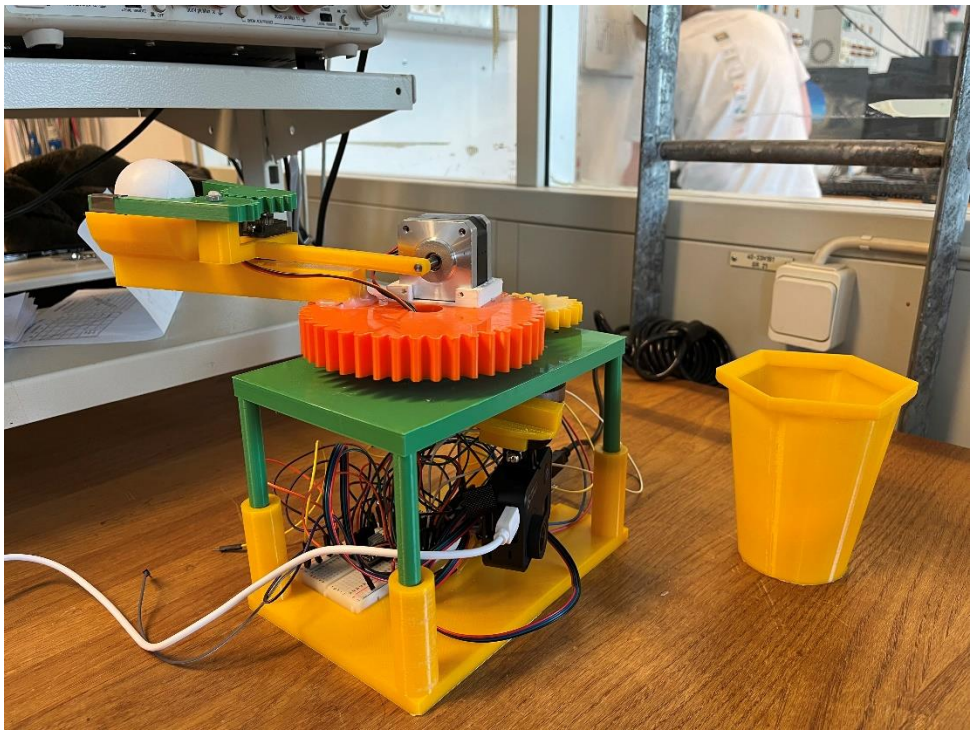
Kandidatexamensarbete inom Teknik

Grundnivå, 15 HP

Pongbot

En robotarm med automatisk bildanalys

ROBIN GUNNARSSON OCH OSCAR BRINDEVI



Stockholm, Sweden 2024

Pongbot

En robotarm med automatisk bildanalys

Robin Gunnarsson och Oscar Brindevi

Kandidatexamensarbete inom Teknik
Datum: 2024-06-03
Handledare: Daniel Frede
Examinator: Daniel Frede
ITM-skolan
English title: Pongbot
English subtitle: A robot arm with automatic image recognition

© 2024 Robin Gunnarsson, Oscar Brindevi

TRITA – ITM-EX 2024:52

www.kth.se

Abstract

Robotic arms have a wide range of uses, for example in the industry. They can be used to automate and standardize manufacturing processes et cetera. Many of these robotic arms are very complex and have many degrees of freedom. The robotic arm that has been manufactured in this project is very different both in design and application. The arm itself has only one degree of freedom similar to a catapult. The purpose of the robot is to play Beer Pong, that is, throw ping pong balls into plastic cups. What is primarily explored in the project is the precision of the robot's throw. The key components are a camera to detect plastic cups with color recognition, as well as measure how far away the cup is. Two stepper motors are used, one to drive the throw and one to rotate the structure. A claw is also used to grip and release the ball, which is powered by a servo motor. All motors are controlled via an Arduino UNO, and control of the camera and all calculations are done in Python on a laptop. When testing the robotic arm, 37% of all throws hit, although about 80% of all throws were very close to hitting and hit the edge of the cup. The robot's maximum throw length is around 60 cm.

Keywords

Mechatronics, robotics, degrees of freedom, kinematics, precision

Sammanfattning

Robotarmar har en bred användning, bland annat inom industrin. De kan användas för att automatisera och standardisera till exempel tillverkningsprocesser. Många av dessa robotarmar är väldigt komplexa och har många frihetsgrader. Den robotarm som har tillverkats i detta projekt är väldigt annorlunda både till design och till användningsområde. Själva armen har endast en frihetsgrad likt en katapult. Syftet med roboten är att spela Beer Pong, det vill säga kasta pingisbollar i plastmuggar. Det som primärt utforskas i projektet är precisionen på robotens kast, det vill säga hur pricksäker robotarmen kan bli. Nyckelkomponenterna är en kamera för att upptäcka plastmuggar med färgigenkänning, samt mäta hur långt bort muggen är. Två stegmotorer används, en för att driva kastet och en för att rotera konstruktionen. En klo används även för att kunna greppa och släppa bollen, som drivs av en servomotor. Alla motorer styrs via en Arduino UNO, och styrning av kameran samt alla beräkningar sker i Python på en laptop. Vid testning av robotarmen träffade 37% av alla kast, trots att cirka 80% av alla kast var väldigt nära att träffa och studsade av kanten på muggen. Robotens maximala kastlängd är omkring 60 cm.

Nyckelord

Mekatronik, robotik, frihetsgrader, kinematik, precision

Förord

Vi skulle vilja tacka Daniel Frede som varit vår handledare, som har stått till hjälp och stöd under projektets gång vid behov. Vi skulle även vilja tacka Kasper Georén, Albin Gunnarsson och Jacob Holst som varit assistenter och som hjälpt till mycket vid konstruktionen och med idéer och allmänt stöd.

Stockholm, Maj 2024
Robin Gunnarsson, Oscar Brindevi

Innehållsförteckning

1 Introduktion 1
1.1 Bakgrund 1
1.2 Problem 1
1.3 Syfte 1
1.4 Mål 1
1.5 Forskningsmetod 2
1.6 Avgränsningar 2
1.7 Rapportstruktur 3
2 Bakgrund 5
2.1 Mikrokontroller 5
2.1.1 Arduino UNO 5
2.1.2 Raspberry Pi 5
2.2 Stegmotorer 6
2.3 Servomotorer 6
2.4 Frihetsgrader 6
2.5 Pyserial 6
2.6 Robotik 6
2.7 Maskininlärning 7
2.8 Tidigare arbeten 7
2.8.1 Tidigare KEX, robotarm med sex frihetsgrader 7
2.8.2 Tidigare kastande robotarm 8
2.8.3 Maskininlärning för att känna igen vardagliga föremål 8
2.9 Sammanfattning 8
3 Metod 9
3.1 Ingenjörstillämpningar och metodik 9
3.1.1 Val av tillverkningsmetoder 9
3.1.2 Igenkänningsmetod för muggar 9
3.1.3 Dator för styrning av Python och kamera 9
3.1.4 Beräkning av kastlängd 10
3.2 Evauleringsmetod av forskningsfrågor 10
3.3 Komponenter 10
3.4 Design 11
3.4.1 Bord 13
3.4.2 Arm och rotation 13
3.4.3 Greppningsmekanism 14
3.4.4 Bollkopp 15
3.5 Kod 16
3.5.1 C++ (Arduino) 16
3.5.2 Python för kamera och beräkningar (Laptop) 16
4 Resultat 21
4.1 Resultat 21
4.2 Trovärdighetsanalys 21
4.3 Giltighetsanalys 21
4.4 Diskussion 21
5 Slutsatser och framtida arbete 23
5.1 Slutsatser 23
5.2 Begränsningar 23

5.2.1 Tid och omfattning 23
5.2.2 Motorkraft 23
5.2.3 Uppskattning av kastlängd 23
5.3 Framtida arbete 24
5.4 Reflektioner 24
Referenser 25
Appendix A: Arduinokod I C++ 27
28
Appendix B: Pythonkod för kamera och beräkningar 29
Appendix C: Tabell över resultat från test av Pongbot 32

Figurlista

Figur 3.4-1: Pongbot framifrån 11
Figur 3.4-2: Pongbot från ovan 11
Figur 3.4-3: Kamerafäste på kugghjuls axel 13
Figur 3.4-4: Rotationsmekanism 14
Figur 3.4-5: Klo 14
Figur 3.4-6: Bollkopp 15
Figur 3.4-7: 3D-printad plastmugg 15
Figur 3.5-1: Flödesschema för kod i Python på laptop 18
Figur 3.5-2: Flödesschema för kod på Arduino vid kast 19

Tabellista

Tabell 3.3-1: Komponentlista 10

Lista över akronymer och förkortningar

ALU	Arithmetic Logic Unit
AD	Analog to Digital
CAD	Computer Aided Design
CPU	Central Processing Unit
ITM	Industriell Teknologi och Management
GPIO	General Purpose Input and Output
KEX	Kandidatexamensarbete
PWM	Pulse Width Modulation
RAM	Random Access Memory

1 Introduktion

1.1 Bakgrund

Robotarmar har en stor användning inom industrin vid processer som inte kräver mänskligt arbete. I många fall används de för att automatisera till exempel tillverkningsprocesser, då de till skillnad mot människor kan arbeta 24 timmar om dygnet och kan programmeras för kontinuerlig drift med minimala avvikelser. [1] Vid massproduktion och standardiserade processer inom industrin är robotarmar välanvända. En applicering som inte är lika vanlig är att använda robotarmar för mer explosiva rörelser, såsom kaströrelser. Även om robotarmar har börjat användas en del för liknande saker på senare år brukar dessa armar bli väldigt komplicerade. Ofta används väldigt många olika motorer som ger ett flertal frihetsgrader. Trots allt detta är många robotarmar inte särskilt pricksäkra med deras kast. När så pass många motorer och frihetsgrader används blir det mycket som måste samarbeta för att ge en hög noggrannhet. [2] I detta arbete ska det undersökas hur bra en robotarm kan användas till att kasta en pingisboll i en plastmugg. Detta ska göras med så få frihetsgrader och motorer som möjligt, för att undvika beräknings- och kastosäkerheter. Det som ska undersökas är hur noggrann robotarmen kan bli, det vill säga hur stor andel av kasten som robotarmen kan träffa i muggen.

