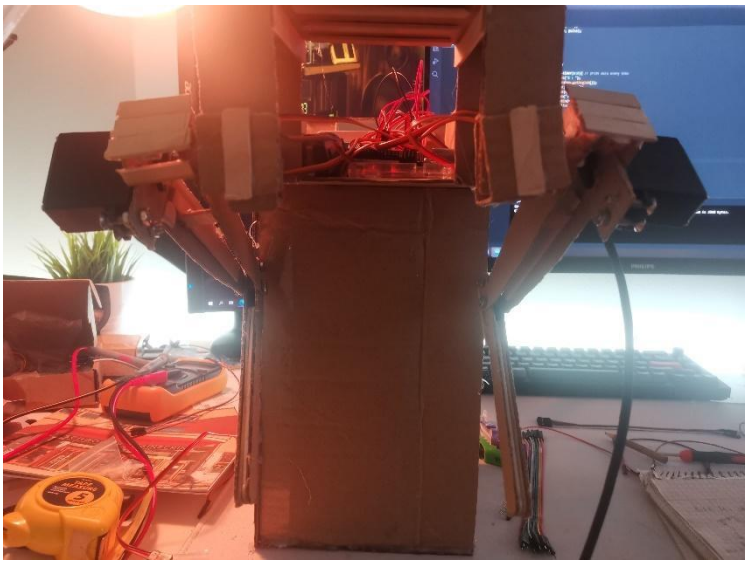

Fig. 58. Benen lutar in i roboten

Fig. 58. visar hur benen lutar sig in mot kroppen vilket orsakar ostabilitet. En temporär lösning var att förflytta benen utåt genom att ändra positioneringen på motorn.

Fig. 59.

Nu när benen är lite mer bort från kroppen så kommer den att bli mer stabil när den står på sina ben. Dock var detta fortfarande inte tillräckligt för att kunna få roboten att stå på sina ben.

All testning på roboten gjordes när roboten är i luften och kommer därför att inte motsvara verkligheten till 100% men ger ändå en bra grund för hur funktionerna beter sig i programmet. Nedan visas några bilder när roboten går framåt.

Fig. 60.

Fig. 61.

Nedan visas några bilder för balanseringen av roboten.

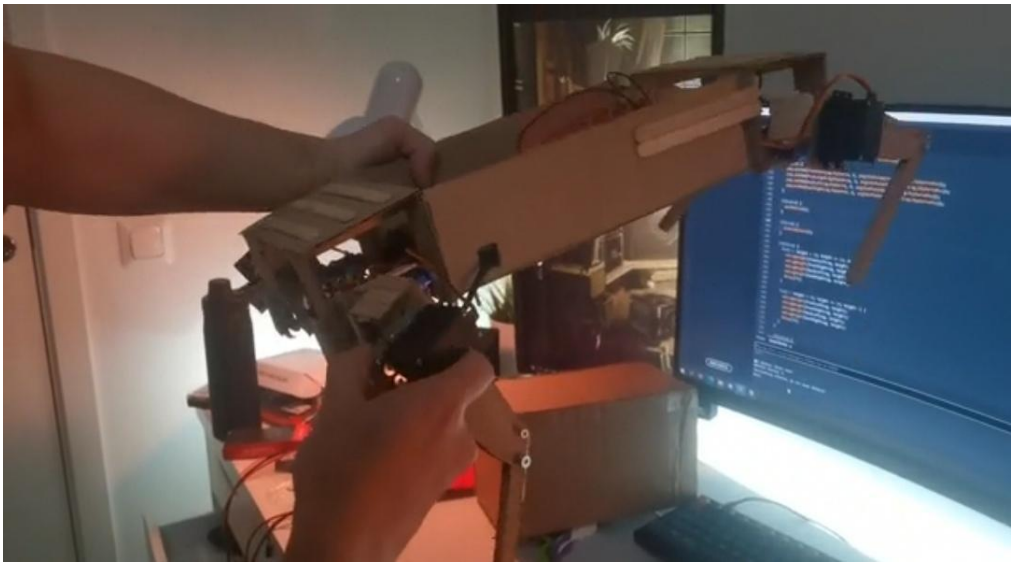

Fig. 62.

