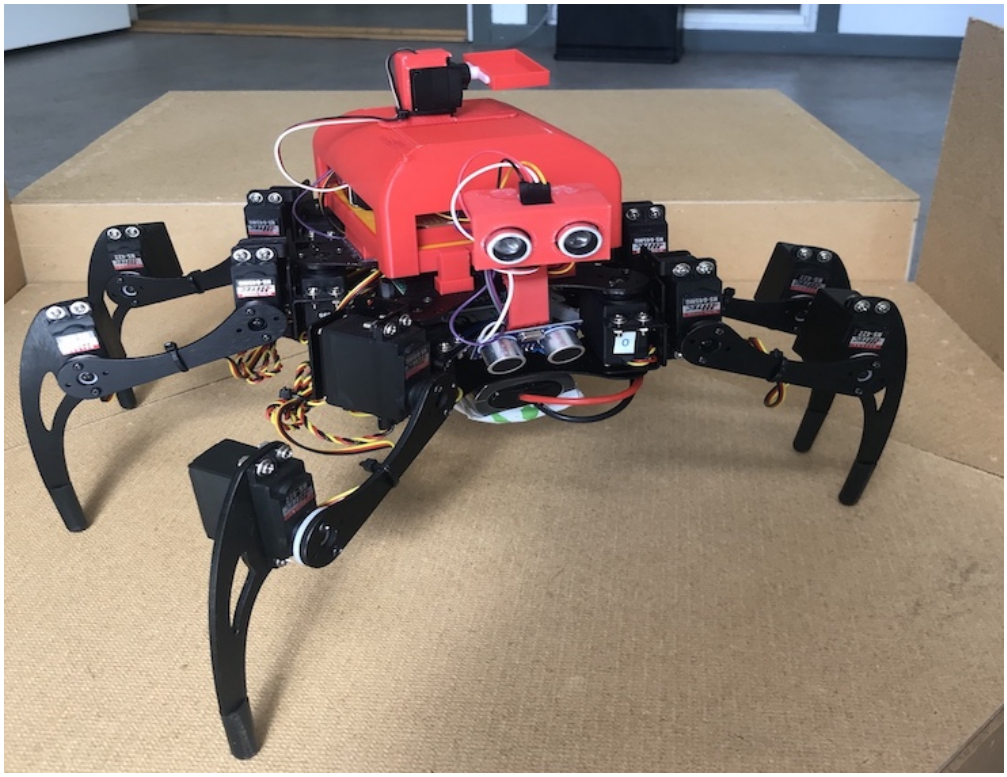
CHALMERS
UNIVERSITY OF TECHNOLOGY



Framtagning och implementering av en autonom klättrande robot

Ett examensarbete med inriktning på mjukvaru- och hårdvaruutveckling

Examensarbete för institutionen Elektroteknik

Erik Rundberg
William Ripgården

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2020

Examensarbete 2020:06

Framtagning och implementering av en autonom klättrande robot

Development and implementation of an autonomous climbing robot

Erik Rundberg
William Ripgården

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Elektroteknik
Chalmers tekniska högskola
Göteborg, Sverige 2020

Framtagning och implementering av en autonom klättrande robot
Development and implementation of an autonomous climbing robot
Erik Rundberg
William Ripgården

Handledare: Björn Bergholm, Broccoli AB
Handledare: Göran Hult, Elektroteknik
Examinator: Göran Hult, Elektroteknik

Examensarbete 2020
Examensarbete vid Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon: +46 31 772 1000

Framsida: Den slutgiltiga prototypen av den framtagna autonoma roboten

Göteborg, Sverige 2020

Abstract

The aim of this project is to in the end present a self-devoloped autonomous prototype of a Arduino-controlled robot with the task of moving freely and being able to avoid walls by turning right or left, but also being able to climb over obstacles, such as stairsteps.
As today´s society and factories are becoming more and more autonomous, this project is very current in the present whose functions can be further developt to find suitable work-areas in society.
The thesis work includes both hardware and software development which suits both graduates ongoing education.
Software development is mainly done by coding in the language used by Arduino (similar to C++), to get the robot to perform functions and other commands using purchased electronic components.
In the terms of hardware, CATIA 3D-models have been drawn and later manufactured in a 3D-printer for use on the robot. Other components and hardware such as servomotors and chassis have also been purchased. To prove that the results have been achieved, a wooden staircase has also been built.

The report describes the underlying theory and areas of application for the necessary components used in the project, as well as its approach and implementations.
The report contains the problems that have arisen during the course of the work as well as our solutions to the problems and why the problems may have arisen from the beginning.

Keywords: Autonoumus, Robot, Hardware, Software, Arduino, Programming, C++, CATIA, 3D-printing

Sammanfattning

Detta projekt syftar till att i slutändan presentera en egenutvecklad autonom prototyp av en Arduino-styrd robot med uppgift att röra sig fritt och kunna undvika väggar genom att svänga höger eller vänster samt klättra över hinder, likt ett eller flera trappsteg.
Då dagens samhälle och fabriker blir allt mer och mer autonoma är detta ett arbete aktuellt i nutiden vars funktioner kan byggas vidare på för att hitta lämpliga arbetsområden i samhället.
Arbetet innehåller både hård- och mjukvaruutveckling vilket passar båda examensarbetares pågående utbildning.
Mjukvaruutveckling sker främst genom programmering i Arduinons programspråk (likt C++), för att få roboten att utföra funktioner och andra kommandon med hjälp av inköpta elektroniska komponenter.
Hårdvarumässigt har det tillverkats 3D-modeller i CATIA för att sedan 3D-printas för att användas på roboten. Det har även köpts in andra komponenter och hårdvara som t.ex. servomotorer och chassi som konstruerats. För att bevisa att resultatet uppnåtts har även en trappa i trä byggts.

Rapporten beskriver den bakomliggande teorin och användningsområden för de nödvändiga komponenter som används i arbetet samt tillvägagångssättet och implementering av dessa.
Rapporten innehåller de problem som har uppstått under arbetets gång samt våra lösningar till problemen och varför problemen eventuellt uppstod från början.

Nyckelord: Autonom, Robot, Hårdvara, Mjukvara, Arduino, Programmering, C++, CATIA, 3D-printing

Tacksägelse

Tack till Björn Bergholm som har tagit emot oss med öppna armar på Broccoli AB samt stöttat projektet ekonomiskt och med goda råd och tips. Björn har kommit med förslag och hjälp när projektet har stannat till och har alltid varit intresserad hur projektet ligger till.
Tack till Göran Hult som bidragit med tips när vi frågat om detta och kunnat erbjuda oss komponenter från Chalmers som behövts till arbetet.
Tack även till Niclas Jernberg som också gör sitt examensarbete på Broccoli AB som har hjälpt oss i startskedet att komma igång med hårdvaru-idéer samt hjälpt till att 3D-modellera i CATIA samt framställa dessa med hjälp av en 3D-printer.

Erik Rundberg
William Ripgården
Göteborg, juni 2020

Innehåll

Figurer

Tabeller

1
Introduktion

När det var tid att söka efter ett examensarbete var det för oss väldigt viktigt vad man faktiskt skulle göra och vilka områden som skulle fokuseras på. Efter många olika förslag så hittades ett intressant arbete (efter egna idéer) som skulle gå ut på att utveckla en robot som i någon mån ska kunna klättra upp för trappor autonomt bärandes en vikt. Detta var grundidén för arbetet som hypotetiskt sätt skulle kunna komma att ändras för att lägga till eller ta bort mål då en viss målbild inte kunde nås.
Tittar man på industrin och samhället överlag så kan man se hur automatiska system och robotar blir allt mer vanligt i vardagen vilket gör detta arbete extra intressant. Arbetet befinner sig i spannet av vår utbildning och det roliga med just ett sådant här arbete är att man utvecklar något från grunden och att man inte snöar in sig på en enstaka inriktning, utan i detta arbete får du arbeta med både mjukvara och hårdvara vilket gör arbetet brett och anpassningsbart i arbetslivet.
Broccoli AB är ett konsultbolag och tillverkar därför inga riktigt egna produkter. Deras största syfte med examensarbetet är att ha nära kontakt med studenter och ge dessa en uppgift som ska slutföras som en bra träning för framtida ingenjörsuppgifter. Denna chans vill Broccoli AB ge studenter då de intresserar sig av att ha arbetande ingenjörer från hela ålders- och erfarenhetsspannet.

Om man fokuserar på grundtanken att en robot ska kunna gå upp för trappor och bära en vikt autonomt så finns det en del användningsområden i dagens samhälle. Den skulle t.ex. vara nyttig för äldre människor som kanske behöver hjälp att bära tunga matkassar upp för sin lägenhetstrappa eller på samma sätt vara behjälplig för funktionshindrade med liknande problem. Eftersom roboten i slutändan kan svänga efter omgivningen kan även roboten fungera som en transport. Det är även lätt att omkonstruera roboten genom att applicera programkod på större motorer för att uppnå andra användningsområden.

1.1 Syfte

Då realiserandet av att problemet med en robot som enbart klättrade upp för trappor inte skulle fungera ändrades syftet tidigt i arbetet. Därför är det slutgiltiga syftet: **Att konstruera en robot som kan ta sig fram med sensorer och klättra i trappor eller över hinder. Tanken är att den utöver detta ska kunna bära en viss vikt och svänga med avseende på sin omgivning.**
Just vårt arbete innefattar att bygga en mindre prototyp, vilket inte gör den så användningsbar i nuläget förutom att Broccoli AB möjligtvis kan visa upp den på framtida arbetsmarknadsmässor, som ett exempel på vad man kan göra i sitt examensarbete.

1.2 Vad ska roboten göra?

Enligt planeringsrapporten var syftet att konstruera en robot som kunde känna av sin omgivning och klättra upp för trappor med en vikt på. Planen ändrades då alla mål inte blev uppnåbara. Därför ska roboten som är framtagen nu kunna ta sig fram autonomt med batterier, känna av väggar och svänga om sådana befinner sig för nära. Roboten ska även kunna klättra upp för en specialbyggd vriden trappa med ca 10cm höga trappsteg och väggar på sidan som visar att roboten kan svänga för att anpassa sig efter trappan. Även en accelerometer finns som används för att styra en motor beroende på robotens position i 3D-rummet. Motorn styr i sin tur en platta som alltid ligger vågrät, vilket man hypotetiskt kan lägga en vikt på som roboten då balanserar.

Mål	Förklaring
1	Klättra upp för olika riktiga trappor
2	Bära en vikt på 20kg och balansera denna
3	Känna av hur trappan är utformad för att anpassa sin klättring efter detta
4	Klättra autonomt

Tabell 1.1: De ursprungliga målen som sattes i början av projektet

1.3 Avgränsningar

Det finns en del problemställningar att ha i åtanke. Till att börja med kan detta arbete göras till ett extremt stort arbete, om man t.ex. skulle lägga till fjärrstyrning genom att ta fram en app. Även prisfrågor är svåra att förutse och därför ett problem när kostnaderna ska hållas nere i så god mån som möjligt. Priser tas därför hänsyn till, men får inte begränsa de optioner som finns vid inköp allt för mycket.
Även hur robotens slutgiltiga utseende skulle bli var ett underliggande problem och behöver därför avgränsas, då fokus låg på att den ska uppnå det syfte som satts och inte på hur tilltalande roboten är i slutet. Dess användarinterface kommer inte behandlas på det sätt att det måste se snyggt ut, utan det ska endast fungera. Hur energisnål den är kommer inte att behandlas då prioriteten ligger i dess prestation, och hur mycket energi roboten drar ska inte begränsa funktionaliteten på roboten.

1.4 Problemställning

På förhand är det svårt att veta vad som hinns med och inte, eller om man kan lägga till fler aspekter för att göra arbetet större. Det största problemet på förhand var att se hur den färdiga produkten skulle bli och fungera samt hur tillvägagångsättet skulle se ut. Hur ska den strömförsörjas? Vilka komponenter behövs? Hur kopplas elektroniken ihop?
Dessa problem är i det stora hela väldigt små problem, men som på förhand hypotetiskt kunde uppstå och göra det svårt att komma vidare i arbetet ifall man skulle fastna för länge på ett av dessa problem. Mer om problemlösningar kan läsas under **3 Utförande**.

2
Teori

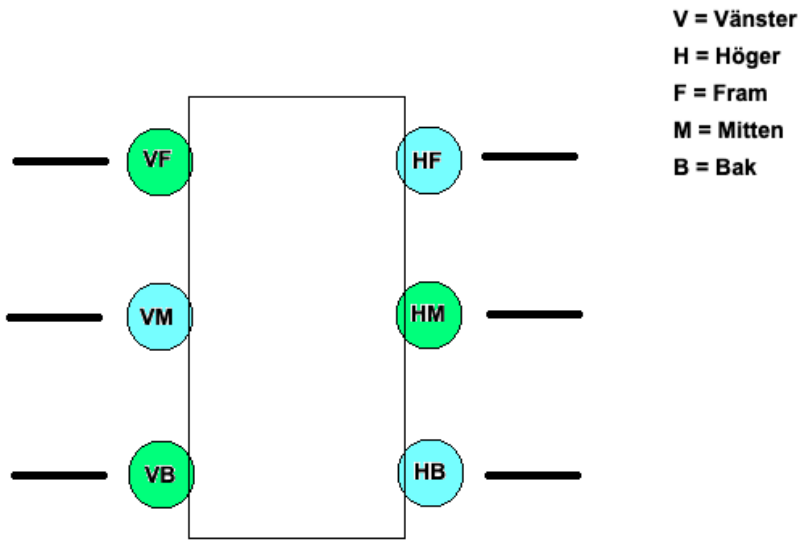
I detta kapitel beskrivs teorin bakom olika komponenter och annan kommunikation samt andra viktiga delar som används i projektet.

2.1 Mikrokontrollen Arduino

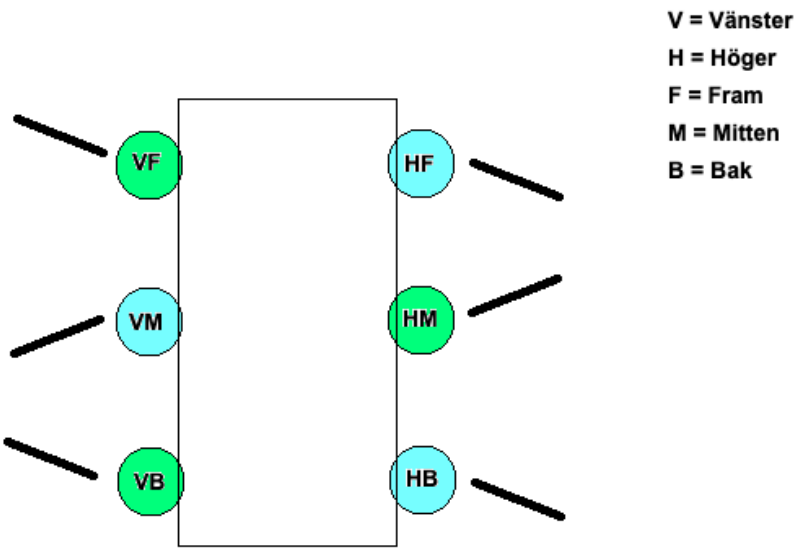
En Arduino är ett mikrokontrollerkort i ett hårdvaraformat. En Arduino har inget operativsystem och går inte att ansluta till en skärm. Den drivs antingen via en USB-port eller genom att ansluta ett externt batteri till Arduinons spänningsingång. Hårdvaran är relativt liten och billig och består av olika komponenter och portar för att kunna utöka hårdvaran för att t.ex. tända och släcka en lampa på en kopplingsplatta. Portarna kan agera ingång genom att läsa och bearbeta kod, via t.ex. ett utomstående knapptryck, för att göra om det till en utgång som kan starta en motor. Detta görs genom att programmera mikrokontrollen som sitter på Arduinon via en "sketch" som är namnet på programmen man programmerar Arduinon med. Denna sketch är skriven i programmeringsspråket C++ med några ytterligare specifika funktioner och metoder skapade just för Arduino. Det finns både digitala och analoga ingångar. De analoga ingångarna används till att mäta ett större spann av värden medan de digitala portarna används för att läsa eller skriva ett enstaka tillstånd, som t.ex. på eller av.
En fördel med Arduino är att man kan använda den på olika plattformar som exempelvis Windows, Macintosh OSX samt Linux. En annan fördel är att det finns många olika variationer av en Arduino med olika många portar, minne, storlek och pris. Det finns därför mycket valmöjligheter när det gäller att välja en Arduino som passar just ditt projekt.
Eftersom en kopplingsplatta och mycket annat kan kopplas samman med en Arduino kan Arduinon användas till väldigt mycket. Det har t.ex. gjorts projekt där industrier styr robotarmar med hjälp av Arduino, ett digitalt schackspel och mycket mer. Man brukar säga att en Arduino är ett verktyg för att styra elektronik [7].