1.2 Problem

Många robotarmar är svåra att få noggranna, bland annat då många av dem har många frihetsgrader och många olika motorer. Går det att få en kastande robotarm att bli noggrann och precis med sina kast genom att endast ha en frihetsgrad? Hur stor andel av alla kast går det att få roboten att träffa? Hur långt kan roboten kasta pingisbollar, och vid vilka längder på kast är precisionen som bäst?

1.3 Syfte

Syftet med arbetet är att undersöka hur väl denna tillämpning fungerar och hur väl vi kan få hela robotarmen och konstruktionen att samarbeta. Om det visar sig att denna applikation fungerar bra och uppför sig som önskat tyder det på att liknande applikationer kan göras i framtiden inom industrin för diverse olika projekt. Syftet med arbetet är med andra ord att undersöka om denna sorts applikation är lämplig samt hur pricksäker denna robotarm kan bli.

Följande forskningsfrågor ska besvaras:

- Hur stor andel av kasten går det att träffa?
- Hur långt kan roboten kasta pingisbollar?

1.4 Mål

Det huvudsakliga målet är att bygga en robotarm som kan kasta pingisbollar i plastmuggar. Om detta uppfylls är projektet i stora drag lyckat. Detta kan delas upp i mindre, mer specifika mål. Ett mål är att AI ska användas för att känna igen den sortens plastmugg som ska användas. Detta ska ske med hjälp av en kamera. När en mugg upptäckts ska först hela konstruktionen vridas så att muggen är rakt framför armen och därmed är i linje med armens kastriktning. Sedan ska avståndet till muggen uppskattas genom en storleksanalys från kameran. Till sist ska denna beräkning användas för att kasta bollen, så att den träffar i muggen. Denna beräkning kommer endast bli en approximation och därför anses det inte rimligt att träffa 100% av alla kast. Om 50% av alla kast träffar i muggen anses robotarmen vara tillräckligt pricksäker och detta mål uppfyllt. För att sammanfatta dessa delmål:

1. Specifika plastmuggar ska kännas igen med AI

2. Konstruktionen ska kunna vridas ungefär 90° så att muggar inom kamerans synfält kan placeras rakt framför kameran.
3. Avståndet till muggen ska uppskattas med hur stor muggen upplevs av kameran
4. Hur hårt bollen ska kastas ska beräknas med matematiska formler
5. Armen ska kunna kasta så pass hårt/löst som beräkningarna föreslår
6. Minst 50% av kasten ska träffa i muggen

1.5 Forskningsmetod

För att hitta vetenskapliga källor och artiklar för att lägga en teoretisk grund för arbetet ska Google Scholar användas. Google Scholar visar endast forskningsartiklar och liknande och sållar bort allt irrelevant för ett arbete som detta. Därför är det att föredra framför att endast använda Google eller en liknande sökmotor. Tidigare liknande arbeten ska även studeras för inspiration.

När tidigare arbeten studerats ska skisser genomföras av hur armen kan se ut. Dessa skisser ska evalueras och undersökas för att komma fram till den skiss som är bäst lämpad för projektet, utifrån projektets mål.

Modeller av länkar i armen och liknande delar ska konstrueras i Solid Edge och sedan 3D-printas. Detta är för att få en detaljerad modell av hur alla delar ska se ut i förväg, och smidigt kunna ändra på alla delar innan de tillverkas.

Kodskelettet som ska stå för sökningen och kastningen antas vara omfattande. Å andra sidan antas det att det finns en hel del liknande kodsnitt på internet som i många fall kan vara källor av inspiration, eller ibland till och med precis vad som söks. Förhoppningsvis och gissningsvis finns en hel del kod på internet som kan vara till hjälp som alltså ska användas så mycket det bara går.

Mycket praktisk testning ska även genomföras när robotarmen börjar bli färdig, för att kalibrera armen utifrån verkligen. Gissningsvis kommer armen inte bete sig precis som tänkt direkt från start, och det kommer behöva undersökas en hel del hur olika parametrar och formler ska se ut för att få armen så pricksäker som möjligt. Med en funktion som kastning antas teorin och praktiken inte vara exakt samma sak.

1.6 Avgränsningar

För att undvika behovet av att integrera flera olika frihetsgrader och motorer vid själva kaströrelsen kommer endast en frihetsgrad och motor användas för kaströrelsen. Själva armen är byggd som en katapult, det vill säga en enda stång utan rörliga leder. Klon som håller fast bollen drivs även av en servomotor. Trots att denna givetvis har en inverkan på hur kastet kommer se ut bidrar den inte med någon extra kraft eller liknande till armens rörelse.

De plastmuggar som ska hittas med hjälp av kameran behöver även stå inom kamerans synfält, vilket är ungefär 90° brett. Skulle en mugg sättas långt utanför kamerans synfält skulle den inte upptäckas. Genom att låta hela robotarmen rotera 360 grader skulle detta kunna åtgärdas. Det anses däremot bidra med nya problem och inte vara nödvändigt. Givetvis kommer inte robotarmen kunna kasta bollar hur långt som helst då motorkraften är begränsad. Plastmuggar behöver därmed placeras inom

ett begränsat fält framför kameran, ungefär ett triangulärt fält som är begränsat av kamerans synvinkel samt armens maximala kastlängd.

1.7 Rapportstruktur

Kapitel 2 behandlar den teoretiska bakgrunden till arbetet, allt från bakgrund om komponenter och koncept till tidigare arbeten inom samma område. I kapitel 3 behandlas ingenjörsrelaterade slutsatser och metod. I kapitel 4 presenteras resultat och en analys av resultatet. I kapitel 5 dras till sist slutsatser och föreslår eventuella vidarearbeten.

2 Bakgrund

I detta kapitel tas grundläggande bakgrundsinformation upp om relevanta begrepp och forskningsfält för arbetet. Tidigare arbeten och projekt av liknande slag undersöks även för att ta reda på vad som tidigare gjorts som är bra, vad som kan förbättras, vad som kanske borde hållas i åtanke och så vidare.

2.1 Mikrokontroller

Mikrokontrollers fungerar som en liten dator och används vanligtvis för att läsa av sensorer och kontrollera mekatroniska komponenter såsom elmotorer och lampor. De introducerades först i början på 1970-talet [3], och idag går de att finna i majoriteten av alla tekniska produkter t.ex. vitvaror, fordon och hemelektronik. En mikrokontroller består vanligtvis av följande komponenter:

- **CPU:** Som i sin tur består av fler enheter, ALU:n utför beräkningar baserat på instruktioner som hämtas och avkodas av kontrollenheten. Register och cache används för att lagra temporär data och instruktioner.
- **Minne:** Dels programminnet där icke-flyktigt innehåll lagras och dataminnet, mer känt som RAM-minne, där flyktigt innehåll lagras (vilket betyder att det återställs då strömmen försvinner).
- **Input- och output portar:** Möjliggör anslutning till andra elektroniska komponenter så som sensorer och motorer.
- **Adress- och databussar:** Sammankopplar de olika delsystemen i mikrokontrollern så att data och instruktioner kan skickas emellan dem.
- **Klocka:** Synkroniserar systemet i sin helthet.

2.1.1 Arduino UNO

Arduino är en plattform för elektronik som består av användarvänlig hård- och mjukvara. Arduino UNO, som är en modell av mikrokontroller och kommer att användas som exempel då den är en utav de främst förekommande modellerna, kan både läsa av digitala inputs, såsom en knapptryckning, och analoga inputs, till exempel en ultraljudssensor. Den kan också ge outputs, dessa är begränsade till digitala outputs varav vissa output pinnar kan användas för PWM-styrning [4]. För att styra Arduino-kontrollern används mjukvaran Arduino IDE där program kan skrivas i C , C++ eller i Arduinos språk[5].