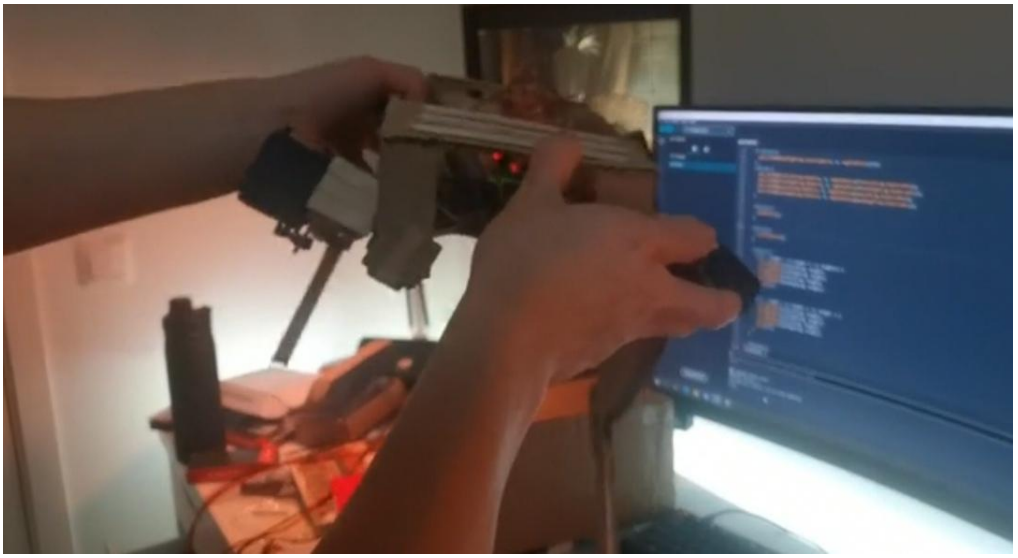

Fig. 63.

I bilderna ovan har benens längd anpassat sig till robotens lutning och har därmed ändrat sin längd för att bibehålla robotens balans. Balanseringen kunde inte testas när roboten står på marken eftersom benen inte kunde lyfta upp roboten tillräckligt.

5. Diskussion

Att roboten inte kunde stå på sina ben berodde på några faktorer. Först och främst är själva materialet inte tillräckligt stark. Kartongen böjer sig när den belastas. Misstaget var att länken mellan underbenen och dess motor var gjord av kartong, samt att kring knäleden så bestod det för mesta dels bara av kartong. Då benen belastas så böjs denna länk samt att knäleden trycks ihop och är orsaken till att benen inte kan lyfta upp roboten. En förbättring skulle ha varit att göra länken av glasspinnar och att göra om hela underbenen till endast glasspinnar och sedan borra ett hål på änden för att kunna sätta skruven för knäleden. En annan lösning är att bygga roboten med en 3d-skrivare som använder sig av starkare material.

Ett annat problem var att rörelserna var skakiga. Motorerna verkar inte förflytta benen effektiv även om vridmomenten är tillräckligt stort. Det kan ha berott på att det inte var bara vridkraften för benen som belastades på motorerna utan även tyngdkraften av benen som har en annan riktning än vridmomenten. Benens tyngdkraft gjorde att benen lutande in i roboten. Motorernas förmåga kommer att minskas eftersom tyngdkraften motverkar vridkraften då de har olika riktningar vilket gör att motorerna blir svagare och ger därför skakiga rörelser. Det beror även på att servomotorns horn är gjord av plast som böjer inåt med tyngdkraften från benen och gör att kontakten mellan motorn och benen inte sitter helt fast. En lösning skulle vara att minska benens storlek och därmed kroppens storlek, men det skulle innebär att hela roboten byggs om. En annan lösning är att byta motorerna till något starkare men skulle öka kostnader.

Funktionen setLegHeight() som sätter benens längd i centimeter ger inte exakt samma resultat i verkligheten. Skillnaden mellan höjden i programmet och höjden i verkligheten är på ett par centimeter, ibland mer. Detta kan bero på att benen inte har exakta dimensioner som programmet använder. Då skulle parametrarna för benens dimensioner behöva ändras. Det är en nackdel med att bygga roboten från kartong och ger inte alltid exakta dimensioner som om roboten hade byggts med en 3d-skrivare. Men att några millimeters skillnad i benstrukturen kan leda till flera centimeters skillnad i höjden är kanske inte rimligt. En annan orsak till problemet som är mer sannolikt är noggrannheten på motorernas positionering. Motorerna kan ha visat en annan vinkel än det som står i programmen. Även på några graders skillnad från alla tre motorer per ben så kan det ändå leda till att höjden får flera centimeters avvikelse.