2.2 Teorin bakom gångstilen

Bas- men också huvudfunktionen för roboten är att den ska kunna gå och röra sig framåt vilket var den första stora milstolpen som arbetades mot. För att kunna få roboten att röra sig framåt så måste servomotorerna programmeras så att benen kan röra sig synkroniserat tillsammans så att roboten kan “dra“ sig framåt samtidigt som den håller balansen.
För att lösa detta togs ett schema fram för hur benen skulle röra sig vid varje tidsstämpel och delade samtidigt upp de sex benen i två grupper. En grupp blev de tre ben som bildade en triangel tillsammans, alltså vänster framben, höger mittben och vänster bakben. Den andra gruppen är därmed höger framben, vänster mittben och höger bakben. Detta gör att när ena gruppen av ben lyfter från marken för att ändra sin position utan att flytta roboten, så kommer den andra gruppens ben hålla balansen på roboten då de står i en triangel-position och därav befinner sig robotens tyngdpunkt i centrum. En grupp rör sig synkroniserat medan den andra gruppen håller balansen. Bengruppen som lyfter upp benen kommer att vrida sig framåt i luften och sedan sätta ner benen i den främre positionen för att återfå balans; för att sedan lyfta det andra gruppen med ben för att upprepa rörelsen. Detta gör att roboten kan röra sig framåt under stabila förhållanden.



Figur 2.1: Illustration ovanifrån på robotens två grupperingar där ena gruppen är markerad i grönt och den andra gruppen är markerad i blått. Varje grupp rör sig synkroniserat



Figur 2.2: Illustration ovanifrån på hur robotens ben ser ut när den går framåt

2.3 Ultraljudssensorn HC-SR04

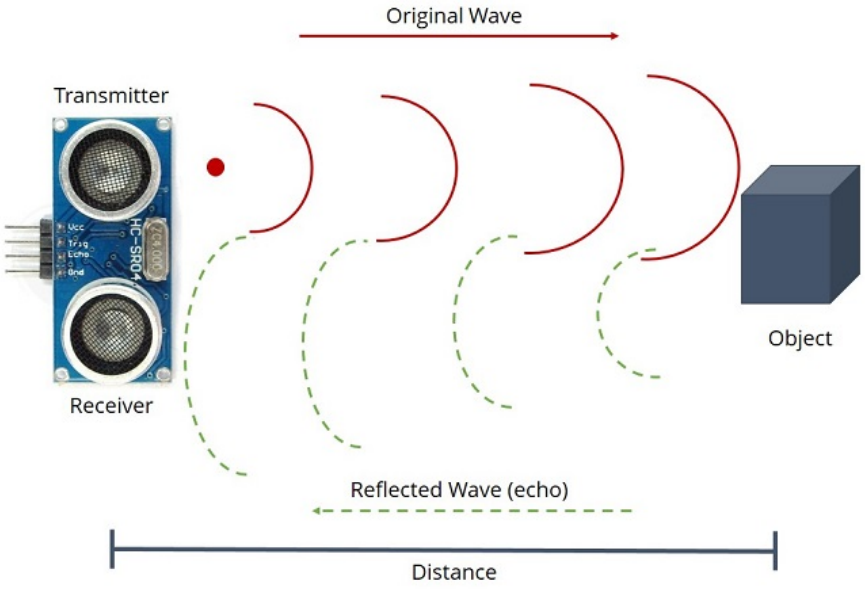
För att roboten ska veta när den ska sluta gå och börja klättra uppför trappan så behöver den en sensor för att bedöma avståndet till trappstegen. Det användes en ultraljudssensorn HC-SR04 då den var ett billigt sätt att mäta avstånd. Den ska även vara hyfsat pålitlig och är därför det bästa alternativet för avståndsavläsning till en Arduino. Ultraljudssensorn funkar så att den skickar ut ett ultraljud på ca 40 000 Hz som studsar mot ett föremål och sedan fångas upp av sensorn igen. Efter det så beräknas tiden från att ljudet skickades till att det togs emot vilket används för att beräkna avståndet.
Sensorn skickar ut ett ultraljud som rör sig med ljudets hastighet som är 340 m/s eller 0,034 cm/μs vilket är lättare att använda för att förklara hur sensorn fungerar [8]. För att beräkna sträckan används formeln $sträcka = tid * hastighet$ men då ljudet måste röra sig fram till hindret och sedan tillbaka så kommer tiden bli dubbelt så lång, därför halveras tiden i formeln för att få fram rätt avstånd. Sträckan utgör avståndet.

$$s = \frac{t}{2} * 340$$

I vårt fall är det rimligare att mäta avstånd i centimeter, därför används den här formeln där sträckan mäts i centimeter och tiden mäts i mikrosekunder.

$$s = \frac{t}{2} * 0,034$$

En nackdel med en ultraljudssensor är att den har problem med att beräkna avstånden till sneda ytor då ljudet inte studsar rätt tillbaka till sensorn vilket krävs för en korrekt mätning av avståndet. Dock kommer detta inte bli ett problem för oss då trappsteg alltid är vertikala eller nära till att vara vertikala vilket fungerar för vårt ändamål



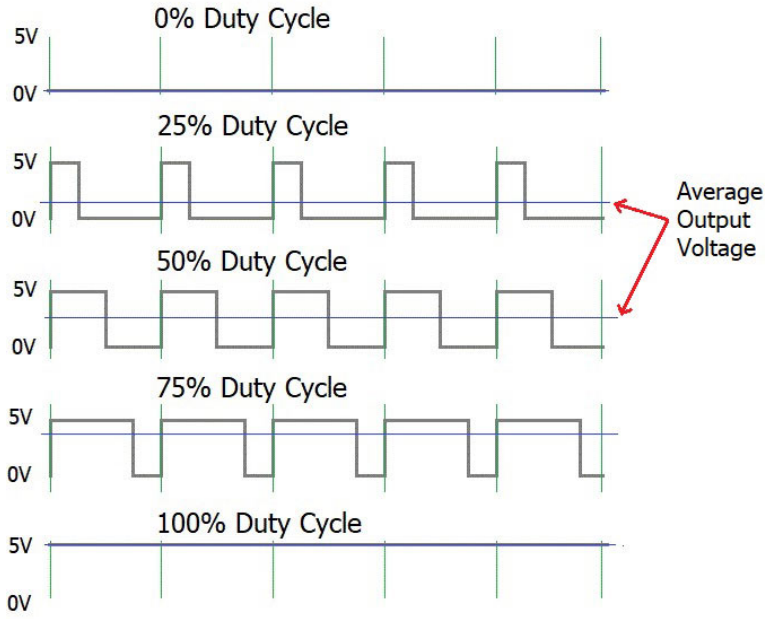
Figur 2.3: Illustration på hur ultraljudsensorn fungerar med avläsning av avstånd till objekt. [1]

Sensorn använder 4 pins för att fungera. Dessa är *Vcc, Trig, Echo* och *Gnd*. För att få ström kopplas *Vcc* till Arduinons 5V:s ingång och sensorns egna *Gnd* kopplas till den gemensamma jord. *Trig* och *Echo* används för att skicka och läsa signaler. För att generera en signal behöver sensorn först trigga en utsignal som kallas för *Transmitt*. Triggern i detta fall är det som heter *Trig* på sensorn. När man triggar utsignalen sänds *Transmitt* ut och modulen mäter hur långt det är fram till nästa hinder genom ultraljud. Så fort *Transmitt* har skickat färdigt sina signaler börjar modulen att räkna hur lång tid det tar för signalen att återvända till sensorn efter att den har hittat ett hinder och studsat tillbaka. Det är värdet (tiden) som modulen då räknat fram som man använder i formeln för att räkna ut det faktiska avståndet till hindret framför.

2.4 PWM - Pulse With Modulation

En PWM-signal är en metod för att generera en analog signal med hjälp av en digital källa. En PWM innehåller ofta två delar som påverkar signalens beteende. Dessa är frekvens och pulskvot. Frekvensen bestämmer hur snabbt PWM-signalen ska slutföra sin pulskvot och själva pulskvoten är tiden för signalen att vara aktivt hög (på) som procent av den totala tiden det tar för att slutföra pulskvoten.
PWM-signaler används för att kontrollera en mängd av olika applikationer. Vanliga användningsområden är styrning av pumpar, motorer eller hydraulik.
För att förtydliga hur PWM fungerar beskrivs ett exempel nedan:
Att skapa en signal som har medelvärdet 4V med en given digital källa vars signal antingen är hög (på) på 5V eller låg (av) på 0V, kan man använda en pulskvot på 80% vilket ger en utsignal på 5V 80% av tiden. Om den digitala källan är låg och 0V så kan medelspänningen beräknas genom att ta den maxspänningen multiplicerad med pulskvoten [9].

$$U, medel = 5V * 0,80 = 4V$$



Figur 2.4: Visar hur en olika stor pulskvot påverkar medelvärdet hos utspänningen. Bildkälla: [2]

2.5 Servomotorer

I projektet används tre olika servomotorer. Detta för att försöka hålla kostnaderna nere så mycket som möjligt. Det framtagna alternativet troddes på förhand fungera och ge tillräckligt mycket kraft till motorerna för att fortfarande lösa grundproblemet. För att styra en servomotor krävs det att en PWM-signal skickas ut med en viss pulsbredd som flyttar servon till en viss position beroende på hur lång pulsbredden är.

2.5.1 HS-645mg

Den första servomotorn som används heter HS-645mg. Detta är den starkaste av de tre olika servomotorerna. Servon används innerst och på mitten av benen för att styra den vertikala och horisontella rörelsen av varje ben. Motorn klarar av inspänningar på 4,8 - 6V där desto högre spänning som matas leder till att motorn drar lite mer ström men samtidigt orkar arbeta med lite tyngre vikter. Servomotorerna i projektet matas med 5V då de två regulatorerna som används för att ge ström båda matar ut 5V, samtidigt som att regulatorerna inte hade klarat av att strömförsörja alla motorerna med 6V då det hade behövts ytterligare mer ström.
En HS-645mg servo som matas med 5V ska klara av ett vridmoment på 7,7$kg \times cm$ och dra ca 370mA vid drift. [10]
Det används 12st HS-654mg i projektet då det var nödvändigt för att få önskad lyftkraft.

2.5.2 HS-422

Den andra servomotorn som används heter HS-422. Denna motor använder sig av samma inspänning som den tidigare nämnda motorn. Denna används ytterst på benen. En HS-422 servo ska klara av ett vridmoment på 3,3$kg \times cm$ och dra ca 160mA vid drift. [11] Det används i sin tur då 6st HS-422 i projektet eftersom det totalt används 18st servomotorer. Placering av motorerna på roboten syns i **Figur 2.5**.

2.5.3 HS-55

Det används 2st HS-55 motorer i projektet. Den ena motorn används för att styra en ultraljudssensors rörelse från vänster till höger som agerar “ögon“ åt själva roboten. Den andra motorn används för att styra en platta som ska hålla sig vågrät med hjälp av accelerometern ADXL345. Motorn ska klara av ett vridmomonent på 1,2$kg \times cm$ och drar ca 160mA. [12]. Till skillnad från de andra servomotorerna så drar dessa servos ström från 9V:s batteriet och inte det stora uppladdningsbara batteriet som styr motorerna.

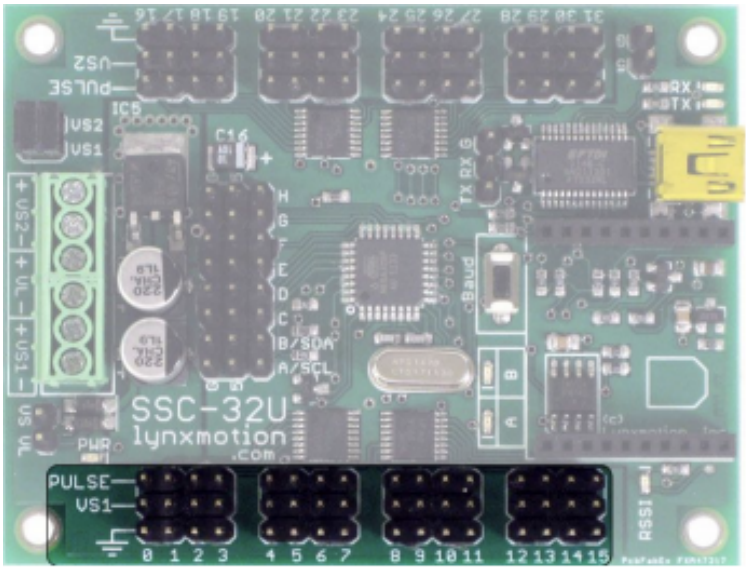
Figur 2.5: Illustration på hur motorerna sitter. Längst ut sitter den svagare motorn HS-422 och innerst sitter två HS-645mg motorer

2.6 Lynxmotion servokontroller SSC-32U

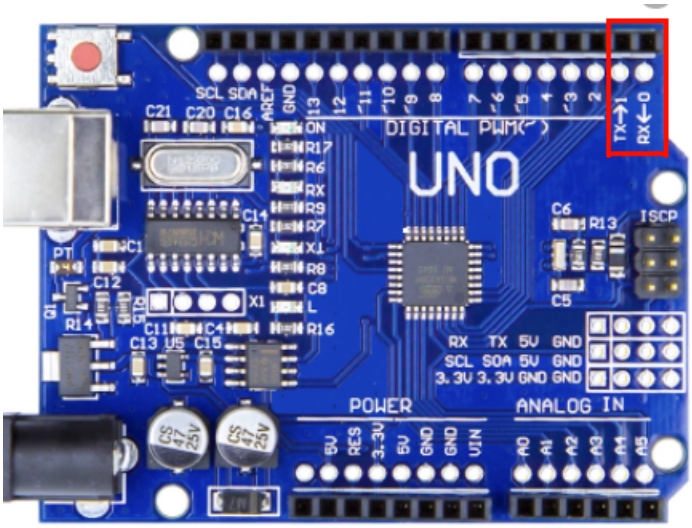
SSC-32U är en servokontroller som har i uppgift att styra servomotorer. I detta projekt kan en sådan användas då en Arduino Uno inte är utrustad med tillräckligt många pins/portar för att kunna styra 18 servomotorer och resten av sensorerna. Det har däremot SSC-32U:n som kan styra upp till 32 servomotorer [13]. Själva kortet är inte gjort för att programmeras. Istället har den i uppgift att ta emot och utföra kommandon från externa enheter, som t.ex. en dator eller en mikrokontroller. Att använda en servokontroller frigör minne på mikrokontroller som istället ständigt uppdaterar servomotorernas position.

2.7 Kommunikationen mellan Arduinon och Servokontrollern

Kommunikationen mellan Arduinon och SSC-32U sker via en speciell kod som hittas i en manual för SSC-32U som finns på Lynxmotions hemsida [3]. Man skriver koden i C++ som är Arduinons programmeringsspråk som sedan skickas till SSC-32U:n genom Tx/Rx portarna. Tx står för “Transmitt“ och Rx står för “Receive“. Det fungerar på samma sätt som när man skickar data från datorn till Arduinon. När ett program ska laddas över från datorn till Arduinon måste Tx och Rx kablarna vara frånkopplade från Arduinon, för att sedan sättas i igen när programmet har överförts.