2.1.2 Raspberry Pi

En Raspberry Pi, som mer kan ses som en liten dator, har fler användningsområden än Arduinon. Den kan användas till att visa videor och surfa på internet men kan också användas som en plattform för att integrera elektronik, precis som Arduinon [6]. Raspberry Pi har GPIO-pinnar som kan användas för att t ex läsa av sensorer och kontrollera hårdvara, det finns dock ingen möjlighet att direkt ansluta analoga sensorer till dessa pinnar då den saknar en AD-omvandlare.

2.2 Stegmotorer

En stegmotor är en elektromekanisk motor som rör sig i diskreta steg, därav namnet stegmotor. En elektrisk insignal roterar motorn med en diskret vinkel som kallas för stegvinkeln [7]. Förutsatt att motorn har tillräckligt med små diskreta steg för ändamålet är en stegmotor väldigt precis och pålitlig med hur stor utsignal som fås då den alltid går exakt lika många diskreta steg för samma insignal. Detta gör en stegmotor till ett perfekt val av motor för både själva kaströrelsen och själva rotationen av hela armen, då dessa rörelser kräver att de är reproducerbara med hög precision.

2.3 Servomotorer

En servomotor är en elmotor som kan rotera olika maskindelar med hög precision och verkningsgrad. Motorns axel kan flyttas till en viss vinkel, och med kontrollerad hastighet. Detta görs med PWM, som gör att styrkan på signalen lätt kan bestämmas. Det finns en min- och en maxpuls. Positionen 0 definieras som positionen där motorns kan rotera lika mycket åt båda håll. Servomotorer kan bland annat bestå av både AC – eller DC motorer, men till skillnad från till exempel en DC motor har även servomotorer sensorer som bidrar med positionsåterkoppling. [8] Med en servomotors snabba behandling av signaler och den höga precisionen är en servomotor ett bra val för motor som öppnar och stänger klon.

2.4 Frihetsgrader

Frihetsgrader definieras inom mekaniken som antalet oberoende variabler som beskriver hur ett system kan röra sig, med andra ord i hur många riktningar systemet kan röra på sig eller vrida sig. [9] En robotarm som kan röra på sig med en led som en katapult, och som även kan rotera har alltså två frihetsgrader. Om armen istället skulle ha två rörbara leder, alltså som en armbåge till exempel, skulle den då ha tre frihetsgrader.

2.5 Pyserial

I programmeringsspråket Python går det att ladda ned och använda sig av olika moduler, som kan beskrivas som en fil som innehåller satser och definitioner som möjliggör användning av diverse kommandon som inte annars hade gått att använda [10]. En modul kan beskrivas som en verktygslåda för att genomföra en viss typ av arbete. Dessa moduler importeras sedan in till en fil för att använda sig av denna verktygslåda. Ett exempel på en sådan modul är modulen ”Pyserial” som tillåter att ett pythonprogram kommunicerar med serieportarna på en dator [11]. Detta möjliggör kommunikation med en mikrokontroller, till exempel en Arduino UNO som är inkopplad till en serieport på datorn som pythonprogrammet körs på.

2.6 Robotik

Robotik är ett relativt nytt fält inom vetenskapen. Det bygger mycket på klassisk mekanik, kombinerat med elektronik. Den breda definitionen för robotik är att det är en intelligent koppling mellan avläsning och genomförande, det vill säga att gå från någon sorts indata till ge någon sorts utdata, som beror på indatan. Data från sensorer kan användas som indata till denna process. Den upptagna informationen processeras med intelligenta algoritmer. Till sist ges kommandon till robotens ställdon, alltså robotens rörliga delar. Väl anpassad hårdvara och mjukvara är därmed viktigt inom robotik för

att kunna ha en robot som både fysiskt klarar av vad det är tänkt att den ska kunna göra, samt kunna känna igen och processa all information den utsätts för. [12] Robotiken är en stor del av detta arbete, då det är just detta som Pongbot är tänkt att utföra. Att läsa in information från kameran, beräkna hur hårt och i vilken riktning armen ska kasta och sedan skicka denna information till armen som utför kastet.

2.7 Maskininlärning

En maskininlärningsalgoritm går ut på att få en robot att med indata genomföra en önskad uppgift utan att faktiskt ha kodat för att just den indatan ska ge den önskade utdatan. Istället för att beskriva en sådan algoritm som hårdkodad skulle ett mer passande uttryck vara ”mjukkodat”. Detta genom att de låter en maskin automatiskt förbättra sina funktioner och förmågor att upptäcka och processa information genom repetition, alltså erfarenhet, som gör roboten bättre och bättre på den önskade aktiviteten. Inlärningsprocessen kallas träning och går ut på att exponera roboten för mängder av olika indatan, tillsammans med önskade utdatan. Tanken med detta är att algoritmen automatiskt efter ett tag ska optimeras för att till slut inte bara kunna känna igen den exakta indatan som den stött på tidigare, utan ska kunna känna igen liknande indata som den aldrig sett tidigare och ge samma önskade utdata. [13]

Det finns många sätt att utöva maskininlärning. Ett av de bästa sätten, som är hur maskininlärning tänks användas i detta arbete, är att lära roboten tänka lite som människor gör med hur människor processar något de ser och utifrån det bestämmer vad det är. Ta som exempel att skilja på äpplen och apelsiner. Trots att varje äpple och varje apelsin är helt unik går det lätt att skilja på dem, och att veta vilken som är vilken. Detta uppnås genom träning, som lär roboten veta vad den ska leta efter och vad som kännetecknar just ett äpple, till exempel. Just detta ska appliceras för Pongbot, dock med plastmuggar istället för frukter. Det är tänkt att det ska vara plastmuggar av en viss färg Pongbot ska lära sig känna igen. Hur denna inlärning ska ske är helt enkelt genom att ställa en plastmugg med en viss färg framför kameran om och om igen och lära roboten **känna** igen den sortens plastmugg, och lära roboten vad den ska göra när den upptäcker en sådan mugg. Det är alltså som tidigare nämnt att den ska rotera så att muggen är rakt framför kameran, och sedan uppskatta avståndet till muggen.

2.8 Tidigare arbeten

Här studeras tidigare arbeten inom samma område eller som liknar detta arbete på något sätt, för att ta vara på tidigare information och erfarenheter.

2.8.1 Tidigare KEX, robotarm med sex frihetsgrader

Trots att arbetet i fråga inte gick ut på att konstruera en robotarm med samma funktioner som den som ska konstrueras i detta arbete är det fortfarande en robotarm, som medför en rad likheter mellan arbetena. Arbetet i fråga besvarar bland annat frågor om hur precis robotarmen är och hur hastigheten på roboten påverkar precisionen. Detta är också en robotarm med sex frihetsgrader, som är fler än vad som ska användas i detta projekt. Robotarmen hade både steg- och servomotorer, 3D-printade delar gjorda i CAD, en greppande klo och så vidare. Den största skillnaden är att det tidigare arbetets robotarms primära uppgift var att plocka upp och placera föremål, istället för att kasta föremål. Det resultat som detta tidigare arbete kom fram till var att precisionen generellt var ganska hög för armen, men att precisionen sjönk när flera olika frihetsgrader samarbetade och när roboten bytte axel under flyttens gång. Detta stöttar tanken om att försöka hålla detta arbete till så få frihetsgrader som möjligt för att undvika den naturliga osäkerheten som fås av många frihetsgrader. En annan slutsats från arbetet var även att precisionen sjönk om roboten arbetade snabbare. Även detta antas vara rimligt och medför frågan hur mycket precision som tappas vid en kaströrelse. [14]

2.8.2 Tidigare kastande robotarm

I detta tidigare arbete har en kastande robotarms byggts, dock en något kraftigare arm än vad som ska byggas i detta projekt. Armen i det tidigare arbete hade 2 frihetsgrader vid kastning, den använde sig av en kastmetod som kallades Rapid Semi-Optimal Motion Planning (RASMO), som automatiskt genererar de optimala banan för bollen utan justeringar för väg, hastighet osv. Armen kunde även kasta bollar i 63,7 km/h, vilket är ganska mycket snabbare kast än vad Pongbot är tänkt att klara av. Kastens precision var inte något som det togs stor hänsyn till i detta arbete, därmed blir det svårt att dra någon slutsats kring det. Generellt sett gjorde denna kastarm mycket liknande vad Pongbot ska göra, dock på en ganska mycket högre nivå med starkare motorer, fler frihetsgrader, och en mer avancerad kastalgoritm. Den slutsats som kan dras är att det definitivt är möjligt att få en kastande robotarm att kasta en lätt boll så långt som det är tänkt att Pongbot ska kunna kasta, förutsatt att de motorer som används är starka nog. Att kasta en pingisboll några få meter borde inte kräva allt för mycket kraft. Mekaniken bakom kaströrelsen i en robotarm verkar dock lovande nog för att det ska anses möjligt, trots att Pongbot endast ska kasta med en frihetsgrad och robotarmen i detta tidigare arbete hade två stycken. [15]