Cirkulär rörelsen hos robotens ben var inte 100% cirkulärt. Vid slutet av en rörelse, alltså nära steglängden, så hoppar robotens ben direkt ner till marken utan att följa resten av halvcirkeln. Det är fortfarande oklart vad som är orsaken till detta, men två faktorer är värt nämna som kan ha haft en effekt. Det första är att motorerna SG-90 och MG955 har olika hastighet vilket kan påverka rörelserna i och med att en av motorerna inte når ändpunkten lika snabb som den andra. Det andra är att det kan ha varit ett matematisk fel.

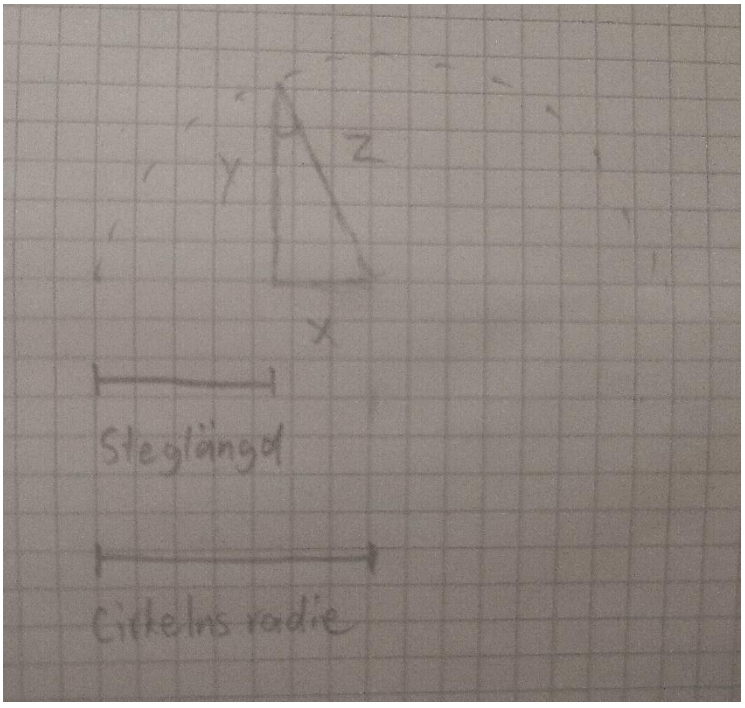

Fig. 64

Vid beräkning för x (Fig. 64.) så har x definierats som cirkelns radie minus steglängden. Detta är felaktig och har antagandet att hypotenusan z har alltid kontakt med cirkelns mittpunkt. Triangeln skulle kunna befinna sig på ett annat ställe där z inte har kontakt med cirkelns mittpunkt. En möjlig lösning skulle vara att inte ta hänsyn till x alls och beräkna endast y och z. y kan beräknas med hjälp av cirkelns ekvation och z kan definieras som y delad på cosinus för toppvinkeln av triangeln. Toppvinkeln av triangeln är samma som vinkeln som motorn för låren behöver gå framåt med i rörelsen (se Fig. 10.).

Det finns flera sätt för att beräkna invers kinematiken och det som användes i detta projekt var kanske inte den mest effektiva. Bland annat linjär algebra och matriser kan användas för att behandla benens koordinater på ett tredimensionellt koordinatsystem. Det som användes för roboten arbetar endast på två axlar, x och y. På grund av begränsad kunskap så kunde inte linjär algebra tillämpas till roboten.

Sammanfattningsvis, det finns flera förbättringar för roboten som valet av materialen, användning av 3d-skrivare för mer detaljerad design, minskning av robotens storlek för att mindre belasta motorerna och en bättre algoritm för inverskinematik. För en första prototyp var resultatet acceptabel och det mesta av teorin fungerar som det ska.

Referenslista

Biswal, P. (2019). *Development of quadruped walking robots: A review.* Arunachal Pradesh: Department of Mechanical Engineering, National Institute of Technology Arunachal Pradesh. Tillgänglig: https://www.sciencedirect.com/science/article/pii/S2090447920302501

York Precision Machining and Hydraulics. (2019). *Hydraulic vs. Pneumatic vs. Eletric Actuators*. Hämtad 2023-03-07 från https://yorkpmh.com/resources/hydraulic-vs-pneumatic-vs-electric-actuators/

Owen, J. [Jared Owen]. (2020, 10 juni). *How does an Electric Motor work? (DC Motor)* [Videofil]. Hämtad 2023-03-30 från https://www.youtube.com/watch?v=CWulQ1ZSE3c