Figur 2.6: En sida av SSC-32U med alla utrustade pins (0 - 15). Denna sida styr 9st servomotorer och får ström från VS1 [3]



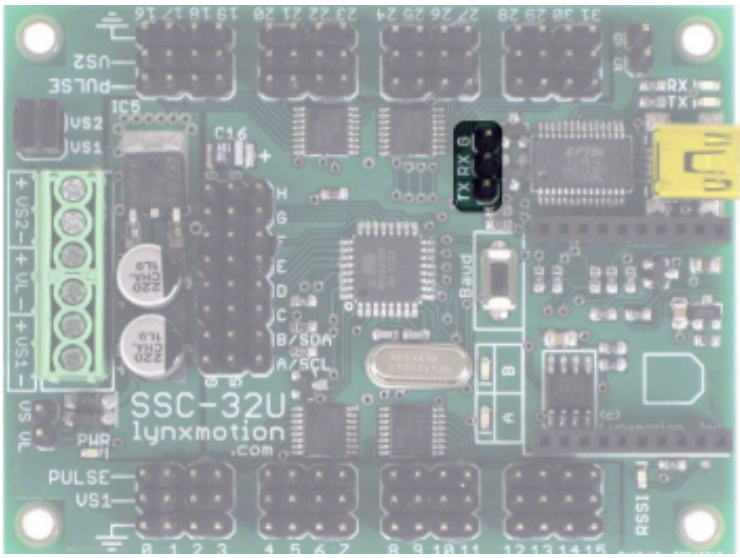
Figur 2.7: Arduinons Transmitt och Receive portar (Markerat i rött). Bildkälla: [4].

Själva koden är uppbyggt på ett speciellt sätt. För att styra en position måste man programmera enligt denna princip:
<ch> P <pw> S <spd> T <time>; där <xxx> betyder

- <ch>: Pin (ingång) till vilken servo som ska styras (0 till 31) decimalt
- <pw>: Önskad puls-bredd (normalt 500 till 2500) i mikrosekunder
- <spd>: Servorörelsens hastighet i mikrosekunder per sekund*
- <time>: Tid i mikrosekunder att ta sig från nuvarande position till önskad position

*1000μs av rörelse motsvarar 90° rotation. Ett hastighetsvärde på x μs per sekund

betyder att det tar $1000/x$ sekunder för servon att flytta sig 90°.

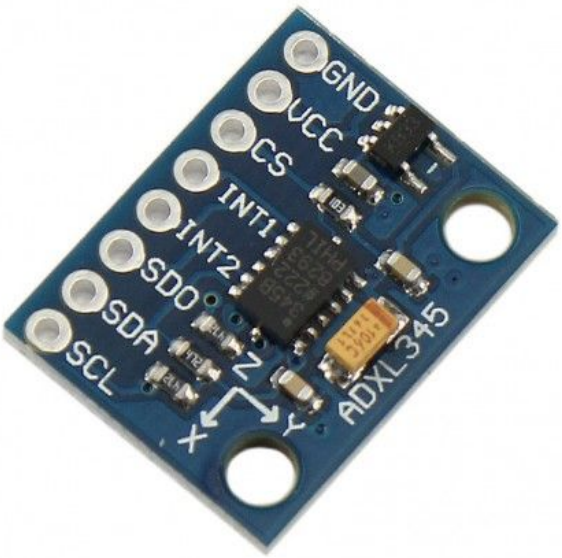


Figur 2.8: Visar Transmitt, Receive och Ground portarna som kopplas till Arduinons Transmitt, Receive och Ground [3]

Exempel: #5P1500T1000
Detta flyttar servomotorn som är kopplad till pin #5 till position 1500 på totalt 1000 mikrosekunder.
Hastigheten kan styras genom att antingen säga att den ska utföra rörelsen under en viss tid med kommandot T eller att den ska röra sig med en viss hastighet med kommandot S.
Spannet som servomotorn kan vrida sig enligt befinner sig i intervallet [500, 2500].

2.8 Accelerometer ADXL345

Accelerometern ADXL345 är en liten komponent som kan mäta statisk acceleration av gravitationen samt den dynamiska accelerationen som t.ex. en krock eller snabb rörelse skapar [14]. Den digitala utdata från ADXL345 skickas genom Arduinons "2-Wire" portar med 16-bit på tvåkomplementsform. Tvåkomplimentsform används i det binära talsystemet för att representera negativa och positiva tal, likt en *signed int* i programmeringsspråket C [15]. "2-Wire" kan till skillnad från de vanliga portarna använda I^2C som är seriell synkron multimasterbuss vars syfte är att koppla låghastighetsenheter till inbyggda system, moderkort och liknande elektroniska enheter. I detta fall för att koppla ADXL345 med Arduino UNO.
Accelerometern har 4st känslighetsnivåer från +/-2G till +/- 16G, vilket går att justera efter behov. Den kan således känna av rörelse i tre riktningar och därav veta var den befinner sig. Detta kan sedan användas för att skriva sin position i x, -y och z-led till Arduinon som sedan kan bearbeta dessa värden.



Figur 2.9: Bild på en ADXL345 [5]

2.9 Strömförsörjning

Strömmen som behövs för att styra Arduinon kan ske via en kabel direkt från datorns USB-port. Denna kabel står också för dataöverföringen mellan programmet på datorn till Arduinon. Man kan också försörja Arduinon med en valfri inspänning på 6V-20V via t.ex. ett batteri. Dock är den rekommenderade inspänningen till Arduinon 7-12V vilket togs hänsyn till. Den bakomliggande tanken är att styra logiken (Arduinon och alla elektronikkomponenter) med ett 9V:s batteri och att styra servomotorerna (genom SSC-32U) med ett annat batteri, då man är på den säkra sidan att alla komponenter får tillräckligt mycket ström. För att försörja alla 18 servomotorer behövs en del ström. Enligt deras datablad behövs en ungefärlig ström på 370mA på vardera HS-645mg motor samt 160mA på vardera HS-422 motor.
Eftersom SSC-32U har ingångar på båda sidor av hårdvaran och där hälften av motorerna är placerade på vardera sida måste varje sida kunna strömförsörja 9st servomotorer. Då en sida använder 6st HS-645mg servos samt 3st HS-422 servos krävs alltså en minsta ungefärlig ström på $(6 * 370mA) + (3 * 160mA) = 2700mA = 2,7A$ för att försörja en sida med motorer [10] [11]. Detta betyder i sin tur att det måste levereras minst 2,7A på varje sida för att få roboten att fungera med alla servomotorer samtidigt. Alla motorer styrs inte samtidigt, men 2,7A garanterar att alla motorer kan styras utan problem.

2.10 Chassi

För att hålla ihop alla delar på roboten behövs ett chassi med plats för ben och komponenter. Chassit som används har sex ben för att kunna förflytta sig med en stil som liknar en spindel. Chassit är av aluminium och kroppen består av två stycken plattor som skapar två nivåer så att 6st servomotorer ska kunna monteras fast mellan dessa plattor. Dessa motorer placeras så att rörelsen sker i sidled. På dessa servomotorer monteras sedan ben med ytterligare en servo monterad på dessa, som möjliggör att en rörelse kan ske i lodrätt riktning. Detta leder i sin tur att benen kan röra sig i 2 frihetsgrader.
Det finns modeller av chassin som bara tillåter rörelse i 2 frihetsgrader vilket fungerar för att roboten att gå framåt/bakåt och svänga, men då behovet för denna robot krävde att klättring ska kunna ske så behövs en rörelse i ytterligare en frihetsgrad. Därför har detta chassi en yttre benbit med plats för en till servomotor som möjliggör rörelser i 3 frihetsgrader. Genom att skjuta ut den yttre benbiten så kan benet nå mycket högre vilket hjälper när ett högre hinder ska överkommas. Den yttre benbiten gör det också mycket smidigare att hitta balansen i ojämn terräng.
Längst ut på varje ben sitter en gummiskoför att skapa friktion så benen inte glider runt på markytan.

2.11 3D-printing

I projektet används CATIA eller TinkerCAD Online (beroende på detaljens svårighet) för att framkalla önskade detaljer till spindeln, då CATIA används för mer komplicerade detaljer.
Tanken med 3D-printingen är att kunna uppfylla behov hårdvarumässigt för att få önskade funktioner att fungera så smidigt som möjligt. Eftersom projektet kräver sensorer ska det skapas detaljer till detta som ska konstrueras ihop med ett chassi som används för att hålla Arduinon och dess tillhörande komponenter på plats.
Chassit som håller Arduinon på plats sitter ovanpå roboten, vilket täcker alla kablar som är anslutna till servokontrollern. Ifall det skulle uppstå fel i dessa kablar är chassit skapat för att vara utdragbart, vilket möjliggör justering av servokontrollern och dessa kablar.

3
Utförande

3.1 Förstudier

Det började med att det fanns ett mål med arbetet som skulle arbetas mot att uppnå. De första veckorna skedde det egentligen bara forskning på området. Teleskopben eller några andra ben? Raspberry Pi eller Arduino? Vilket batteri fungerar att använda? Vilka motorer fungerar för att uppfylla målet att lyfta en vikt?

3.2 Beställningar

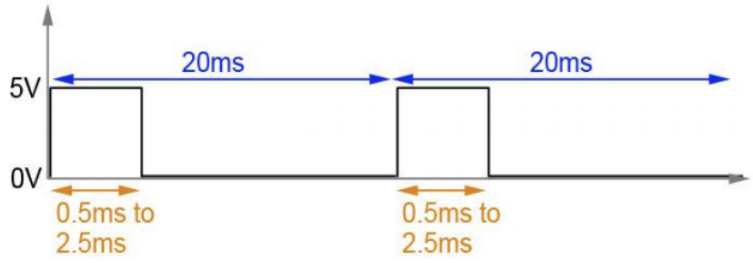
Det var många frågor från start som det behövdes svar på. När de flesta av svaren dykt upp beställdes det delar och elektronik för att ta oss vidare. När delarna dykt upp i form av ett chassi, motorer samt servokontroller byggdes alla ben ihop med motorerna för att kunna gå vidare i arbetet med fler tester gällande servomotorerna och strömförsörjningen av dessa.

3.3 Valet av mikrokontroller

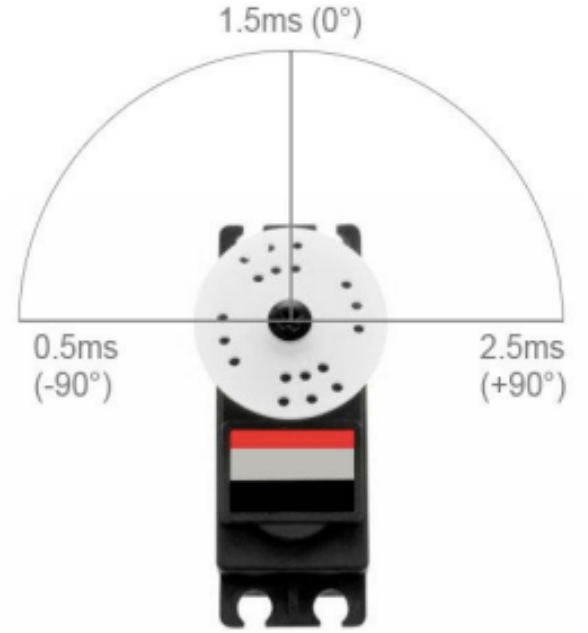
I början fanns det två val i valet av mikrontroller. Det ena var en Raspberry Pi och den andra var en Arduino. I början av arbetsgången så prioriterades Raspberry Pi då denna har fler portar och klarar av mer funktioner. För att jämföra en Raspberry Pi med en Arduino kan man säga att en Raspberry Pi är en Arduino fast med fler funktioner och är bättre att använda när du vill styra fler utomstående komponenter. Raspberry Pi:s grundfunktioner programmeras i programmeringsspråket Python medan Arduinons grundfunktioner finns i programspråket C++ [16].
Varför valdes det då att använda en Arduino?
Svaret på den frågan är för att alla extra funktioner på en Raspberry Pi inte skulle behövas. Samtidigt hade undersökningar om skillnader mellan dessa mikrokontrollers gjorts och påpekat att Arduinon skulle göra ett lika bra jobb. Även programmeringsspråken hade en del att göra i valet, då bekvämligheten med C++ är större än med Python. Arduinon är också något billigare och mindre än en Raspberry Pi.

3.4 Servomotorer

Servomotorerna fungerar genom att Arduinon skickar kod till SSC-32U som sedan skickar ut pulser till servomotorerna via PWM-signaler. Motorerna väntar alltså på att få en upprepad signal på 5V. Låter man sedan signalen vara aktiv (på en servomotor med ett 180° spann) i 0,5 millisekunder kommer motorn vrida sig till -90°. På en signal som varar 2,5 millisekunder vrider sig motorn till +90°, därför är 1,5 millisekunder lika med en 0° vinkel och därför centrum för servon. I projektet används dock servomotorer med ett 360° spann. Därför är maxläget på en signal som varit aktiv i 2,5ms lika med +180° samt -180° efter 0,5ms.
För att ta fram rätt värden till gångstilen så gjordes det upprepande tester för att hitta så bra värden som möjligt för en så stabil gång som möjligt.



Figur 3.1: Visar ett exempel på hur PWM-signalen till en motor kan se ut [3]

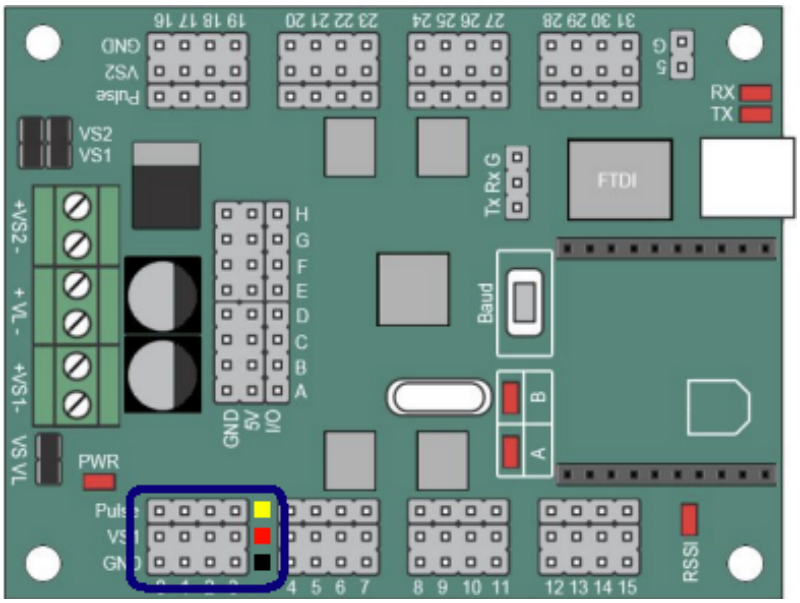


Figur 3.2: Visar hur vinkeln på motorn opererar beroende på signal på en motor med ett 180° spann [3]

Det finns rekommenderade restriktioner på hur länge en servomotor ska vara igång kontra vara avstängd. Rekommendationerna säger att en normal servomotor ska köras i maximalt 10 minuter; för att sedan “kylas“ av i 30 minuter. Detta togs givetvis hänsyn till vid körning och testning av roboten.

3.5 Servokontrollern SSC-32U

Denna användes som beskrivet i **2.6** för att styra servomotorerna eftersom Arduinon inte har tillräckligt mycket portar för att styra alla motorer. En servomotor är vid inköp kopplade till 3st 22AWG-koppartrådar (AWG = American Wire Gauge; $22AWG = 0,325mm^2$). Dessa 3 kablar är i färgerna svart, röd och gul. För att t.ex. koppla ihop en motor på port 1 på SSC-32U:n ska den svarta kabeln kopplas till jord, den röda kabeln kopplas till spänningsingången samt den gula till pulsingången.