2.8.3 Maskininlärning för att känna igen vardagliga föremål

Detta arbete gick ut på att lära AI känna igen vardagliga föremål i hemmet, främst tänkt till folk med nedsatt syn. Detta arbete använde sig av en YOLO-struktur (You Only Look Once) som alltså innebär att algoritmen endast får en enda bild, och så skulle den även känna igen flera olika föremål i bilden. Det undersöktes om det förbättrade algoritmen om man använde färre olika föremål av samma sort vid träningen, vilket förkortade en träningscykel från 6 timmar till 45 minuter. Det visade sig att detta stämde. 107 träningscykler genomfördes, och roboten lärdes att känna igen 35 olika objekt, allt från husdjur till bestick. Trots att så många olika objekt skulle kunna kännas igen visade det sig att det fungerade och att roboten kunde känna igen föremålen med en ganska bra noggrannhet. Förvisso genomfördes väldigt mycket inlärning, men slutsatsen är ändå lovande för Pongbot. [16]

2.9 Sammanfattning

För att dra en slutsats kring teorin och alla tidigare arbeten tyder allt på att det är möjligt att konstruera Pongbot såsom det är tänkt med AI-igenkänning och arm stark nog för att kasta en lätt pingisboll några få meter. Tidigare arbeten har gjorts med både starkare armar, mer avancerad AI-igenkänning osv. Dessa projekt har varit mer omfattande och på en högre nivå än vad Pongbot kommer vara på. Däremot är det tydligt att det är möjligt att göra allt som ska göras. Det som blir mest intressant är egentligen hur pass pricksäker Pongbot kan bli, vilket också är vad som är tänkt att undersökas.

3 Metod

3.1 Ingenjörstillämpningar och metodik

3.1.1 Val av tillverkningsmetoder

Samtliga delar av robotens skelett har konstruerats med CAD i Solid Edge, och sedan skrivits ut med 3D-skrivare. Detta på grund av att modeller lätt kan konstrueras i 3D med en hög noggrannhet i CAD, och sedan tillverkas i verkligheten med i princip samma noggrannhet med 3D-skrivare. Även mer komplicerade geometrier går att tillverka på detta sätt, vilket är önskvärt. En annan möjlig tillverkningsmetod är laserskärning. Nackdelen med laserskärning är att det endast går att göra detaljer i 2D och inte i 3D, vilket behövs för de flesta delarna i detta projekt.

3.1.2 Igenkänningsmetod för muggar

Den ursprungliga tanken för hur igenkänningen av plastmuggar skulle ske var med hjälp av AI. I tidiga stadier av arbetet med koden upptäcktes dock ett bibliotek i Python som med hjälp av en kamera och några moduler lätt kan känna igen en viss färg och rita upp en ruta kring objektet. Ett bibliotek är som en samling av moduler. Detta sker utan AI, och går endast ut på att känna igen koncentrationer av en given färg. Med hjälp av rutan som ritas kan både avstånd till centrum av skärmen samt objektets storlek beräknas smidigt. Då detta är de värden som behövs för projektets applikationer ströks tanken med AI-igenkänning, och detta program kom att användas istället. En stor fördel är att hela maskininlärningsprocessen kunde strykas vilket sparar mycket tid. Detta bilbiotek i Python heter OpenCV, som står för Open Computer Vision. Just det programmet som applicerats är skrivet av Computer Science Engineer, och är taget från Github [17] och sedan omskrivet för att fungera precis som tänkt. Pyserial är även implementerat för att kunna skicka de erhållna värdena till Arduinon och därmed styra ställdonen.

3.1.3 Dator för styrning av Python och kamera

För att använda sig av en kamera krävs mer datorkraft än vad en mikrokontroller kan åstadkomma. En riktig dator behövs därför för detta. Utgångstanken för detta projekt var att använda en Raspberry Pi till detta. Tanken var att ladda ned biblioteket OpenCV på Raspberryn och att all Pythonkod skulle köras på den. Efter en mängd strul och problem med att ladda ned och att köra OpenCV på Raspberryn, kombinerat med den relativt låga datorprestandan en Raspberry Pi har jämfört med en vanlig laptop kom det till att använda en laptop istället för att använda en Raspberry Pi. Detta innebär att kameran kopplas direkt in i laptopen, och att all Pythonkod körs från laptopen istället för att köras på en Raspberry Pi. Det enda mindre problemet som uppstod som en konsekvens av detta var att en annan kamera än den erhållna behövdes, då den erhållna endast hade en ingång specifikt för en Raspberry Pi. Detta åtgärdades genom att använda en egen kamera med USB-ingång som enkelt kan kopplas in i en Laptop. Detta val gjorde att all kod i Python var mer lättillgänglig och lättare att redigera då det ligger på en laptop, moduler och bibliotek lättare att ladda ned samt att mer datorkraft finns tillgängligt då en laptop är kraftigare än den Raspberry Pi som skulle användas.

3.1.4 Beräkning av kastlängd

Den ursprungliga tanken var att enbart använda matematik för att beräkna hur långt pingisbollen ska kastas. Detta i huvudsak genom att räkna på en kastparabel med luftmotstånd. Det visade sig dock att dessa beräkningar blev omöjliga analytiskt och väldigt komplicerade numeriskt. Vidare stämde de inte överens med vad som tidigare hade observerats genom tester vilket delvis beror på att det inte är perfekta förhållanden under kastet och att förluster uppstår. Därmed uppskattades kastlängden istället genom att empiriskt testa olika rotationshastigheter för armen vid olika distanser och se vilka muggstorlekar i kameran som motsvarade vilka hastigheter, där muggstorleken används för att uppskatta avståndet till muggen. När tester vid 10 olika avstånd genomförts där olika hastigheter på armen framtogs från olika muggstorlekar i kameran användes en exponentialfunktionsberäknare online [18] där en exponentialfunktion togs fram med hjälp av minstakvadratmetoden med givna data. Denna funktion användes sedan för att uppskatta kasthastigheten vid en godtycklig muggstorlek, vilket den gjorde på en tillfredsställande nivå. Denna funktion är följande:

$$maxval = 1101.2146 * 1.0013^{cupsize}$$

Variabeln ”maxval” är i koden den minsta fördröjning mellan varje steg vid kastet, därmed ger ett mindre värde på maxval ett längre kast. Rotationshastigheten accelererar från att bero på en fördröjning på 4000 mikrosekunder till att nå maxval vid steget då klorna börjar öppna sig. Acceleration implementerades då det möjliggjorde en mycket högre hastighet vid kastet än vad konstant hastighet tillät. En begränsande faktor för den maximala hastigheten var vridmomentet. Variabeln ”cupsize” är skillnaden mellan y-koordinaten på toppen av koppen och y-koordinaten på botten av koppen, dvs y2-y1 enligt kameran. Mer om dessa i kapitel 3.5.

3.2 Evauleringsmetod av forskningsfrågor

För att undersöka hur bra Pongbot fungerar och för att besvara forskningsfrågorna har kvalitativa mätningar genomförts. För att bestämma hur långt pingisbollar kan kastas mättes avstånd från armens mitt till mitten på en plastmugg på en förbestämd sträcka. Sedan förlängdes denna sträcka tills att muggen var så långt bort att armen inte längre nådde muggen med maximal styrka på kasten. För att mäta hur stor andel av kasten som Pongbot kan träffa ställdes plastmuggen någonstans inom kamerans synfält samt på ett avstånd kortare än den maximala kastlängden. Därefter fick Pongbot själv centrera sig och utföra kastet utifrån dess beräkningar. Detta repeterades 100 gånger, och antalet kast som satt i muggen blev då pricksäkerheten, mätt i procent.