Budimir, M. (2023, 24 mars). Comparing stepper and brushless dc motors [Blogginlägg]. Hämtad 2023-03-30 från https://www.motioncontroltips.com/comparing-stepper-and-brushless-dc-motors/

Evans, P. [The Engineering Mindset]. (2022, 23 januari). *Servo Motors, how do they work?* [Videofil]. Hämtad 2023-03-30 från https://www.youtube.com/watch?v=1WnGv-DPexc

Or, B. (2022, 19 augusti). Inertial Measurement Unit (IMU) Explained [Blogginlägg]. Hämtad 2023-03-31 från https://builtin.com/internet-things/inertial-measurement-unit

Douglas, B. [MATLAB]. (2018, 22 maj). *What is PID Control? | Understanding PID Control, Part 1* [Videofil]. Hämtad 2023-01-09 från https://www.youtube.com/watch?v=wkfEZmsQqiA

PID. (2023, 4 februari). I *Wikipedia*. Hämtad 2023-01-09 från https://en.wikipedia.org/wiki/PID_controller

MathWorks. (År okänt). *Inverse kinematics (IK) algorithm design with MATLAB and Simulink.* Hämtad 2022-10-08 från https://www.mathworks.com/discovery/inverse-kinematics.html

Aqueous, N. [Nic Aqueous]. (2021, 3 mars). *Building a robot dog #1 Hardware, Inverse Kinematics* [Videofil]. Hämtad 2022-10-06 från https://www.youtube.com/watch?v=wtf0RDwPl0c

Aqueous, N. [Nic Aqueous]. (2021, 10 april). *Building a robot dog #2 First Steps, Inverse Kinematics, Full Code* [Videofil]. Hämtad 2022-10-06 från https://www.youtube.com/watch?v=1xFajou-J3I

Beedle, A. [Adam Beedle]. (2022, 28 juli). *Designing a compliant leg for my robot quadruped* [Videofil]. Hämtad 2022-10-07 från https://www.youtube.com/watch?v=bDxItdyQ3jc

Kjell & Company. (2017, 30 november). *SG90 Micro Servo*. Hämtad 2022-11-01 från https://www.kjell.com/globalassets/mediaassets/701916_87897_datasheet_en.pdf?ref=4287817A7A

DatasheetsPDF.com. (u, å). *MG995 High Speed Metal Gear Dual Ball Bearing Servo*. Hämtad 2022-11-01 från https://datasheetspdf.com/pdf-file/839879/ETC/MG995/1

Arduino. (u, å). *Arduino UNO Rev3*. Hämtad 2022-10-06 från https://store.arduino.cc/products/arduino-uno-rev3

Adafruit. (u, å). *Adafruit 16-Channel 12-bit PWM/Servo Driver – I2C Interface – PCA9685*. Hämtad 2022-10-06 från https://www.adafruit.com/product/815

Texas Instruments. (2015). *Understanding the I2C Bus*. Texas: Texas Instruments.

[Garage Geek Guy]. (2018, 12 november). *How to use multiple i2c devices on the same bus with the Arduino* [Videofil]. Hämtad 2023-02-14 från https://www.youtube.com/watch?v=nEySekIIxpw

LG Chem. (2015). *Product Specification Rechargable Lithium Ion Battery Model: INR18650HG2 3000mAh.* Seoul: LG.

DNK Power. (u, å). *Why 18650 Battery Would Explode*. Hämtad 2022-03-14 från https://www.dnkpower.com/why-18650-battery-explode/

SparkFun Electronics. (u, å). *What is a Battery?*. Hämtad 2022-12-23 från https://learn.sparkfun.com/tutorials/what-is-a-battery/usage

SparkFun Electronics (u, å). *How to Use a Breadboard*. Hämtad 2023-02-12 från https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all

Bilaga 1

Videolänk till demonstration av robotens setLegHeight funktion:
https://youtu.be/CtBXIVd-Y-s

Bilaga 2

Videolänk till demonstration av robotens cirkulära rörelse:
https://youtu.be/rcmRjJhF-vY

Bilaga 3

Videolänk till demonstration av robotens framåt rörelse:
https://youtu.be/aiFeO2Nt8Gc

Bilaga 4

Videolänk till demonstration av robotens balansering utan PID:
https://youtu.be/pcD58t9XcVY

Bilaga 5

Länk till källkoden för roboten:
https://github.com/kaidev04/Quadruped-Robot-Arduino