Figur 3.3: Illustration var kablarna mellan servomotor till SSC-32U:n ska kopplas in. [3]

3.6 Strömförsörjning

Vid konstruktionen av roboten så har flera steg genomgåtts när det gäller strömförsörjningen. Till en början så sköttes strömförsörjningen bara genom en USB-kabel till Arduinon som också stod för dataöverföringen när testning av grunderna för olika komponenter. När det senare skulle testköras med servomotorerna behövdes ett batteri för att kunna ge ut en tillräckligt stor ström så att servomotorerna kan köras. För att detta batteri skall kunna styra servomotorerna så behöver det tillföra en spänning på 4,8-6V, men efter lite efterforskning upptäcktes det att inga vanliga batterier i butik har de spänningarna tillsammans med kapaciteten som behövdes.
Då tanken i början av projektet var att testköra motorerna och andra komponenter innan roboten kopplades ihop som en helhet, så köptes ett 9V litiumbatteri då det var möjligt att sänka spänningen och att en 9V kontaktadapter till en kopplingsplatta redan införskaffats.
För att sänka spänningen så servomotorerna kunde köras skapades en "spänningsnedsänkning" med hjälp av en spänningsregulator och två kondensatorer. För att konstruera en "spänningsnedsänkning" hittades till en början komponenter på Broccoli. Det som fanns var 2st 100μF kondensatorer och en **LM7805C** regulator. När spänningsregulatorn var byggd kunde testen genomföras på servomotorerna med 9V-batteriet. Det visades sig då att batteriet kunde strömförsörja och styra två ben (6st servomotorer) samtidigt.
Eftersom 9V:s batteriet endast klarade av att strömförsörja två ben behövdes ett nytt batteri köpas in. Därför gjordes dessa tester i början för att få en uppfattning om hur starkt batteri som behövdes. Efter vidare forskning köptes ett 7,2V 4000mAh NiMH-batteri (uppladdningsbart) som fortfarande kunde användas genom att sänka spänningen till 5V. Efter leverans av detta gjordes mer tester för att testa kapacitet i det nya batteriet.

Det som missades under vidare testning var värmeutvecklingen i regulatorn. Det nya 7,2V batteriet kunde inte prestera mycket bättre än 9V batteriet.
Efter konsultation med handledare Göran testades det att mäta strömmen efter regulatorn som gick upp till 1A för att snabbt sedan börja sjunka samtidigt som regulatorn blev mycket het. Det som hände var att vid testning med 7,2V in och 5V ut med 1A genom så blev förlusteffekten $P = (7,2 - 5)V * 1A = 2,2W$.

I teoriavsnittet kan man läsa att motorerna kräver minst 2,7A, men detta stämmer inte alltid i verkligheten då alla motorerna inte alltid är igång samtidigt med maximal ström. Därav gjordes tester med olika regulatorer som levererade olika ström ut.

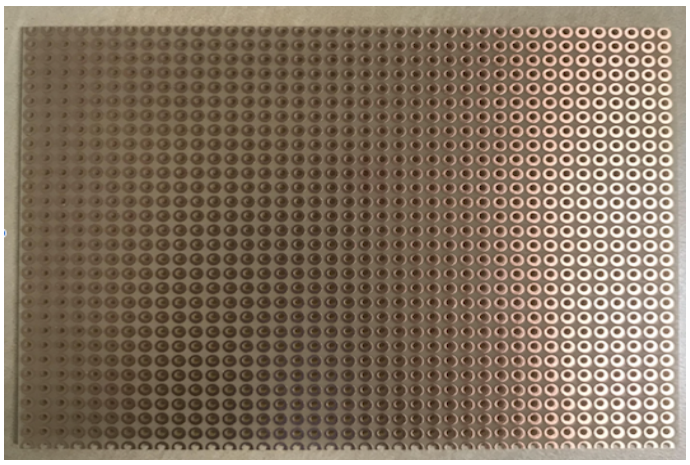
Det som missats i databladet för **LM7805C** var den termiska resistansen mellan "junction to ambient" som är 50°C/W, alltså en temperaturökning på 50° per watt. Då förlusteffekten var 2,2W blev $\Delta T = 50 * 2,2 = 110°C$. Temperaturen inne i regulatorn beräknas genom formeln nedan: [17].

$$T, reg = Omgivningstemperaturen + \Delta T$$

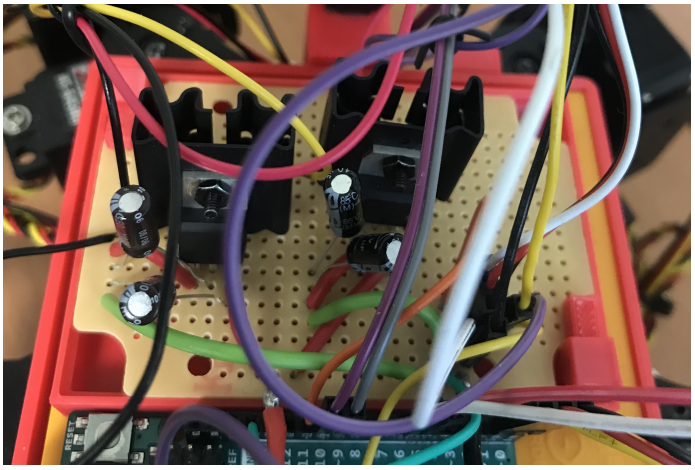
Den blev då $25 + 110 = 135°C$. Som en lösning på problemet föreslog Göran en **L78S05CV** kallad 5V-2A regulator som skall kunna tänkas lösa problemet. Skillnaden på den här regulatorn jämfört med den tidigare hittade regulatorn på Broccoli var att denna kunde leverera ut 2A istället för 1A. Vid test av dessa regulatorer var strömmen ut till motorerna fortfarande för låg. På samma sätt som ovan kunde förlusteffekten räknas ut. Den blev $P = (7,2 - 5)V * 2A = 4,4W$. Detta medförde att $\Delta T = 50 * 4,4 = 220°C$. Temperaturen inne i regulatorn blev då $25 + 220 = 245°C$. Detta var inte goda besked och arbetet stannades upp för att mer undersökningar skulle göras. Tillslut hittades två lösningar på problemet med för lite ström och för varma regulatorer. Den första lösningen var en kisel-diod som klarar inspänningar på upp till 600V samt en ström på 30A. Denna har ett spänningsfall på 1,5V vilket gör man hade kunnat få ut $7,2 - 1,5 = 5,7V$, vilket motorerna hade klarat av. Detta är dock inte testat då det även köptes in två stycken **LD1085V50 TO-220**-regulatorer som ger ut 3A. I det här läget var bekvämligheten större att använda regulatorer så de första testerna skedde med dessa. Det visade sig att problemet med strömförsörjningen äntligen löstes med dessa och roboten fick ström nog till att få roboten att gå. För att motverka överhettning på dessa regulatorer skruvades även en kylfläns fast på vardera regulator. För att **LD1085V50 TO-220** ska fungera så bra som möjligt används två kondensatorer på 10uF i kretsen enligt LD1085V50 TO-220 datablad [18].

Man tyckt att det borde räknats mer på ut-strömmarna innan vissa testningar. Problemet är att det är svårt att beräkna exakt hur mycket ström som pumpas in vid en exakt tidpunkt, då alla motorer inte är i drift samtidigt. T.ex. så fungerade det att styra 6 motorer med 1A men att styra 9 motorer med 2A gav problem.

Alla komponenter kopplades till en början från Arduinon till en kopplingsplatta. Fram mot slutet byttes denna kopplingsplatta ut mot ett experimentkort där kablar och komponenter löddes fast på undersidan. Detta hanterar våra höga strömmar utan problem till skillnad från en kopplingsplatta som endast är till för att hantera mindre strömmar.



Figur 3.4: Bild på ett experimentskort



Figur 3.5: Vårt experimentkort med tillhörande komponenter som kondensatorer samt regulatorer med kylflänsar

För att förtydliga vilka batterier som tillslut används styrs robotens alla ben med ström från det uppladningsbara 7,2V batteriet medan all annan logik som t.ex. Arduinon och sensorerna styrs med ström från 9V batteriet.

3.7 Gångstil & klättringsstil

När strömproblemet var löst skulle en gångstil tas fram, främst via testning. Under planeringsfasen togs en gångstil fram som strävades efter att efterlikna. Efter att en gångstil tagits fram så togs även en klättringsstil fram. Detta var ett tidskrävande arbete men vitalt för att roboten ska kunna utföra sina uppgifter.

3.8 Hårdvara

Tanken från början var att konstruera ett eget stort chassi med ben och kropp i CATIA för att sedan 3D-printa så att det gick att anpassa chassit efter servomotorerna. Det insågs snabbt att detta skulle ta väldigt lång tid så det letades även efter färdiga chassin på internet. Det hittades ett chassi i spindelformat som passade med servomotorerna. Detta kostade ca 2000kr och kändes därför som ett rimligt alternativ då detta också medförde en mycket högre kvalité än ett chassi gjort av plast. För att veta vilket alternativ som skulle gås vidare med jämfördes för- och nackdelar med de båda alternativen.
Nackdelarna med att göra ett eget chassi med hjälp av 3D-printing är jämfört med att köpa ett färdigt chassi:

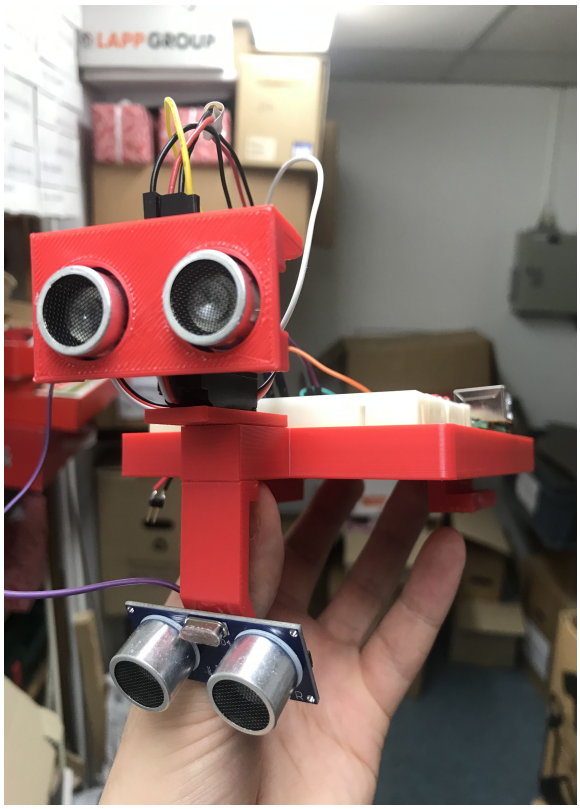
- Tar lång tid. Både att rita i CATIA samt själva 3D-printingen
- För stor del av projektet. Prioriteringen låg inte i att designa och ta fram ett eget chassi för en robot med sex ben, utan att programmera en Arduino med olika funktioner för att sedan styra motorerna och sensorerna på en robot
- Chassit skulle bli för ömtåligt då chassit måste kunna klara av vissa stötar och påfrestningar, och med den 3D-skrivaren som fanns tillgänglig så skulle inte plasten i chassit klara av att uppfylla dessa mål

Fördelar med att göra ett eget chassi med hjälp av 3D-printing är:

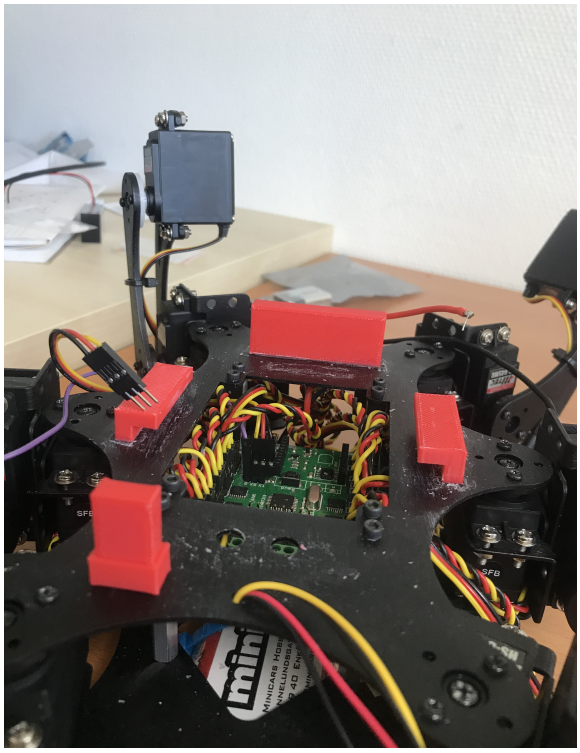
- Lätt att designa chassit efter våra ändamål, då det är vi som ritar
- Billigare

Efter att ha vägt fördelar med nackdelar om det skulle göras ett eget chassi eller köpa in ett färdigt så föll valet på att köpa in ett färdigt. Detta har som beskrivits i **2.9** ett spindelliknande format [19].
Efter leverans började chassit att byggas ihop. Det första som gjordes var att skruva fast benen och servomotorerna. Efter detta kan man säga att man “klämde“ ihop benen med två plattor som tillhör chassit. Alltså klämdes benen fast under- och överifrån. I mitten på chassit fanns det även rum för att skruva fast SSC-32U-kortet. Den sitter fast men ser ut att “sväva“ mitt i mellan den övre och den undre plattan. Detta gör att kablarna från servomotorerna slipper sticka upp och störa resterande elektronik som används samt att arbetsytan optimeras för att lättare kunna addera fler komponenter.

Det insågs också att det är bra om elektroniken som används till bland annat sensorerna (som sitter ovanpå spindelns översta platta), kan vara så samlad som möjligt. Därför skapades ett chassi till detta med en “slide in, slide out“ funktion som gör att man kan dra ut chassit ifall det skulle uppstå fel med kablarna från servomotorerna. För att skapa ett chassi med denna funktion gjordes två olika spår som passade in med varandra. Även en stoppkloss längst bak designades för att hålla chassit på plats. Till sist designades två tappar som agerar stoppklossar längst fram för att få maximal stabilitet på chassit.



Figur 3.6: Illustration på chassit som sitter på ovansidan. Sensorerna och Arduinon sitter fast på detta. Även ett av “spåren“ syns till höger



Figur 3.7: Illustration på hur SSC-32U sitter mellan den undre och övre plattan. Visar även de två “spåren“ på sidan samt stoppklossen där bak och en av tapparna där fram

3.9 Programmera HC-SR04

För att roboten ska vara autonom så krävs sensorer för att kunna känna av omgivningen. Ultrajudssensorn returnerar en integer (ett tal) som är ett mått på hur lång tid det tar för ett ultraljud att skickas ut och studsa tillbaka på en vägg eller ett hinder. Detta används sedan i koden i en loop för att jämföra avståndet med robotens position. Det används 2st ultraljudssensorer, som använder sig av de digitala portarna 5,6,9 och 10 på Arduinon. Dessa portar inkluderar PWM som används för att trigga en utsignal.
Detta görs i en while loop i programmet. När sedan ett visst avstånd nås så kallar programmet på olika funktioner som roboten utför. Det kan t.ex. vara att roboten ska gå rakt fram sålänge roboten befinner sig mer än 5cm från väggen eller ett hinder.

Till en början installerades en ultraljudssensor för att känna av eventuella trappsteg framför. Vid testning så hittades ett bra avstånd som sedan implementerades i koden för att samverka med klättringskoden. Vid detta laget kan roboten gå av sig själv rakt fram, känna av ett trappsteg och klättra upp för det.
När detta fungerade som tänkt skulle den sista sensorn sättas på. Denna har som uppgift att känna av väggar på sidorna. För att detta ska vara möjligt skapades en detalj i CATIA för att kunna sätta ihop en servomotor med sensorn, som gjorde att sensorn kan vridas och svepa rummet. Då svepningen var en komplicerad metod behövdes inspiration tas från internet. Det hittades ett liknande program på internet med idén om en roterande sensor där inspiration togs för att sedan skapa vår liknande funktion [20].