3.3 Komponenter

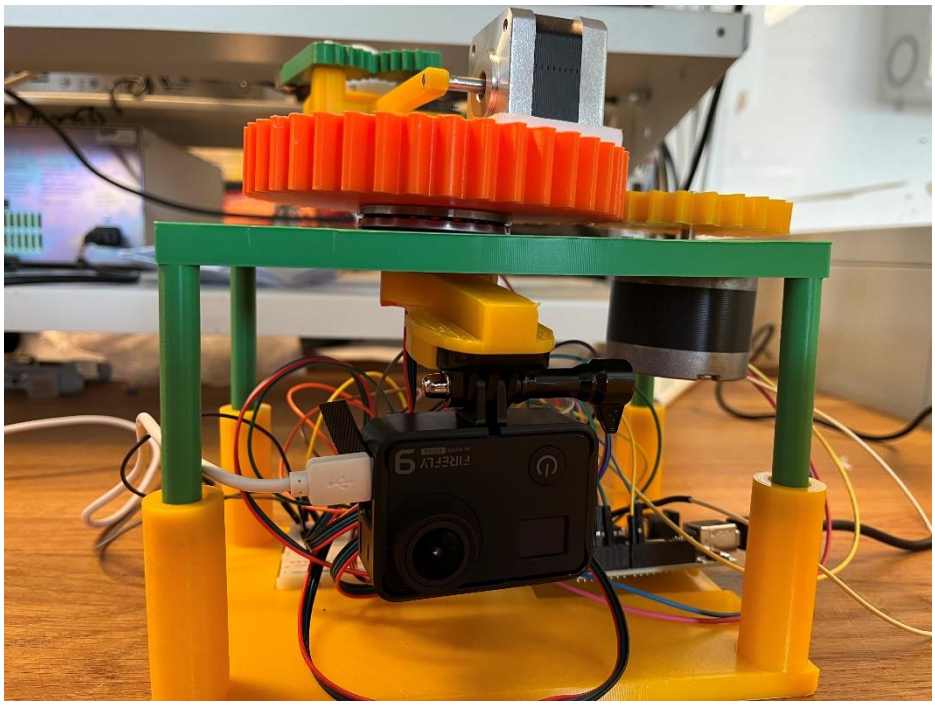
De elektroniska komponenter som använts I konstruktionen presenteras nedan I Tabell 3.3–1.

Tabell 3.3-1: Komponentlista

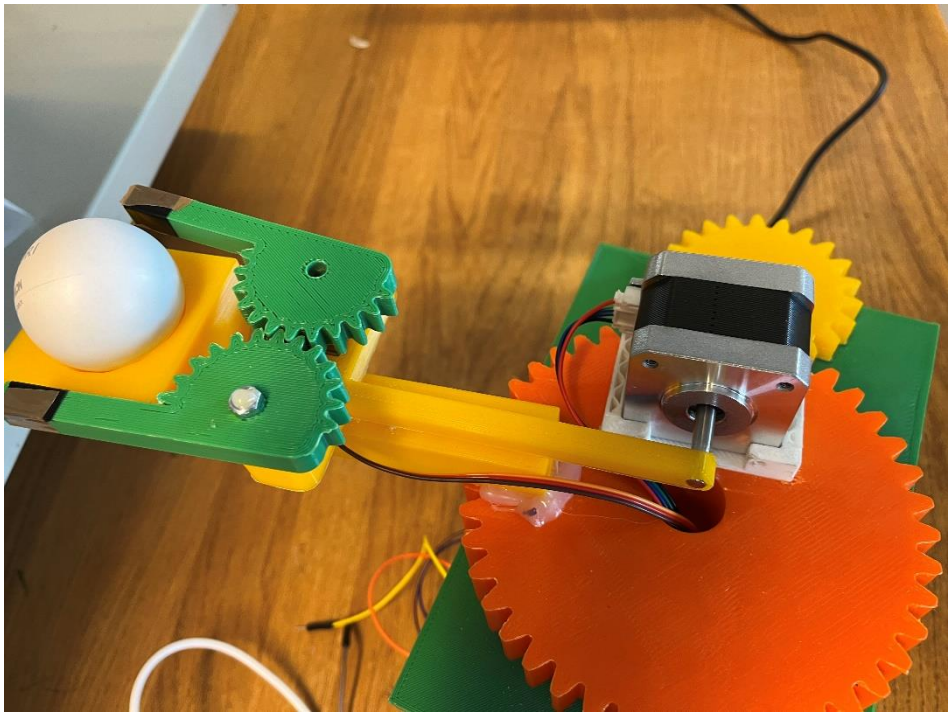
Komponent	Antal
Arduino UNO	1
Laptop (för att köra Python)	1
Breadboard	1
Servomotor	1
Stegmotorer	2
DRV8825 Stegmotordrivers	2
Kamera	1
Kondensatorer (för att minska brus)	2

3.4 Design

Nedan visas några övergripande bilder över designen av Pongbot.



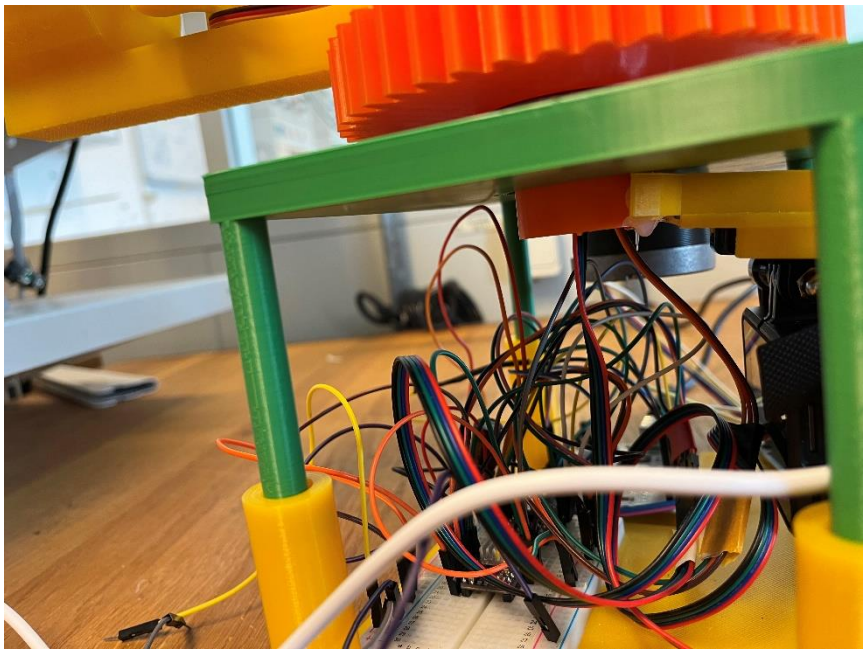
Figur 3.4-1: Pongbot framifrån



Figur 3.4-2: Pongbot från ovan

3.4.1 Bord

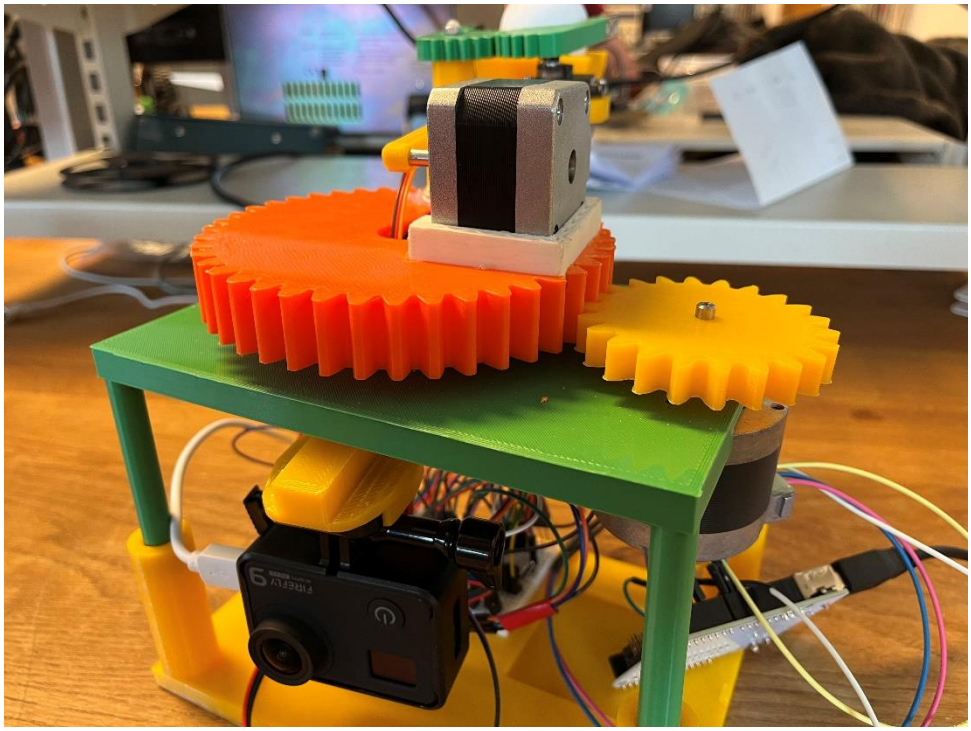
Basen av hela konstruktionen är ett bord, som det syns i Figur 3.4–1. Detta bord har utformningar för de delar som ska sitta i det, såsom det större kugghjulet samt en stegmotor. Mer om dessa i kapitel 3.4.2. Bordet är i sig uppdelat i två delar, en överdel och en underdel. Detta är för att underlätta vid 3D-utskriften. Det övre bordet ser ut som ett vanligt bord med 4 ben och en bordskiva med de tidigare nämnda utformningarna. Det undre bordet har tjockare ben som har hål i mitten, där det övre bordets ben går in. Detta gör att hela bordet förhöjs med totalt 9 cm. Den platta delen av underbordet har utformningar för att trycka i Arduinon samt Breadboarden så de inte bara ligger löst.