3.10 Accelerometer

Det sista som gjordes var att implementera en accelerometer som har i uppgift att känna av var den befinner sig i x-, y- och z-led. Denna ska sedan med hjälp av en servomotor hålla en platta i ett vågrätt skick som visar att roboten kan balansera en hypotetisk vikt. Det första som gjordes var att testa att endast köra accelerometern för att få fram värden i koden som beskrev accelerometerns position.

För att vinkla en platta när roboten ska klättra är det Y-koordinaten som behövs i detta fall. Y-koordinaten divideras med 256 då känsligheten för 2G används då denna är mest känslig för rörelser, vilket är bra då den uppdaterar sin position oftare. Värdet den får efter det befinner sig i spannet [-1,1] beroende på position. För att omvandla detta till ett användbart gradtal som motsvarar vinkeln på plattan, så multipliceras värdet med -90 så att värdet från acceleremotern befinner sig i spannet [-90°, 90°]. Att vinkeln multipliceras med -90 är för att invertera värdet från accelerometern så att plattans vinkel ska håller sig vågrät under klättring istället för att matcha accelerometerns vinkel. Efter det så adderas 90 till värdet så att graderna istället varierar från [0°, 180°] för att matcha plattans vågräta lutning på 90°.
Då roboten aldrig klättrar i en vidare hög lutning så spärras servons rörelse till att bara jobba inom intervallet [50°, 130°] (framtaget genom testning) genom 2st

if-satser. Efter det skickas det slutgiltiga värdet ut till servon som använder detta för att vinkla plattan.

När detta var färdigtestat skapades två 3D-detaljer för att fungera med servomotorn. Det skapades först en hållare till servomotorn som gör att den kan stå upprätt för att skapa höjd som behövs för att plattan inte ska slå i underlaget, samt en platta som ska limmas fast med servomotorn som då ska hållas vågrät. Då accelerometern använde ett annat sätt att skicka signaler (I^2C) hittades en grund på internet på hur man kunde läsa av accelerometern i I^2C-protokollet med hjälp av Arduinos Wire-funktion [21]. Dessa delar sattes fast på överdelen (taket) som har tagits fram via 3D-printing samt så löddes servomotorns Jord- och spänningskablar ihop på experimentkortet för att få ström. Den sista kabeln går till Arduinons port 3 som kan hantera PWM som behövs för att styra motorn. Snabbt efter detta gjordes koden om till en funktion som testades. När denna sedan fungerade lades funktionen till i klättra-funktionen vilket gjorde att lutningen på den balanserade plattan uppdateras när roboten befinner sig i sin klättringsfas.

3.11 Funktioner

För att roboten ska uppnå de mål som satts har det skapats funktioner i C++. Dessa är:

- Setup som ställer roboten upp från liggande läge
- Gå framåt
- Gå långsamt framåt
- Klättra
- Svänga höger/vänster
- Läsa avstånd till trappsteg + väggar
- Stanna
- Tilta en platta så den hålls vågrät
- Main-program

3.11.1 Gå framåt

Gå framåt-funktionen används som en basfunktion för roboten att ta sig fram och gå i normal takt. Själva funktionen gör att roboten tar två steg framåt.

3.11.2 Gå långsamt framåt

Gå långsamt-funktionen används för att sakta ner gångfarten om den befinner sig på ett avstånd mellan 16-24cm fram till trappsteget. Detta är för att gå framåt-funktionen tar större steg och riskerar att gå in i trappsteget om den befinner sig för nära trappsteget. Med gå långsamt-funktionen stöter man inte på detta problemet.

3.11.3 Klättra

Det finns en funktion för att klättra upp för trappsteg. Denna funktion blir aktiv när roboten befinner sig närmare än 16cm från ett trappsteg. Funktionen börjar med att ta några små steg framåt, för att sedan lyfta höger framben uppför trappsteget. Efter det tar roboten ett steg fram med alla ben, för att sedan lyfta upp vänster framben på trappsteget. Sedan går roboten i samma mönster med mitten- respektive bakben; med höger sida som lyfter först, tar sedan ett steg fram med alla ben, för att till sist lyfta upp vänster sida. När roboten befinner sig uppe på trappsteget hittar den balansen genom att smyga sig upp, likt strukturen i setup-funktionen.

3.11.4 Svänga höger/vänster

Roboten har en sensor som agerar ögon och sitter på en servomotor som agerar vridbart huvud. Detta gör att det skapats en funktion som kan se sin omgivning och svänga om det skulle vara en vägg eller liknande inom ett avstånd på 20cm. Sensorn kommer att åka från höger till vänster, och beroende på om det är en vägg på höger sida eller på vänster sida, kommer roboten att svänga bort från väggen tills att sensorn uppfattar att det inte är en vägg ivägen längre.
Exempel:
Det är en vägg på vänster sida. Roboten sveper sensorn från höger till vänster och uppfattar en vägg på vänster sida. Då svänger roboten höger tills det inte längre är en vägg ivägen och går sedan framåt förutsatt att det inte är ett trappsteg framför.

3.11.5 Avståndsavläsning

Två sensorer används för att läsa av avstånd. Dessa sensorer har varsin funktion som läser av ett avstånd 4 gånger, för att sedan dela de 4 värdena den får med 4 för att hitta ett medelvärde som sedan returneras och används i huvudprogrammet som en jämförare när roboten antingen ska gå framåt eller klättra. Detta görs för att förhindra att sensorerna skickar ut ett för lågt eller ett för högt värde, då sensorerna är ganska känsliga och lätt kan skicka fel värden.

3.11.6 Stanna

Det finns kod för att stanna och stå kvar på stället. Denna funktion har mest använts i början när mycket tester och likande gjordes. Den finns dock kvar då det kan vara användbar när man minst anar det.

3.11.7 Vinkla en platta som hålls vågrät

Funktionen jobbar till en början med ett inbyggt Arduino-bibliotek som heter Wire vilket är ett annat sätt än det vanliga att skicka och läsa av signaler på. Wire krävs för att arbeta med I^2C-applikationer, vilket accelerometern ADXL345 som används är, som förklarat i **2.12**. Funktionen hämtar ett värde från accelerometern för att sedan bearbeta värdet och skicka vidare värdet till servomotorn som styr plattan så att den hålls vågrät.

3.11.8 Main-program

Det finns ett stort main-program som sammanväver alla funktioner så att roboten styrs och gör sina funktioner i den ordning som krävs för att få den att utföra sina uppgifter.

3.11.8.1 Förklaring av main-programmet

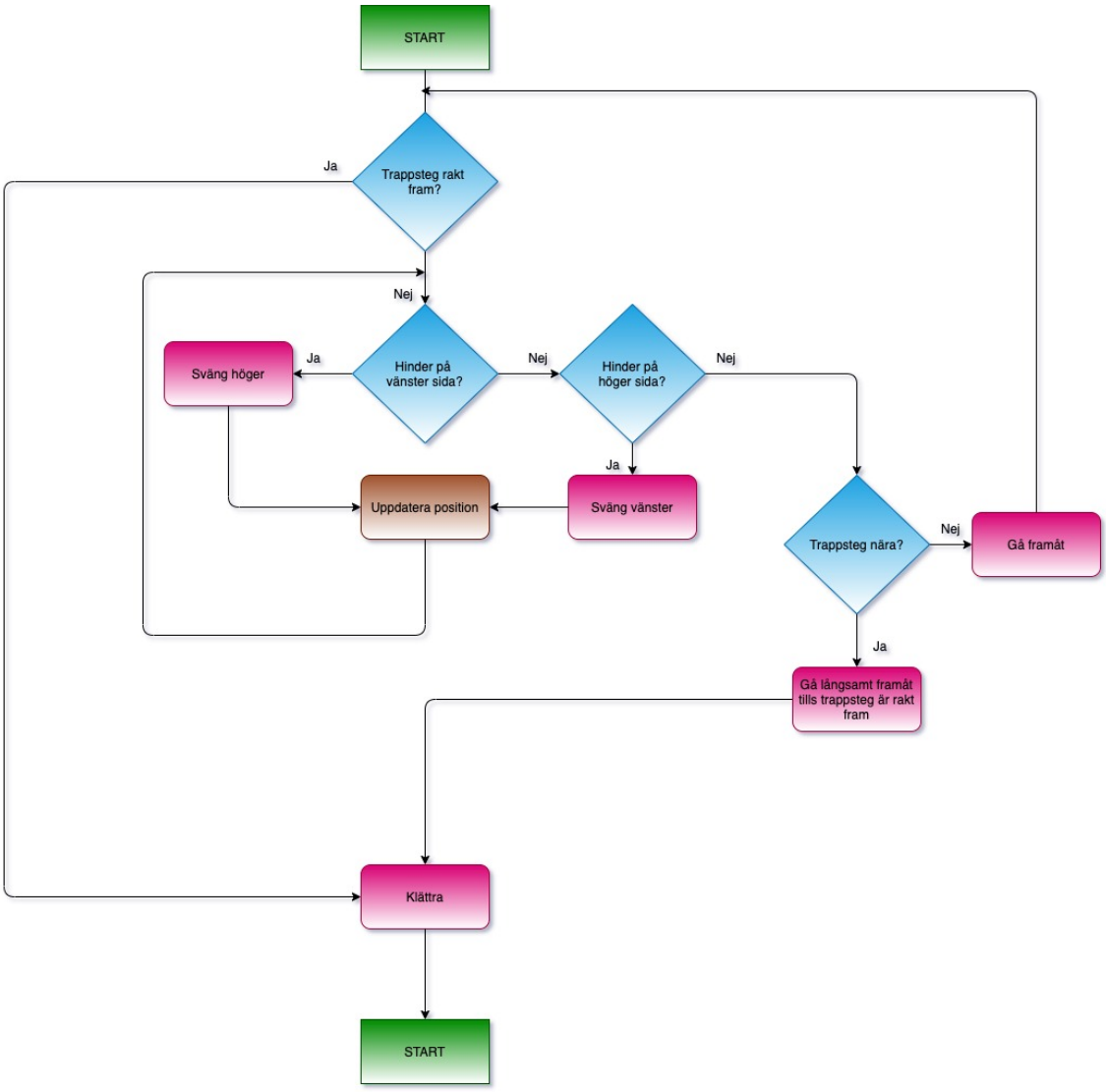
När main-programmet startar så börjar programmet med att jämföra om robotens undre sensor befinner sig längre bort än 16cm från ett trappsteg samt om en *Bool* som heter "loopOK" är *True* vilket den sätts till i initieringen. *Boolen* används på fler ställen längre fram i koden och används mest för att behandla eventuella buggar och breakouts ur loopar. Jämförelsen befinner sig i ett while-villkor som skapar en while-loop. Denna while-loop ligger som dominant för hela programmet och beroende på dess booleska uttryck kommer programmet utföra en rad olika funktioner och jämförelser. Det dominanta villkoret programmet utgår ifrån är alltså:
Sålänge den undre sensorn hittar ett värde som är större än 16 så kommer villkoret vara *True*.
Här nedan beskrivs vad som händer beroende på om det dominanta villkoret är *True* eller *False*.

- **Vid *False***:
  Om villkoret skulle vara *False* så kommer programmet hoppa förbi allt innehåll i while-loopen för att ta sig till en if-sats som blir *True* om villkoret som frågar om robotens nedre sensor befinner sig på ett avstånd mindre eller lika med 16cm (vilket den borde vara eftersom det första villkoret i början av programmet var *False*). Är villkoret *True* (vilket det borde vara om ingen felavläsning uppstått) kommer roboten att gå in i sin klättra-funktion och sätta "loopOK" till *True* för att försäkra att programmet ska kunna köra vidare från en början igen.
  Varför det finns en if-sats som dubbelkollar om avståndet till trappsteget är mindre eller lika med 16cm är för att sensorerna ofta kan skicka fel värden. Detta förhindrar att roboten börjar klättra när den egentligen inte ska göra det.
- **Vid *True***:
  Om villkoret däremot är *True* betyder det till att börja med att det inte finns ett trappsteg inom 16cm. För att få ett så precist avstånd till trappsteget som möjligt när roboten väl ska klättra används en funktion att gå långsamt. Det programmet då undersöker är om robotens undre sensor befinner sig på ett avstånd mer än 16cm bort från ett trappsteg men samtidigt mindre än 24cm ifrån samma trappsteg. Detta sker via en if-sats 2 gånger, även här för att försäkra sig att inga felvärden fås från ultraljudssensorerna. Om det villkoret är *True* kommer roboten att kalla på sin gå långsamt-funktion för att gå långsamt. Om istället villkoret är *False* vilket innebär att roboten befinner sig på ett avstånd mer än 24cm ifrån ett trappsteg, kommer roboten att kalla på sin gå framåt-funktion som gör att den går framåt i vanlig takt. *Programmet befinner sig fortfarande i den dominanta while-loopen vid detta laget.*

Efter detta kommer robotens översta sensor mäta avstånd till eventuella väggar eller andra föremål. Det gör den via en funktion som sveper rummet från vänster till höger genom en for-loop som adderar 2° (vilket gör att servon vrider sig) för varje gång koden kommer tillbaka till for-loopen efter att ha kört resterande innehållande kod *om* "loopOK" är *True*. "loopOK" finns med i villkoret för att programmet ska ha en chans att kunna bryta sig ur loopen med anledning att inte programmet ska ligga och köra en evighetsloop under for-loopen.
Skulle "loopOK" vara *False* kommer hela for-loopen att avbrytas och hoppa förbi all annan kod ner till en tilldelning där "loopOK" blir *True* igen. Detta för att det är det enda platsen (förutom efter att den kört klättra-funktionen) i koden där "loopOK" sätts till *True* igen efter att blivit *False*. Tilldelningen ligger på en plats i koden där programmet kört färdigt sina funktioner med att svänga eller gå samt innan klättra-funktionen.
Skulle dock "loopOK" vara *True* så kommer programmet återigen till en if-sats som undersöker om den undre sensorn är mindre eller lika med 16. Skulle avståndet vara mindre eller lika med 16 (alltså att if-satsen är *True*) kommer "loopOK" sättas till *False* och därav hoppa ur for-loopen. Skulle if-satsen dock vara *False* (vilket den borde vara) händer ingenting och programmet kommer gå vidare till en while-loop med villkoret som blir *True* om den övre sensorn mäter att roboten befinner sig på ett avstånd mindre än 20cm från en vägg eller liknande. Därefter går koden vidare för att jämföra i vilket läge den övre sensorn befinner sig i sitt svepande. Med hjälp av sin position på den övre sensorn avgör programmet om det är ett hinder på höger eller vänster sida och svänger därefter tills att avståndet till väggen blir 20cm eller mer. Är det inget hinder på varken höger eller vänster sida kommer roboten istället att gå framåt.
På samma sätt som sensorn svepte rummet från höger till vänster kommer sensorn att svepa från vänster till höger. Samma process som användes när sensorn svepte från höger till vänster anpassas sedan igen.
När programmet har kommit såhär långt har den kört igenom all kod som innebär att svänga och gå framåt. Programmet hamnar även nu på platsen där tilldelningen av "loopOK" sätts till *True*. Eftersom all kod med svängning och gång har körts igenom hoppar även programmet ur while-loopen som är dominant. Vid den här tidpunkten har alltså while-loopen körts en gång.
Här hoppar sedan koden upp till början av main-loopen för att jämföra det avståndet från den undre sensorn till ett trappsteg igen och gå igenom samma långa process igen.

3.11.8.2 Flödesschema

Nedan visas en väldigt förenklad modell av programmets uppbyggnad. De gröna områdena är main-programmets start-loop. De blåa områdena beskriver frågor som programmet ställer via if-satser eller for-loopar. De cerise områdena beskriver rörelser som roboten utför samt det bruna området är en positions-uppdatering.