Figur 3.4-3: Kamerafäste på kugghjuls axel

3.4.2 Arm och rotation

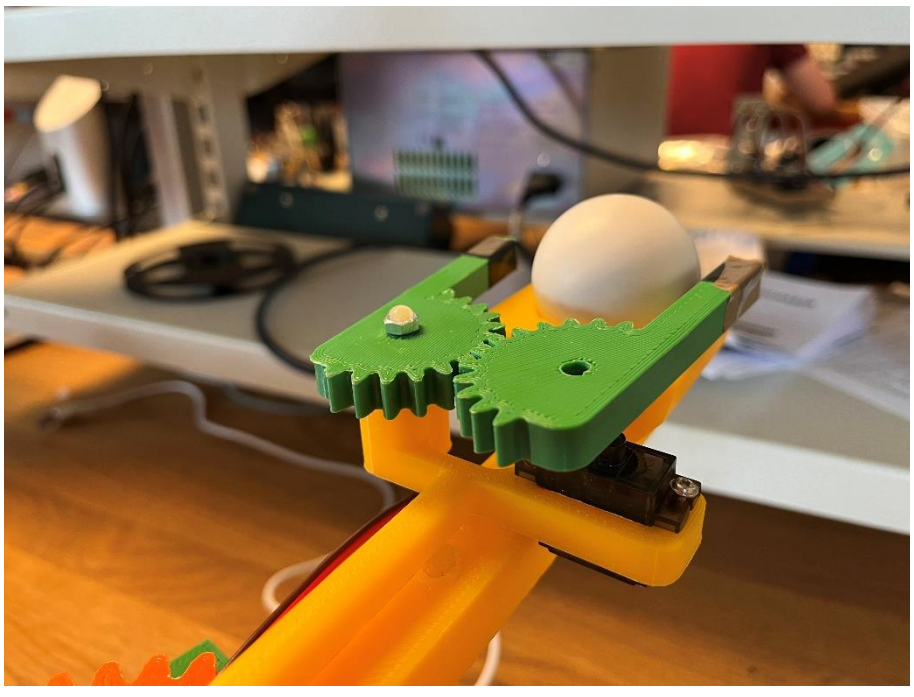
Som det syns i Figur 3.4–2 sitter armen på ett stort kugghjul, som drivs av ett drev som sitter fäst på en stegmotor. Denna stegmotor sitter på undersidan av bordskivan och skruvas fast. Dess utgående axel sticker upp genom bordet, och på den sätts drevet. På så sätt kan bordet roteras. Det är utväxling 2 mellan dessa kugghjul, vilket ger en bra nog precision på rotationen. Stegmotorn roterar ett visst antal steg, och det större kugghjulet roterar hälften så mycket som det mindre kugghjulet på grund av utväxlingen mellan dem. Det stora kugghjulet har som en ihålig nedgående axel, som går ned i hålet som finns i bordet. Bordets hål går ned i två steg, först ett steg med lite större diameter och sedan lite mindre diameter. Detta är för att det sitter ett axiallager i hålet för att ge det större kugghjulet en mjukare samt en tystare gång, samt utan friktionsförluster. Detta syns i Figur 3.4-1. Kugghjulets ihåliga axel sticker sedan ned genom bordet. Dels sitter kamerans faste fastlimmat på denna axel. Detta visas i Figur 3.4-3. Utöver det matas alla kablar genom detta hål, rakt ned till breadboarden och Arduinon som sitter längst ned. På det stora kuggjulet finns en utfasning, gjord för en liten motorlåda som en stegmotor sitter i. Denna stegmotors axel pekar ut över det stora kugghjulets centrum som är dess rotationscentrum. Denna stegmotor driver kasten, och armen sitter fast på motorns axel.



Figur 3.4-4: Rotationsmekanism

3.4.3 Greppningsmekanism

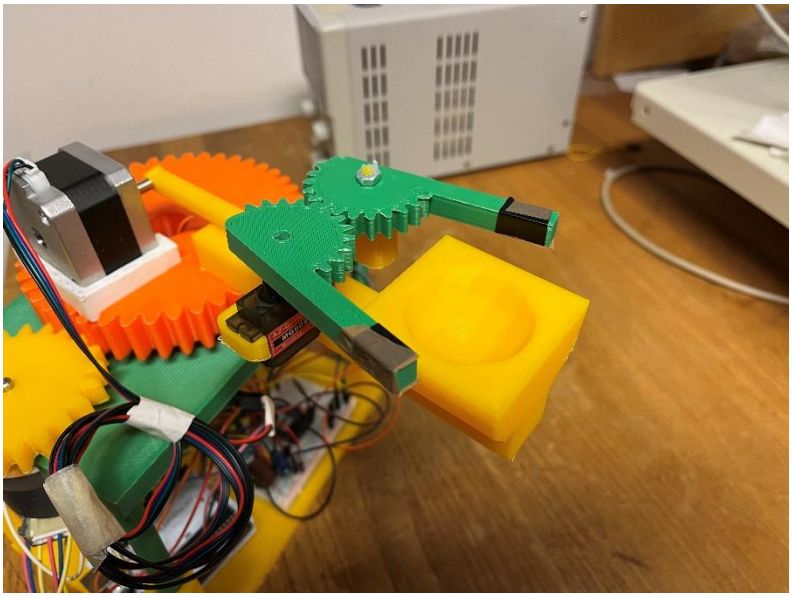
Som det syns nedan i Figur 3.4–5 sitter det som en gaffel längst ut på armen. På denna axel sitter två klor som kan greppa bollen, och släppa bollen vid kast. Den ena av dessa klor sitter fäst i en servomotor, med ett fäste geometriskt utformat för servomotorns axelfäste, som klon även är fastlimmat i. Den andra klon sitter löst på en icke-drivande axel, och sitter med en mutter. Denna klo är fri att rotera. Dessa klor har som delar av ett kugghjul vid deras bas vilket gör att båda klor drivs av endast en servomotor.



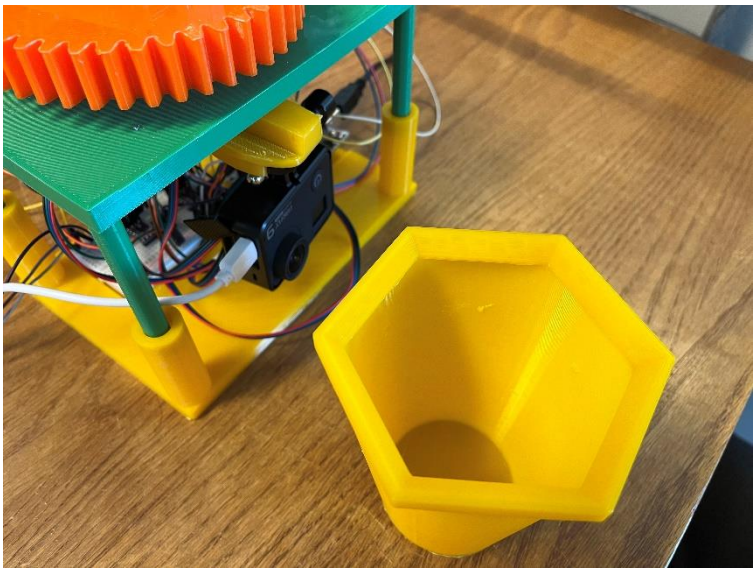
Figur 3.4-5: Klo

3.4.4 Bollkopp

Vid armens tänkta startposition som är då armen ligger ned finns en fastsatt förlängning av det roterande kugghjulet som dels är förhöjt något, och som har en skål utformad efter pingisbollens storlek. Denna skål ligger i höjd med klorna. På detta sätt kan en boll läggas i skålen, och så kan klorna stängas när de är i sitt startläge för att ta tag i en pingisboll.



Figur 3.4-6: Bollkopp



Figur 3.4-7: 3D-printad plastmugg

3.5 Kod

Den kod som används för att driva allt är uppdelad i två olika språk på två olika enheter. Dels är det ett program i C++ som körs på en Arduino Uno. Denna kod styr alla ställdon, det vill säga ger signaler till alla motorer som står för alla rörelser. Utöver detta används även en del olika program i Python, som körs på en laptop. Denna kod styr dels kameran med det tidigare nämnda programmet som använder sig av biblioteket OpenCV. Alla beräkningar med data från kameran sker även i Python. Det kan ses som att koden i Python bestämmer vad som ska göras, och koden i C++ på Arduinon verkställer det som bestäms i Python.

3.5.1 C++ (Arduino)

Högst upp i koden importeras en klass för att driva servomotorn som är en standardklass för driva en servomotor. Därefter definieras vilka pins som är kopplade till vilka komponenter så att Arduinon vet vilka pins den ska läsa av samt var den ska skicka signaler. Detta är alltså pins för motorerna. Efter allt är definierat och allt är uppstartat körs den koden som tar emot och skickar signaler om och om igen, i vad som kallas för en loop. Denna loop består av en mängd if-satser som bygger på varandra. Den grundläggande if-satsen är att koden ska börja agera om det finns ett meddelande som skickas via serieporten, dvs från koden i Python som körs samtidigt. Om det finns ett meddelande jämförs meddelandet med en mängd olika förbestämda ord, och om dessa matchar körs en bit av koden. Om till exempel meddelandet ”Close” skickas går en signal till servomotorn om att rotera några grader så att klon stänger sig, om den inte redan är stängd. Isåfall händer ingenting. Om meddelandet inte matchar med något förbestämt värde händer ingenting, och loopen fortsätter. Då går koden tillbaka till toppen och väntar på ett nytt meddelande, som det sedan återigen jämför med de förbestämda värdena. Hela koden bygger på denna princip, och finns bifogad under *Appendix A: Arduinokod i C++*.

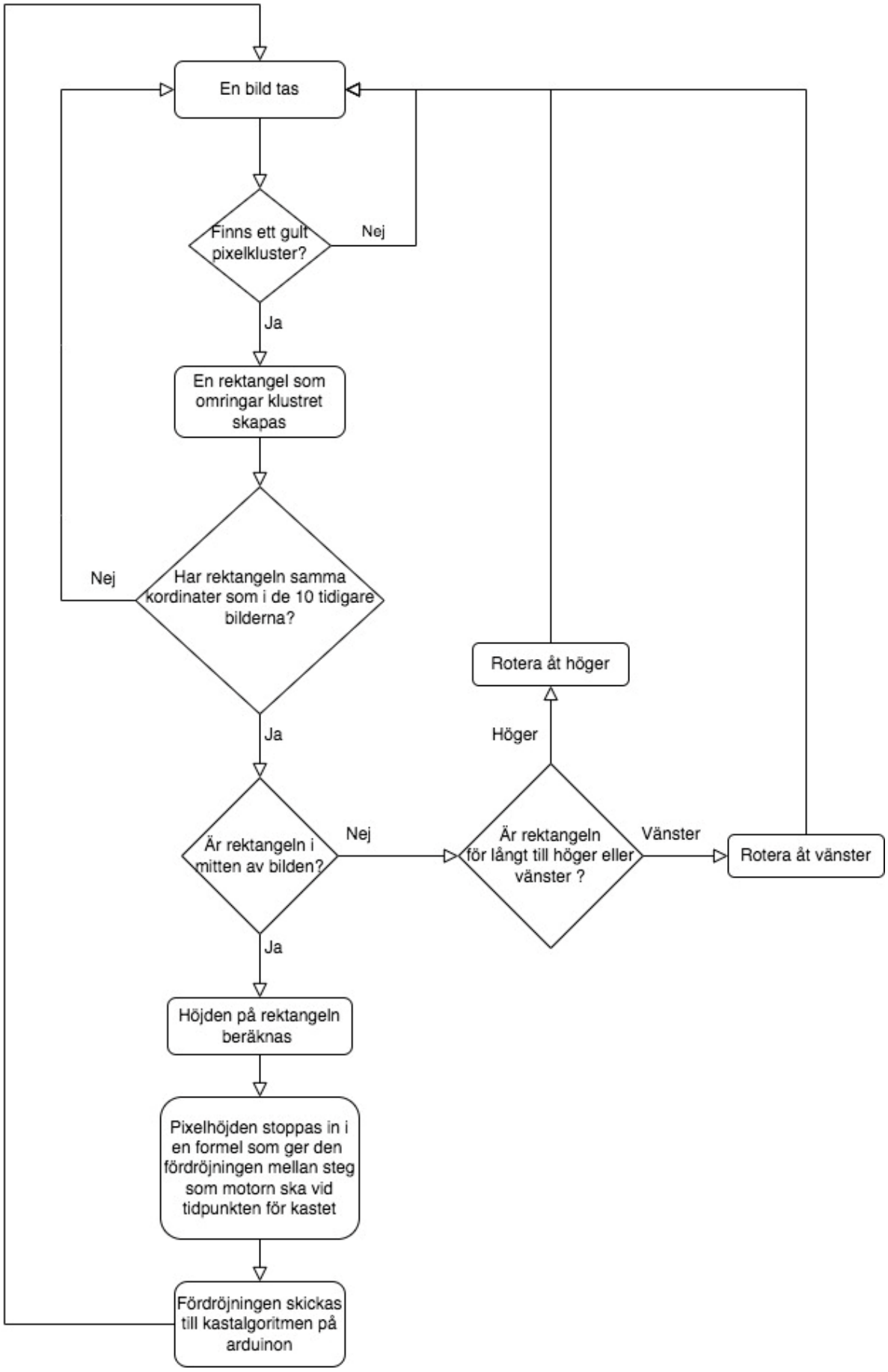
Två av kodens funktioner är något mer komplicerade, samtidigt som de fortfarande bygger på samma princip. Dessa är funktionerna för att rotera armen, samt för att kasta bollen. Båda dessa funktioner tar utöver en sträng även in ett numeriskt värde från Python. Om centrum på muggen är långt från kamerans centrum roterar armen mer än om det är närmre kamerans centrum. Det skickas därmed ett numeriskt värde på hur många steg som ska tas av den roterande stegmotorn.

Kastfunktionen är ännu mer komplicerad, även om den också är uppbyggd likadant. För att kastets drivande stegmotor ska klara av att rotera armen vid en hög nog hastighet krävs en acceleration. Den börjar väldigt långsamt och bygger upp till en topphastighet som är uppnådd strax innan bollen släpps och kastet sker. Hela kastet är ett halvt varv, som är 100 steg av stegmotorn då ett helt varv är 200 steg. Vid den 25:e steget skickas signaler till servomotorn som styr klon att öppnas, och bollen släpps. Det som styr armens hastighet är som tidigare nämnt fördröjningen mellan varje steg som stegmotorn tar. För alla kast börjar motorn rotera med en fördröjning på 4000 mikrosekunder mellan varje steg, och så går den snabbare och snabbare tills den når en fördröjning som är precis mindre än värdet på ”maxval” som är den variabel som skickas från Python efter beräkningen, och som avgör längden på kastet. Motorn accelererar därmed genom att få en kortare och kortare fördröjning, tills den när värdet på ”maxval”. När den kommer dit blir fördröjningen konstant, fram till det 50:e steget av motorn, då den börjar retardera eftersom kastet redan gjorts. Hur koden arbetar under kastet beskrivs nedan i *Figur 3.5-2*.

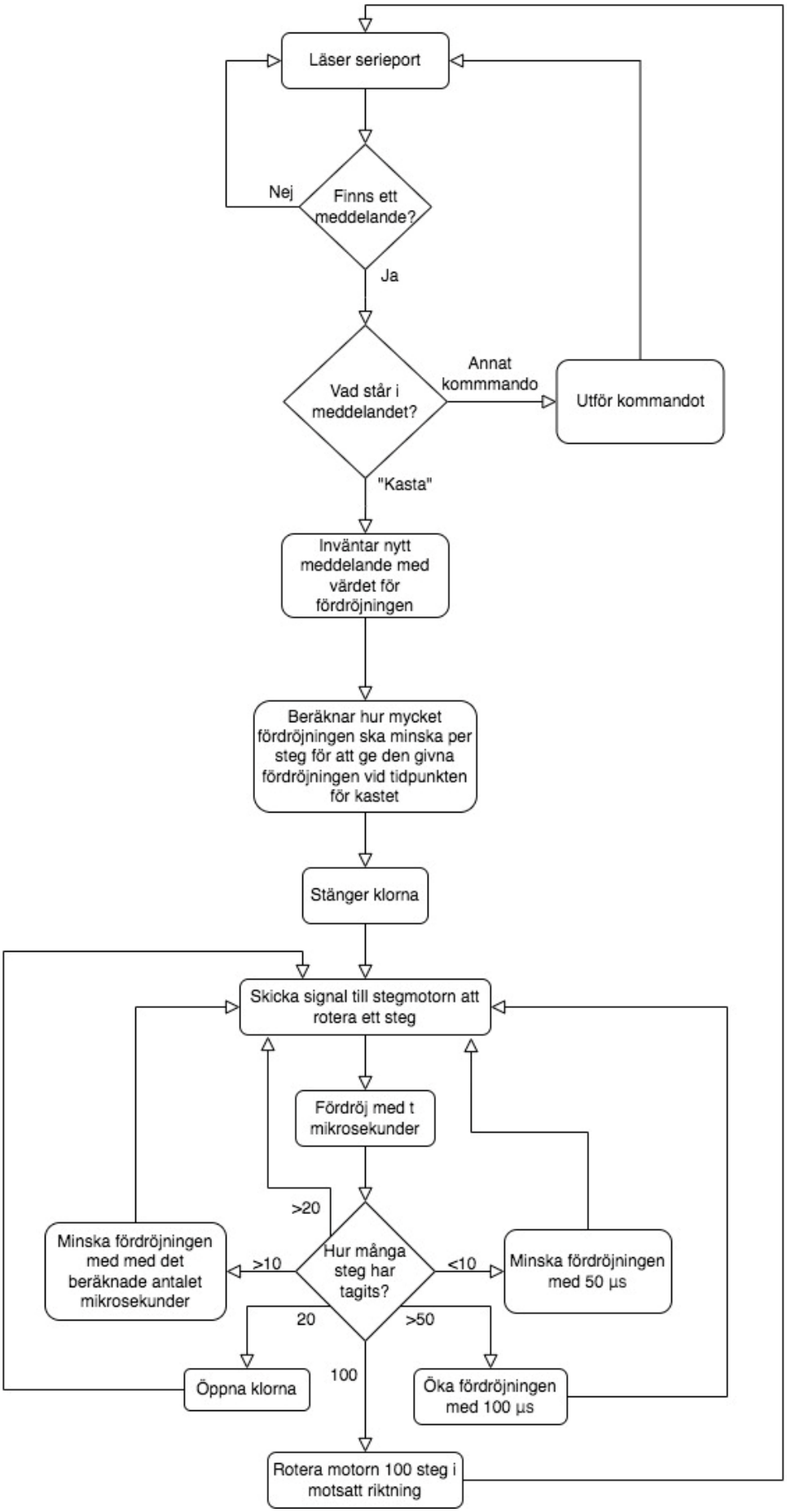
3.5.2 Python för kamera och beräkningar (Laptop)