Figur 3.8: Bild på programmets flödesschema

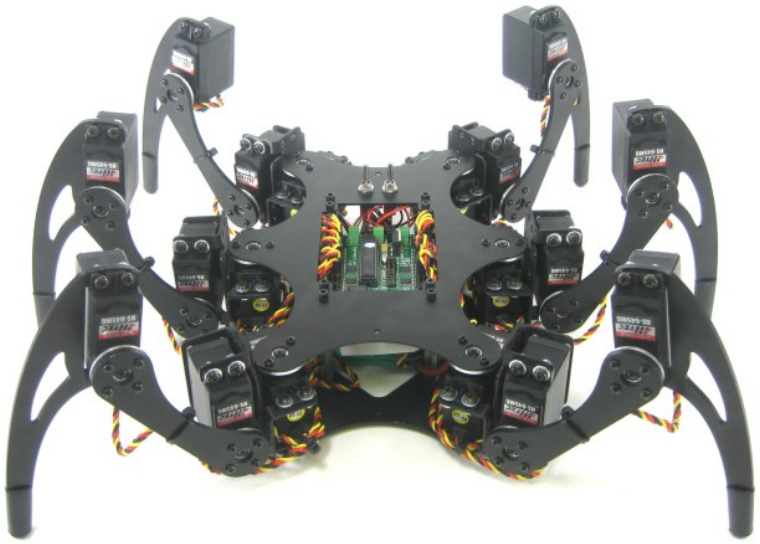
3.12 Elkretsschema

Kretsschema över använd elektronik finns under **Bilaga C**.

3.13 Hårdvara

Här finns information av den slutgiltiga hårdvara som används bortsett från servomotorerna och komponenterna.

3.13.1 Spindelchassi



Figur 3.9: Bild på chassit tagen från tillverkarens hemsida [6]

3.13.2 3D-printing

Bilder på alla 3D-figurer som skrivits ut med hjälp av en 3D-skrivare finnes under **Bilaga A**.

3.13.3 Trappkonstruktion

Det byggdes en trappkonstruktion med vridbart trappsteg med väggar som gör att roboten kan klättra upp för ett trappsteg samt känna av omgivningen i trappan och på så sätt kunna veta om den ska svänga höger eller vänster för att klättra upp för nästa trappsteg. Trappstegets storlek: 53 x 43 x 10 (cm) (Bredd x Djup x Höjd).

Figur 3.10: Bild på egenbyggd trappa med vridbart trappsteg

3.14 Kod i C++

Bilder på all kod finnes under **Bilaga B**.

4
Resultat

Syftet från början var att få en robot som kunde klättra upp för trappor. Med tanke på att benen är för korta och chassit är för litet för att klättra i riktiga trappor justerades detta mål relativt tidigt i arbetet. Istället ställdes nya mål upp, som t.ex. att klättra för ca 10cm höga trappsteg samt kunna svänga vid eventuella väggar.

Tillagda Mål	Förklaring
5	Känna av väggar och svänga enligt dessa
6	Gå framåt och känna av när hinder/trappan börjar autonomt

Tabell 4.1: Tillagda mål

Mål	Uppnått mål	Avvikelse
1	Delvis	Ej riktiga trappor, utan egenbyggd vriden trappa, 10cm höga trappsteg
2	Delvis	Kan bära nuvarande vikten på 2,2kg. Kan balansera en platta
3	Delvis	Kan inte känna av höjd på trappan. Kan svänga i trappan
4	Ja	-
5	Ja	-
6	Ja	-

Tabell 4.2: Alla mål och om de har uppnåtts samt avvikelser

Resultatet av projektet är en robot som delvis uppfyller 3 av de 4 ursprungliga målen samt uppfyller det resterande ursprungliga målet. De 2 tillagda målen uppfylls även. Huvudmålet som var att få den att klättra uppför en trappa samt bära med sig en vikt har uppnåtts. Roboten väger totalt ca 2,2kg.

Mål 2 är ett mål som ej exakt är testat vad roboten klarar av att bära. Detta då ytan för en extra vikt inte fanns och därmed begränsade testning av vad den klarar av att bära.

5
Slutsats

Här beskrivs och diskuteras de största och svåraste problem som stötts på under arbetets gång.

5.1 Det ursprungliga målet & nya mål

En bit in i arbetet förstods det att det ursprungliga målet att klättra upp för trappor inte kommer vara fullt möjligt så den ursprungliga planen ändrades under arbetets gång. Varför det inte var fullt möjligt är för att robotens chassi helt enkelt är för litet. Benen är för korta och motorerna för svaga. Den hade i teorin kunnat gå upp för riktig trappor om man köpt in bättre servomotorer och större chassi, men detta hade blivit väldigt dyrt.
Då blev det en ny plan som innebar att roboten ska kunna känna av sin omgivning och svänga höger och vänster om den upptäcker väggar, samt klättra upp för mindre trappsteg (ca 10cm).
Ett annat nytt mål var även att implementera en accelerometer som känner av sin position i rummet. Tanken med detta är att med den information sedan styra en servomotor med en tillhörande platta som alltid håller sig vågrät för att hypotetiskt kunna lägga på en vikt. Detta var då för att försöka få med ett av grundmålen i arbetet som var att lyfta en vikt, även om roboten redan lyfter en vikt i form av batteriet. Den totala vikten på roboten är ca 2,2kg, vilket roboten kan fungera med. Det är oklart vad roboten faktiskt klarar av att bära, då som beskrivet innan, yta på roboten inte fanns för att testa detta.
Därför skulle det som helhet underlättat att redan innan arbetet startade haft en större inblick i hur stor roboten hade behövt vara för att klara att gå upp för vanliga trappor och hur starka motorer det hade krävts. Även kanske mer prisundersökningar kunde gjorts för att försöka hålla ner kostnaderna ännu mer.

5.2 Strömförsörjning

Detta var det första och även mest tidskrävande problemet. Det var först i vecka 4 som alla ben kunde testas då det krävts förstudier och inköp innan detta. Det tog sedan ytterligare 4 veckor att testa, forska vidare och köpa in komponenter som kunde hjälpa oss att lösa problemet med strömförsörjning. I efterhand skulle det gjorts mer mätningar och mer undersökningar på strömmen och servomotorerna. Då hade en klarare bild fåtts om hur mycket ström som faktiskt behövs. I början var inte strömförsörjningen en sådan stor del då montering samt programmering var

i fokus, men strömförsörjningen var viktigare och svårare än förväntat. Steget från att inte få tillräckligt med ström till att alla motorer får ström var ett väldigt viktigt steg för att kunna gå vidare i arbetet, så det skulle lagts mer fokus på att få det att fungera från början.

5.3 Gång- & Klättringstilen

När strömproblemet var löst kunde äntligen gångstilen testas. Teorin bakom gångstilen var redan färdig och noga framtagen. Problemet var att det var svårt att översätta teorin bakom gångstilen till den faktiska programmeringen då servokontrollens egna språk användes. När förståelse om programmeringen alstrades så togs en gångstil fram. Snabbt efter detta började testningar göras för att få roboten att klättra. Här krävdes det bara testning för att hela tiden göra små-förändringar i koden så att roboten fortfarande står stabilt medan den har några ben uppe på trappsteget. Det tog lång tid att utveckla en klätterstil men efter veckor av testning fungerade det tillslut.
Eftersom det tog så mycket mer tid än förväntat blev det mindre tid för att implementera andra nödvändiga funktioner, som t.ex. accelerometern som implementerades i sista stund.

5.4 Sensorerna

Sensorerna har en viktig del i att det är beroende på avstånd till vissa väggar eller hinder som roboten utför andra funktioner. Sensorerna är därför vitala för roboten ska gå autonomt. Vid testningar av endast en sensor fungerade det utan problem. Eftersom det dock används två sensorer var tanken att sensorn mäter avstånd till trappsteg ska vara dominant, och så fort den upptäcker ett trappsteg ska den börja klättra. Riktigt så enkelt var inte fallet då efter vidareforskning det visade sig inte fungera att ha igång två sensorer samtidigt. Därför lades otroligt mycket tid ner på att programmera förbi detta för att försöka undvika detta problem.
Det löstes bland annat genom att skapa en flagga med datatypen *Bool* som används och måste vara *True* för att programmet ska gå in i en loop. Loopen i fråga är det som i början avgör om roboten ska klättra eller gå. När flaggan är *True* och avståndet till trappsteget är större än 16cm blir villkoret sant och programmet går in i loopen där den vidare kan läsa av väggar eller trappsteg. Snart efter detta dubbelkollas avståndet till trappsteget igen; skulle det vara så att trappsteget finns inom räckhåll för att börja klättra kommer flaggan sättas till *False* vilket gör att programmet kommer hoppa ur sin aktiva loop för att gå över i klättringsfunktionen. *Boolen* används också för att hindra den ena sensorn från att läsa av samtidigt som den andra när det kommer till programmets villkor.
Även om både studenterna har haft kurser i programmering skiljer det sig från Arduino-programmering. Språket i fråga är relativt likt men det svåra är att implementera alla komponenter då de behövs programmeras på olika sätt. Hade man haft mer erfarenhet av komponenterna i fråga hade man haft lättare att förstå hur de skulle fungera, och inte behövt lägga lika mycket tid på att lära sig just det.

5.5 Reflektion

Hade man kunnat göra någonting annorlunda?
Ett stort problem under projektets gång har varit önskan att köra flera processer samtidigt. Roboten kan t.ex. inte gå samtidigt som den läser av väggar, vilket hade varit bättre än den nuvarande situationen där roboten måste stanna för att sedan läsa av väggar. En Arduino kan egentligen bara göra en sak åt gången men det finns sätt att kringgå detta (t.ex. med sensorerna) vilket har använts till fullo för att uppfylla robotens uppgifter så bra som möjligt. Ett annat sätt hade varit att köra processen via två Arduinos samtidigt vilket gör att två processer kan köra samtidigt. Då hade man kunnat dela upp de olika funktionerna på vardera Arduino så att styrningen av benen sker via en Arduino, och sensorer och andra komponenter körs via den andra. Det hade till exempel vart möjligt att köra den roterande sensorn som känner av omgivningen samtidigt som roboten går framåt istället för att köra dessa processer seriellt.

Ett återkommande problem som uppstår sällan men för lite ofta för att kunna ses som en liten bugg är att processen hänger sig mitt i körningen för att några sekunder senare “resetta“ och sedan fastna i en reset-loop. Till en början låg misstankarna att koden låste sig i en loop som den inte kunde bryta sig ur men vid användning av “serial monitor“ som printar ut varje kodrad som skickas från Arduinon upptäcktes det att “Serial.print“-funktioner bröts mitt i, så att bara halva meddelandet skrevs ut vilket inte ska kunna hända. Tyvärr har ingen lösning på detta problem hittats än men det hade eventuellt gått att lösa om mer tid hade funnits tillgodo och då andra delar i projektet inte prioriterats över.

De första målen som sattes fick tyvärr justeras då djupare kunskap inom projektet skaffades och då kostanden för komponenter skulle bli allt för dyra för att klara av målen. Då priserna för chassi och motorer ökade exponentiellt med storleken hittades en gräns där priserna inte blev för höga samtidigt som målen kunde uppfyllas till viss grad. T.ex. målet med att bära med sig en 20 kg vikt uppför trappan justerades om till att bära med sig sitt 600g NiMH-batteri samt ha en lastplatta som balanserar och håller sig vågrät genom hela klättringen för att simulera att en vikt bärs med och samtidigt hålls stabil ovanpå roboten. Detta mål kändes rimligare och mer genomförbart utan att lägga för mycket pengar.

Efter genomfört projekt har många nya erfarenheter och kunskaper erhållits. Då mycket av det som arbetet innefattade har varit abstrakt på förhand för att slutligen se en färdig prototyp ses resultatet som lyckat. Även om programmeringsdelen var hyfsat lik tidigare kurser så har problemlösningsförmågan satts på prov ett flertal gånger, både gällande implementering av komponenter och hårdvara.
En väldigt stor och bred praktisk kunskap har fåtts, då många olika moment som är bra att ha med sig i framtiden har använts som t.ex. koppling av elektronikkomponenter, programmering, hårdvaruutveckling, strömförsörjning samt problemlösning.

Litteraturförteckning

[1] Rui Santos, "Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino," https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/.

[2] A. Raj, "What is PWM: Pulse Width Modulation," https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation (2018/09/19).

[3] Lynxmotion, "Lynxmotion SSC-32U USB Servo Controller Board," http://www.lynxmotion.com/images/data/lynxmotion_ssc-32u_usb_user_guide.pdf.

[4] "UNO R3 Board CH340 For Arduino," https://kuongshun.com/products/uno-r3-board-ch340-for-arduino.

[5] "ADXL345 Digital 3-axis acceleration of gravity tilt module," https://electropeak.com/accelerometer-sensor-adxl345-digital.

[6] "Isometric image," http://www.lynxmotion.com/images/jpg/phoenb1.jpg.

[7] Arduino, "What is Arduino?" https://www.arduino.cc/en/guide/introduction.

[8] Dejan, "Ultrasonic Sensor HC-SR04 and Arduino Tutorial," https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/.

[9] J. Heath, "PWM: pulse width modulation: What is it and how does it work?" https://www.analogictips.com/pulse-width-modulation-pwm/ (2017/05/04).

[10] "HS-645MG Servo," https://www.servocity.com/hs-645mg-servo.

[11] "HS-422 Servo," https://www.servocity.com/hs-422-servo.

[12] "HS-55 Servo," https://www.servocity.com/hitec-hs-55-servo.

[13] “Lynxmotion SSC-32U USB servo controller,” https://www.robotshop.com/en/lynxmotion-ssc-32u-usb-servo-controller.html?gclid=Cj0KCQjwzZj2BRDVARIsABs3l9IR-NHACKrlIVb9_aU8r34ymCfJ_WBRWTYGsxejq5oSfLyfeGPjRC8aAgIHEALw_wcB.

[14] Analog Devices Inc, “Digital Accelerometer,” https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf.

[15] D. M. H. Sarah L. Harris, “Two’s complement number,” https://www.sciencedirect.com/topics/computer-science/twos-complement-number (2016).

[16] “What are the differences between raspberry pi and arduino?” https://www.electronicshub.org/raspberry-pi-vs-arduino/(2017/12/06).

[17] R. Kleim, “Thermal design with linear voltage regulators,” https://www.allaboutcircuits.com/technical-articles/thermal-design-with-linear-voltage-regulators/ (2016/03/10).

[18] ST Microelectronics, “3 A low drop positive voltage regulator: adjustable and fixed,” https://www.electrokit.com/uploads/productfile/41012/CD00001883.pdf (2013/10).

[19] “Lynxmotion Phoenix 3DOF Hexapod - Black (No Servos / Electronics),” https://www.robotshop.com/en/lynxmotion-phoenix-3dof-hexapod---black-no-servos---electronics.html.

[20] Makerfactory, “Robobug / hexapod walking and avoiding obstacles,” https://www.servocity.com/hitec-hs-55-servo (2019/09/12).

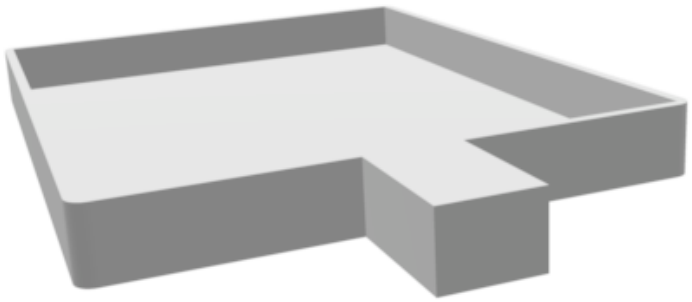
[21] Dejan, “How To Track Orientation with Arduino and ADXL345 Accelerometer,” https://howtomechatronics.com/tutorials/arduino/how-to-track-orientation-with-arduino-and-adxl345-accelerometer/.

A

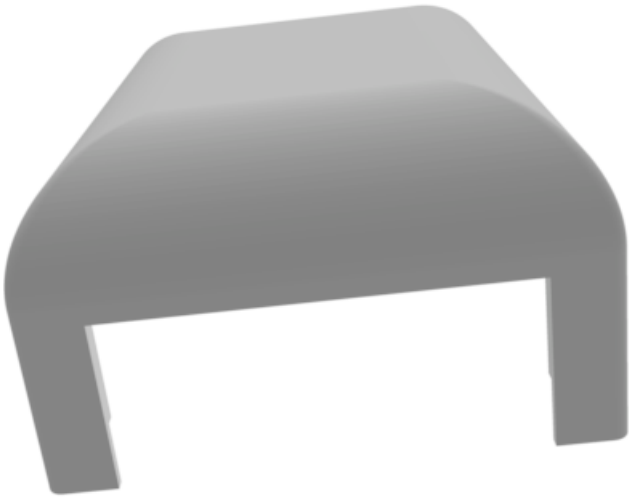
Bilagor 3D-printing

Här finns bilagor på 3D-figurer som tagits fram i arbetet.

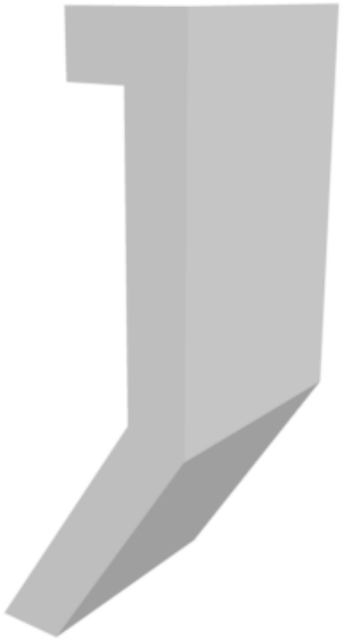
A.1 Underdel



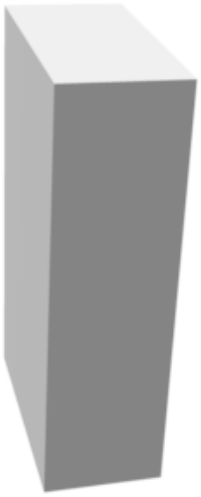
A.2 Överdel



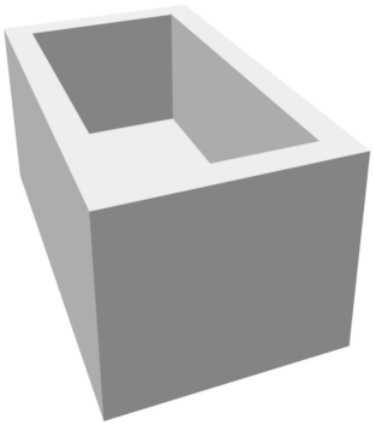
A.3 Hållare till nedre sensorn



A.4 Tapp

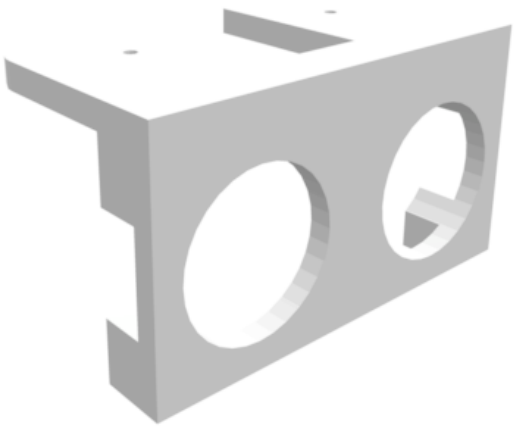


A.5 Tapphållare

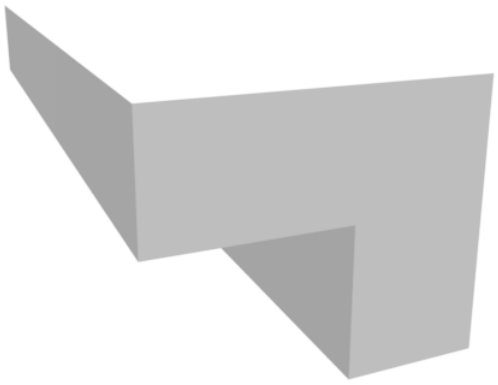


A.6 Stopp till underdelen

A.7 Servo till sensor hållare

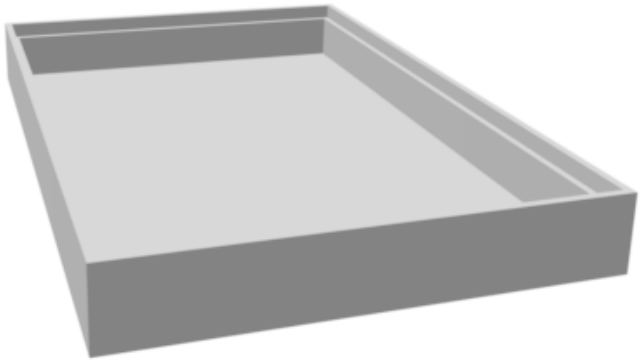


A.8 Första rälsen till "slide in, slide ut" underdelen. Fäst på spindeln

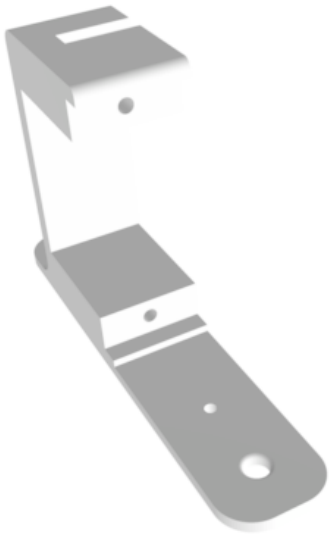


A.9 Andra rälsen till "slide in, slide ut" underdelen. Fäst på underdelen

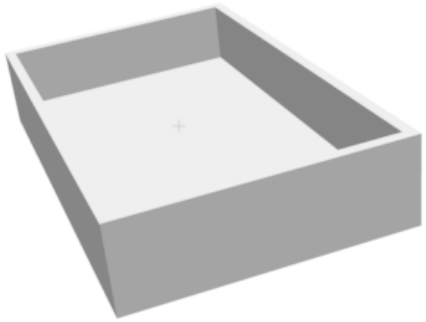
A.10 Behållare till experimentkortet



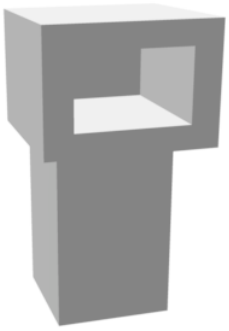
A.11 Stående servo-hållare



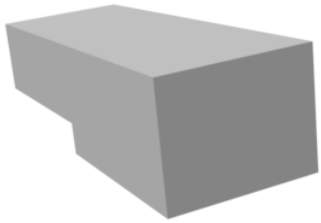
A.12 Balanserade platta fäst på servo



A.13 Tapphållare till experimentkortet



A.14 Tapp till tapphållaren



B

Bilagor kod i C++

Här finns bilagor på kod som använts i arbetet.

B.1 Initieringar

```
#include <Wire.h>
#include <Servo.h>
const int trigPin = 9;
const int echoPin = 10;
const int trigPin1 = 6;
const int echoPin1 = 5;

Servo servo_pan;
Servo servo_tilt;
int ADXL345 = 0x53; // I2C ddress for the ADXL345 sensor
float X_pos, Y_pos, Z_pos;

#define Sens_Center  75 // Upper_sensor center position
#define Sens_Max_Left 20 // Leftmost value of Upper_Sensor
#define Sens_Max_Right 150 // Rightmost value of Upper_Sensor
#define collisiondistance 20 // Collisiondistance for the Upper_Sensor
#define stairdistance  16 // Distance to start the climb
#define stairnear  24 // Distance to move slowly towards the stair
bool loopOK = true;
```

B.2 Setup-funktion

```
void setup() {

Serial.begin(9600); // Start serial communication
pinMode(trigPin, OUTPUT); // Set trigPin as output
pinMode(trigPin1, OUTPUT); // Set trigPin1 as output
pinMode(echoPin, INPUT); // Set echoPin as input
pinMode(echoPin1, INPUT); // Set echoPin1 as input

Wire.begin(); // Initiate Wire library
// Set ADXL345 in measuring mode
Wire.beginTransmission(ADXL345); // Starts the communication with the ADXL345
Wire.write(0x2D); // Talk to POWER_CTL Register
Wire.write(8); // 8 because 8 in decimal is 0000 1000 in binary, Bit D3 High for enabling measurment
Wire.endTransmission();
delay(10);
servo_tilt.attach(3);
servo_pan.attach(11);
servo_tilt.write(90);

delay(1000);
Serial.println(F("#0P2200#4P1350#8P1950#16P1300#20P2000#24P1750T1000"));//startposition sideway engines
delay(1000);
Serial.println(F("#1P1100#5P1000#9P1100#17P2400#21P2500#25P2400T1000"));//middlelegs high
delay(1000);
Serial.println(F("#2P1800#6P1550#10P1450#18P1400#22P1350#26P1550T1000"));//outer engines stable position for ascension
delay(1000);
Serial.println(F("#1P1600#5P1700#9P1600#17P1900#21P2000#25P1900T500"));// lower outer engines slightly
delay(500);
Serial.println(F("#1P1400#21P2200#9P1400T500"));//raise tripod A slightly
Serial.println(F("#2P1900#22P1250#10P1600T500"));//A outer slightly outward
delay(500);
Serial.println(F("#1P1800#21P1800#9P1800T500"));//lower A
Serial.println(F("#17P2100#5P1500#25P2100T500"));//raise tripod B slightly
Serial.println(F("#18P1300#6P1650#26P1400T500"));//B outer slightly outward
delay(500);
Serial.println(F("#17P1700#5P1900#25P1700T500"));//lower B
delay(500);
//1 steg
Serial.println(F("#1P1600#21P2000#9P1600T500"));//raise A slightly
Serial.println(F("#2P2150#22P1000#10P1800T500"));//A outer slightly outward
delay(500);
Serial.println(F("#1P2000#21P1600#9P2000T500"));//lower A
delay(500);
Serial.println(F("#17P1900#5P1700#25P1900T500"));//raise B slightly
Serial.println(F("#18P1050#6P1900#26P1200T500"));//B outer slightly outward
delay(500);
Serial.println(F("#17P1500#5P2100#25P1500T500"));//lower B
delay(500);
servo_tilt.write(90);
}
```

B.3 Funktion att gå framåt

```
void go_forward() {

Serial.println(" go forward ------------------------------ go forward");// For debugging

//1
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1800#21P1900#9P1800T500"));//A height low to mid
delay(500);

//2
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1700#21P2000#9P1700T500"));//A height mid to high
delay(10);
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral rear to center
delay(10);
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral front to center
delay(500);

//3
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1800#21P1900#9P1800T500"));//A height high to mid
delay(10);
Serial.println("#0P2050#20P2100#8P1850T500");//A lateral center to front
delay(10);
Serial.println("#16P1150#4P1550#24P1550T500");//B lateral center to rear
delay(500);//100

//4
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P2100#21P1600#9P2100T500"));//A height mid to low
delay(500);

//5
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#17P1800#5P1600#25P1700T500"));//B height low to mid
delay(500);

//6
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral front to center
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
delay(10);
Serial.println(F("#17P1900#5P1500#25P1800T500"));//B height mid to high
delay(500);

//7
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println("#0P2350#20P1900#8P2150T500");//A lateral center to rear
delay(10);
Serial.println("#16P1450#4P1150#24P1850T500");//B lateral center to front
delay(10);
Serial.println(F("#17P1800#5P1700#25P1650T500"));//B height high to mid
delay(500);

//8
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#17P1500#5P1950#25P1500T500"));//B height mid to high
delay(500);
}
```

B.4 Funktion att gå långsamt framåt

```
void slow_walk() {
Serial.println(" go forward ------------------------------ go slow"); // For debugging
//0
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1800#21P1900#9P1800T500"));//A height low to mid
delay(500);

//1
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1700#21P2000#9P1700T500"));//A height mid to high
delay(10);
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral rear to center
delay(10);
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral front to center
delay(500);

//2
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P1800#21P1900#9P1800T500"));//A height high to mid
delay(10);
Serial.println("#0P2125#20P2050#8P1900T500");//A lateral center to front
delay(10);
Serial.println("#16P1225#4P1450#24P1650T500");//B lateral center to rear
delay(500);

//3
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#1P2100#21P1600#9P2100T500"));//A height mid to low
delay(500);

//4
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#17P1800#5P1600#25P1700T500"));//B height low to mid
delay(500);

//5
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral front to center
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
delay(10);
Serial.println(F("#17P1900#5P1500#25P1800T500"));//B height mid to high
delay(500);

//6
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println("#0P2275#20P1950#8P2050T500");//A lateral center to rear
delay(10);
Serial.println("#16P1375#4P1250#24P1800T500");//B lateral center to front
delay(10);
Serial.println(F("#17P1800#5P1700#25P1650T500"));//B height high to mid
delay(500);

//7
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
Serial.println(F("#17P1500#5P1950#25P1500T500"));//B height mid to high
delay(500);
}
```

B.5 Funktion att klättra (Del 1)

```
void climb() {

Serial.println(F("Klättra------------------Klättra"));// For debugging
Serial.println(F("#17P1500#5P2000#25P1500T500"));// B height upward
Serial.println(F("#2P2450#22P600#10P2300T500"));// A outer engine in startposition
Serial.println(F("#18P680#6P2300#26P750T500"));// B outer engine in startposition
delay(500);
Serial.println(F("#16P1400#4P1300#24P1800T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1320#5P2200#25P1300T500"));// B height downwards
delay(500);
Serial.println(F("#1P2100#21P1600#9P2100T500"));// A height upward
senstilt();
Serial.println(F("#16P1300#4P1400#24P1700T500"));// B lateral backwards
delay(500);
Serial.println(F("#0P2100#20P2000#8P2000T500"));// A lateral forward
Serial.println(F("#2P2450#22P600#10P2300T500"));// A outer engine in startposition
Serial.println(F("#18P680#6P2300#26P750T500"));// B outer engine in startposition
delay(500);
senstilt();
Serial.println(F("#1P2320#21P1430#9P2300T500"));// A height downwards
delay(500);
Serial.println(F("#0P2200#20P1900#8P2100T500"));// A lateral backwards
delay(500);

// one slow step
senstilt();

Serial.println(F("#17P1500#5P2000#25P1500T500"));// B height upward
 Serial.println(F("#2P2450#22P600#10P2300T500"));// A outer engine in startposition
 Serial.println(F("#18P680#6P2300#26P750T500"));// B outer engine in startposition
delay(500);
Serial.println(F("#16P1400#4P1300#24P1800T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1320#5P2200#25P1300T500"));// B height downwards
delay(500);
Serial.println(F("#1P2100#21P1600#9P2100T500"));// A height upward
senstilt();
Serial.println(F("#16P1300#4P1400#24P1700T500"));// B lateral backwards
delay(500);
Serial.println(F("#0P2100#20P2000#8P2000T500"));// A lateral forward
 Serial.println(F("#2P2450#22P600#10P2300T500"));// A outer engine in startposition
 Serial.println(F("#18P680#6P2300#26P750T500"));// B outer engine in startposition
delay(500);
senstilt();
Serial.println(F("#1P2320#21P1430#9P2300T500"));// A height downwards
delay(500);
Serial.println(F("#0P2200#20P1900#8P2100T500"));// A lateral backwards
delay(500);

// one step each tripod
senstilt();

Serial.println(F("#17P2300#5P2000#25P1500T500"));// B height upward
Serial.println(F("#18P1400T500"));//frontlegs lower height
delay(500);
Serial.println(F("#16P1500#4P1200#24P1900T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1700#5P2200#25P1300#18P1300T500"));// B height downwards
delay(500);
Serial.println(F("#17P1850T500"));//adjustment
senstilt();
delay(500);
Serial.println(F("#16P1400#4P1400#24P1600T500"));// B lateral backwards
Serial.println(F("#1P1300#21P1600#9P2100#2P1500#22P500T500"));// A height upward
delay(500);
senstilt();
Serial.println(F("#0P1900#20P2100#8P1900T500"));// A lateral forward
delay(500);
Serial.println(F("#1P1850#21P1250#9P2350#2P2150T500"));// A height downwards
delay(500);

// two steps done, front legs on stairstep
```

B.6 Funktion att klättra (Del 2)

```
senstilt();

Serial.println(F("#17P2300#5P2000#25P1500T500"));// B height upward
Serial.println(F("#18P1400T500"));//frontlegs lower height
delay(500);
Serial.println(F("#0P2300#20P1800#8P2100T500"));// A lateral backwards
delay(500);
Serial.println(F("#16P1500#4P1200#24P1900T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1700#5P2350#25P1300T500"));// B height downwards
Serial.println(F("#18P1300#6P2400T500"));//outer engines folow
delay(500);
Serial.println(F("#17P1850T500"));//adjustment
senstilt();
delay(500);
Serial.println(F("#16P1300#4P1400#24P1600T500"));// B lateral backwards
delay(500);
Serial.println(F("#1P1300#21P1600#9P2100T500"));// A height upward
Serial.println(F("#2P1500#22P500T500"));//outer engines follow
delay(500);
senstilt();
Serial.println(F("#0P2100#20P2100#8P1900T500"));// A lateral forward
delay(500);
Serial.println(F("#1P1850#21P1300#9P2300T500"));// A height downwards
Serial.println(F("#2P2200T500"));//outer engines folow
delay(500);

// three steps done, front legs on stairstep
senstilt();

Serial.println(F("#17P2300#5P2000#25P1500T500"));// B height upward
Serial.println(F("#18P1400T500"));//frontlegs lower height
delay(500);
Serial.println(F("#0P2300#20P1800#8P2100T500"));// A lateral backwards
delay(500);
senstilt();
Serial.println(F("#16P1400#4P1200#24P1900T500"));// B lateral forward
delay(500);
Serial.println(F("#17P1700#5P2350#25P1300T500"));// B height downwards
Serial.println(F("#18P1200#6P2400T500"));//frontlegs lower height
delay(500);
Serial.println(F("#17P1850T500"));//adjustment
senstilt();
delay(500);
Serial.println(F("#16P1300#4P1400#24P1600T500"));// B lateral backwards
delay(500);
Serial.println(F("#1P1300#21P2200#9P2100T500"));// A height upward
Serial.println(F("#2P1600#22P1300T500"));//frontlegs lower height
delay(500);
senstilt();
Serial.println(F("#0P2100#20P2100#8P1900T500"));// A lateral forward
delay(500);
Serial.println(F("#1P1850#21P1950#9P2300T500"));// A height downwards
Serial.println(F("#2P2200#22P1000T500"));//frontlegs lower height
delay(500);

// four steps done, right middleleg on stairstep
senstilt();

Serial.println(F("#17P2300#5P1200#25P1500T500"));// B height upward
Serial.println(F("#18P1400#6P1500T500"));//frontlegs lower height
delay(500);
Serial.println(F("#0P2300#20P1900#8P2100T500"));// A lateral backwards
delay(500);
Serial.println(F("#16P1400#4P1200#24P1900T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1700#5P1700#25P1200T500"));// B height downwards
Serial.println(F("#18P1050#6P1900#26P700T500"));//frontlegs lower height
delay(500);
Serial.println(F("#17P1850T500"));//adjustment
senstilt();
delay(500);
Serial.println(F("#16P1300#4P1400#24P1600T500"));// B lateral backwards
delay(500);
Serial.println(F("#1P1300#21P2200#9P2100T500"));// A height upward
Serial.println(F("#2P1600#22P1250T500"));//frontlegs lower height
delay(500);
senstilt();
Serial.println(F("#0P2100#20P2100#8P1900T500"));// A lateral forward
delay(500);
Serial.println(F("#1P2000#21P1900#9P2400T500"));// A height downwards
Serial.println(F("#2P2200#22P1000#10P2400T500"));//frontlegs lower height
delay(500);

// five steps done, both middlelegs on stairstep
```

B.7 Funktion att klättra (Del 3)

```
senstilt();

Serial.println(F("#17P2300#5P1200#25P1500T500"));// B height upward
Serial.println(F("#18P1400#6P1500T500"));//frontlegs lower height
delay(500);
Serial.println(F("#0P2300#20P1800#8P2100T500"));// A lateral backwards
delay(500);
Serial.println(F("#16P1400#4P1200#24P1900T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1600#5P1800#25P1200T500"));// B height downwards
Serial.println(F("#18P1050#6P1900#26P700T500"));//frontlegs lower height
senstilt();
delay(500);
Serial.println(F("#16P1300#4P1400#24P1600T500"));// B lateral backwards
delay(500);
Serial.println(F("#1P1300#21P2200#9P2100T500"));// A height upward
Serial.println(F("#2P1600#22P1300T500"));//frontlegs lower height
delay(500);
senstilt();
Serial.println(F("#0P2100#20P2000#8P1900T500"));// A lateral forward
delay(500);
Serial.println(F("#1P2100#21P1900#9P2400T500"));// A height downwards
Serial.println(F("#2P2200#22P900#10P2400T500"));//frontlegs lower height
delay(500);

// six steps done, both middlelegs on stairstep
senstilt();

Serial.println(F("#17P2300#5P1200#25P2200T500"));// B height upward
Serial.println(F("#18P1400#6P1500#26P1400T500"));//frontlegs lower height
delay(500);
Serial.println(F("#0P2300#20P1800#8P2100T500"));// A lateral backwards
delay(500);
senstilt();
Serial.println(F("#16P1400#4P1200#24P1900T500"));// B lateral forward
delay(500);
senstilt();
Serial.println(F("#17P1600#5P1800#25P2000T500"));// B height downwards
Serial.println(F("#18P1050#6P1900#26P1500T500"));//frontlegs lower height
delay(500);
senstilt();
delay(500);
Serial.println(F("#16P1300#4P1400#24P1700T500"));// B lateral backwards
delay(500);
Serial.println(F("#1P1300#21P2200#9P1400T500"));// A height upward
Serial.println(F("#2P1600#22P1300#10P1900T500"));//frontlegs lower height
delay(500);
senstilt();
Serial.println(F("#0P2100#20P2000#8P1700T500"));// A lateral forward
delay(500);
Serial.println(F("#1P2100#21P1900#9P1750T500"));// A height downwards
Serial.println(F("#2P2200#22P900#10P1600T500"));//frontlegs lower height
delay(1000);

// seven steps done, both backlegs on stairstep
senstilt();

Serial.println(F("#9P1950#25P1800T500"));
Serial.println(F("#5P1900#21P1800T500"));
Serial.println(F("#1P1900#17P1800T500"));
delay(500);

// balance found
```

B.8 Funktion att klättra (Del 4)

```
senstilt();

Serial.println(F("#1P1400#21P2200#9P1400T500"));//raise A slightly
Serial.println(F("#2P1900#22P1100#10P1600T500"));// A outer engines outward
delay(500);
senstilt();
Serial.println(F("#1P1800#21P1800#9P1800T500"));//lower A
delay(500);
senstilt();
Serial.println(F("#17P2100#5P1500#25P2100T500"));//raise B slightly
Serial.println(F("#18P1300#6P1650#26P1400T500"));//B outer engines outward
delay(500);
senstilt();
Serial.println(F("#17P1700#5P1900#25P1700T500"));// lower B
delay(500);
//1 step
senstilt();
Serial.println(F("#1P1600#21P2000#9P1600T500"));//raise A slightly
Serial.println(F("#2P2150#22P1000#10P1800T500"));// A outer engines outward
delay(500);
senstilt();
Serial.println(F("#1P2000#21P1600#9P2000T500"));//lower A
delay(500);
senstilt();
Serial.println(F("#17P1900#5P1700#25P1900T500"));//raise B slightly
Serial.println(F("#18P1050#6P1900#26P1200T500"));//B outer engines outward
delay(500);
senstilt();
Serial.println(F("#17P1500#5P2100#25P1500T500"));// lower B
delay(500);
senstilt();
delay(1000);
}
```

B.9 Funktion att svänga höger

```
void turn_right() {
Serial.println(" turn right ------------------------------ turn right");// For debugging
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral front to center
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
Serial.println("#17P1500#5P1950#25P1600T500");//B height mid to high
Serial.println("#1P2100#21P1600#9P2100T500");//A height mid to low
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
delay(500);
Serial.println("#17P1900#5P1500#25P1800T500");//B height mid to high
delay(500);
Serial.println("#16P1150#4P1200#24P1600T500");//B lateral right
delay(500);
Serial.println("#17P1500#5P1950#25P1600T500");//B height mid to high
delay(500);
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
Serial.println("#1P1700#21P2000#9P1700T500");//A height mid to high
delay(500);
Serial.println("#0P2050#20P1850#8P1800T500");//A lateral right
delay(500);
Serial.println("#1P2100#21P1600#9P2100T500");//A height mid to low
delay(500);
}
```

B.10 Funktion att svänga vänster

```
void turn_left() {
Serial.println(" turn left ------------------------------ turn left");// For debugging
Serial.println("#0P2200#20P2000#8P1950T500");//A lateral front to center
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
Serial.println("#17P1500#5P1950#25P1600T500");//B height mid to high
Serial.println("#1P2100#21P1600#9P2100T500");//A height mid to low
Serial.println("#2P2300#22P850#10P1950T500");// A outer engines in startposition
Serial.println("#18P900#6P2050#26P1050T500");// B outer engines in startposition
delay(500);
Serial.println("#17P1900#5P1500#25P1800T500");//B height mid to high
delay(500);
Serial.println("#16P1450#4P1500#24P1900T500");//B lateral left
delay(500);
Serial.println("#17P1500#5P1950#25P1600T500");//B height mid to high
delay(500);
Serial.println("#16P1300#4P1350#24P1750T500");//B lateral rear to center
Serial.println("#1P1700#21P2000#9P1700T500");//A height mid to high
delay(500);
Serial.println("#0P2350#20P2150#8P2100T500");//A lateral left
delay(500);
Serial.println("#1P2100#21P1600#9P2100T500");//A height mid to low
delay(500);
}
```

B.11 Funktion att läsa av övre sensorn

```
int Upper_Sensor() { // Upper sensor that calculates ditsnce to obstacles

  long upper_duration = 0;
  int upper_distance  = 0;

  for (int i = 0; i < 4; i++) { // Average value 4 times

    digitalWrite(trigPin1, LOW);// Clears the trigPin
    delayMicroseconds(2);

    digitalWrite(trigPin1, HIGH);// Sets the trigPin on HIGH state for 10 micro seconds
    delayMicroseconds(10);
    digitalWrite(trigPin1, LOW);

    upper_duration = pulseIn(echoPin1, HIGH);// Reads the echoPin, returns the sound wave travel time in microseconds

    upper_distance += upper_duration*(0.034/2);//Calculate distance in cm and add all 4 samples
    delay(5);
  }

  Serial.print("Upper_Sensor: ");
  Serial.println(upper_distance/4);
  return round(upper_distance/4); // Return distance divided by 4 to get the average value of 4 measures
}
```

B.12 Funktion att läsa av nedre sensorn

```
int Lower_Sensor () { //Lower sensor that calculates the distance to the step

  long lower_duration = 0;
  int lower_distance  = 0;

  for (int i = 0; i < 4; i++) {// Average value 4 times

    digitalWrite(trigPin, LOW);// Clear trigPin
    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);// Set trigPin high for 10 microseconds
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    lower_duration = pulseIn(echoPin, HIGH);// Read echoPin to calculate time for the sound in microseconds

    lower_distance += lower_duration*(0.034/2);//Calculate distance in cm and add all 4 samples
    delay(5);

  }

  Serial.print("Lower_Sensor: ");
  Serial.println(lower_distance/4);
  return round(lower_distance/4); // Return distance divided by 4 to get the average value of 4 measures
}
```

B.13 Funktion att stanna

```
void stop_position() {

  Serial.println("Stanna-----------------Stanna");// For debugging
  Serial.println("#2P2500#22P600#10P2300T500");// A outer engine in startposition
  Serial.println("#18P680#6P2300#26P750T500");// B outer engine in startposition
  Serial.println("#0P2200#20P2000#8P2100T500"); // A lateral Center
  Serial.println("#16P1300#4P1400#24P1700T500");// B lateral Center
  Serial.println("#1P2320#21P1430#9P2300T500");// A height middle
  Serial.println("#17P1320#5P2200#25P1300T500");// B height middle
}
```

B.14 Funktion att vinkla en platta

```
void senstilt() {
  Wire.beginTransmission(ADXL345);
  Wire.write(0x32); // Start with the 0x32 register
  Wire.endTransmission(false);
  Wire.requestFrom(ADXL345, 6, true); // Read 6 registers total where each axis value is stored in 2 of the 6 registers
  X_pos = ( Wire.read()| Wire.read() << 8); // Position for X axis
  X_pos = X_pos/256; // Divided by 256 according to the datasheet for the sensitivity +-2g
  Y_pos = ( Wire.read()| Wire.read() << 8); // Position for Y axis
  Y_pos = Y_pos/256; // Divided by 256 according to the datasheet for the sensitivity +-2g
  Z_pos = ( Wire.read()| Wire.read() << 8); // Position for Z axis
  Z_pos = Z_pos/256; // Divided by 256 according to the datasheet for the sensitivity +-2g

  Y_pos = ((-90*Y_pos)+103); //Converting horizontal value 0 to right angle to send to servo, A slight adjustment to match reallife

  if (Y_pos>130) { // Limit servoangle to 110 if sensor goes higher
    Y_pos = 130;
  }
  else if (Y_pos<50) { // Limit servoangle to 50 if sensor goes lower
    Y_pos = 50;
  }

    Serial.print("Vinkeln =  ");
  Serial.println(Y_pos);
  servo_tilt.write(Y_pos); // write position to servo

}
```

B.15 Main-program

```
void loop() {


  while((Lower_Sensor() > stairdistance) && loopOK) {

    if((Lower_Sensor() > stairdistance) && (Lower_Sensor() <= stairnear)) {


      if((Lower_Sensor() > stairdistance) && (Lower_Sensor() <= stairnear)) {
        slow_walk();
      }
    }

    else {
      go_forward();
    }

    for (int sensangle = Sens_Max_Left; (sensangle <= Sens_Max_Right) && loopOK; sensangle = sensangle + 2) { // scanner turn to full right

      if(Lower_Sensor() <= stairdistance){
        loopOK = false;
        break;
      }

      while (Upper_Sensor() < collisiondistance) {

        if (sensangle <= Sens_Center) { // obstacle on the left side
          turn_right();
        }

        else if (sensangle > Sens_Center) { // obstacle on the right side
          turn_left();
        }
      }

      servo_pan.write(sensangle);// sweep servo
    }

    if(loopOK) {

      if((Lower_Sensor() > stairdistance) && (Lower_Sensor() <= stairnear)) { //check if distance to short to stair to do (go_forward())


        if((Lower_Sensor() > stairdistance) && (Lower_Sensor() <= stairnear)) { //Doublecheck
          slow_walk();
        }
      }

      else {
        go_forward();
      }
    }

                // scanner turn to full left
    for (int sensangle = Sens_Max_Right; (sensangle >= Sens_Max_Left) && loopOK; sensangle = sensangle -2) {

      if(Lower_Sensor() <= stairdistance) {
        loopOK = false; //break out of loop
        break;
      }

      while (Upper_Sensor() < collisiondistance) {

        if (sensangle <= Sens_Center) {// obstacle on left side
          turn_right();
        }

        else if (sensangle > Sens_Center) {// obstacle on right side
          turn_left();
        }
      }

      servo_pan.write(sensangle);  // sweep servo
    }

    loopOK = true;
  }

  if(Lower_Sensor() <= stairdistance) {
    climb();
    loopOK = true;
  }
}
```

C

Andra bilagor

C.1 Elkretsschema