Som tidigare nämnt använder sig koden i Python av diverse moduler och ett bibliotek. Den viktigaste modulen för att allt med läsningen av serieportarna ska fungera är Pyserial. Denna modul möjliggör

som tidigare nämnt kommunikation från Python direkt till Arduinon, som inte vanligtvis är möjligt. Det är på grund av detta som meddelanden kan skickas på sättet det beskrivs i 3.5.1. Ett viktigt bibliotek som används är OpenCV-biblioteket som står bakom färgigenkänningen som ritar upp en ruta kring objekt av den färgen som specificerats. Hur detta används i koden är genom att det först definieras som ett koordinatsystem så att det blir noll i mitten av kamerans synfält. Ett mått som kallas ”centerdistance” definieras som x-koordinaten vid ena kanten plus x-koordinaten vid andra kanten, delat på två. Detta blir hur långt rutans centrum är från mitten av kameran. Då detta avstånd är större än 7 avståndsenheter skickas signaler till Arduinon och sedan till stegmotorn om att rotera kugghjulet tills att detta avstånd är litet nog. Om avståndet är positivt eller negativt avgör rotationsriktningen. Ett numeriskt värde skickas även på avståndet, så att armen roterar mer om avståndet är större. När koppen är centrerad beräknas avståndet till muggen, genom att beräkna storleken. Storleken på objektet beräknas genom att ta den övre y-koordinaten på rutan minus den undre y-koordinaten. Storleken säger hur långt bort objektet är då koppar som är närmre kameran uppfattas som större och så vidare. Hur hårt bollen kastas varierar alltså på hur långt bort muggen uppskattas vara på detta sätt. Med hjälp av formeln som togs fram med minstakvadratmetoden som beskrivs i Kapitel 3.1.4 uppskattas den fördröjning mellan varje steg motorn som driver armen tar under kastet. Detta är värdet som kallas ”maxval” i koden. När detta har bestämts går koden in i en ny funktion som dels skickar värdet på ”maxval” till Arduinon, samt skickar signaler om att stänga klon, kasta bollen och sedan återställa sig tillbaka till startläget efter kastet. Mellan alla dessa steg finns en fördröjning på några sekunder, för att signaler ska hinnas skickas och för att låta användaren lägga tillbaka bollen i bollkoppen samt flytta på den gula muggen. Styrningen av alla delar sker i Arduinokoden och beskrivs mer utförligt i Kapitel 3.5.1. Detta är i huvudsak det som sker i Python, det vill säga användning av kameran samt beräkningar på hur långt bort koppen är samt hur långt bort från centrum den är. Ett flödesschema för hela koden finns nedan under *Figur 3.5-1*. Hela koden finns bilagad under *Appendix B. Pythonkod för kamera och beräkningar*.



Figur 3.5-1: Flödesschema för kod i Python på laptop



Figur 3.5-2: Flödesschema för kod på Arduino vid kast

4 Resultat

4.1 Resultat

Av de 100 genomförda kasterna träffade 37 kast i muggen. Medellängden för de kasten var ungefär 37 cm. Av dessa 100 kast var den längsta distansen som träffade ungefär 50 cm. Statistik med beskrivningar finns bifogat under *Appendix C: Tabell över resultat från test av Pongbot*.

4.2 Trovärdighetsanalys

De 100 kast som genomfördes anses vara tillräckligt många för att en slutsats kring noggrannheten ska kunna dras. Trots att många fler kast än 100 inte hade skadat trovärdigheten, snarare tvärtom, anses det orimligt att genomföra så många tester. Därmed anses 100 kast vara en rimlig mängd.

4.3 Giltighetsanalys

Testerna anses vara giltiga då Pongbot helt centrerades automatiskt och kastade bollen automatiskt, genom att endast ta in signaler och sedan skicka ut signaler. Centreringen skedde även mellan varje kast. Resultatet av testerna är även lätt mätbara då bollen antingen är i eller utanför muggen. Resultatet anses därmed vara giltigt.

4.4 Diskussion

Projektets huvudsakliga mål var att minst 50% av alla kast skulle sitta i koppen. Trots att detta inte uppnåddes med de 37% i koppen från testerna, anses resultatet vara godtagbart. De absolut flesta kast som inte satt i koppen studsade av koppens överkant och var väldigt nära. Gissningsvis rörde runt 80-90% av alla kast koppens ovansida och studsade ut. Den mugg som användes under testerna var 3D-printad och hade väldigt vinklade kanter, vilket gjorde att bollen väldigt lätt studsade ut om den rörde vid kanten. Med en vanlig plastmugg kan det antas att fler än 37% av alla kast sitter i muggen, då en träff på insidan av kanten skulle sitta i muggen medan en liknande träff inte gjorde det med den använda muggen. Att de absolut flesta kast var väldigt nära att träffa även om de inte gjorde det tyder också på en viss noggrannhet. Under testerna var det längsta kastet som träffade ungefär 50 cm, dock lyckades Pongbot träffa kast på ungefär 60 cm tidigare under tidigare tester. Alla kast runt 60 cm visade sig däremot under testerna vara för korta. En gissning är att den funktion som togs fram för att uppskatta kastlängden avviker för mycket från det behövda värdet för att kasta runt 60 cm för att nå så långt. För de flesta kast under 50 cm visade sig däremot funktionen fungera tillfredsställande. Om man undersöker tabellen över resultatet syns det att kortare kast generellt sett har en bättre pricksäkerhet, vilket är väldigt rimligt.

5 Slutsatser och framtida arbete

5.1 Slutsatser

Pongbot uppnådde inte den önskade pricksäkerheten på 50%. Å andra sidan rörde de flesta kast vid muggen, och de som inte träffade i muggen studsade av kanten. En viss precision uppnåddes därmed, trots att den inte var på den nivå som eftersöktes. Pongbots maximala kastlängd på ungefär 60 cm är även kortade än vad som behövs för att spela Beer Pong på ett vanligt bord. Däremot är den uppnådda längden bra relativt den motor som användes för att driva kastet. Om projektet skulle genomföras igen skulle definitivt en kraftigare stegmotor användas för att driva kastet. En annan sorts plastmugg skulle även användas vid testerna, då den som användes hade väldigt annorlunda kanter jämfört med en vanlig plastmugg vilket innebar att träffar på insidan av kanten studsade ur muggen.

5.2 Begränsningar

5.2.1 Tid och omfattning

En tydlig begränsning är projektets tid och omfattning. Då projektet genomförts av två studenter över fem månader samtidigt som andra kurser lästs, tillsammans motsvarande lika mycket högskolepoäng som projektet, har tiden och energin som lagts på projektet varit begränsad. Med mer tid hade samtliga steg kunnat genomförts med mer omsorg och omtanke.

5.2.2 Motorkraft

Den största begränsande faktorn vid kastet av pingisbollen är styrkan av stegmotorn som driver armen. Redan vid tidiga stadier upptäcktes att det inte med den använda motorn går att kasta pingisbollar längre än ungefär 60 cm. Vid försök av att driva motorn snabbare än så upptäcktes det att momentet blev för stort och att motorn antingen hoppade över några steg, eller vid alldeles för höga hastigheter endast vibrerade. Trots att motorn betedde sig som önskat vid väldigt höga hastigheter utan armen på axeln visade det sig att tröghetsmomentet från armen har en tämligen stor påverkan på hur snabbt motorns axel kan roteras.

5.2.3 Uppskattning av kastlängd

Som tidigare nämnt bestämdes funktionen för kastet genom att ta fram en funktion kring olika punkter uppmätta genom en empirisk undersökning. Denna funktion visade sig fungera någorlunda bra vid avstånd lägre än ungefär 50 cm. Däremot blev alla kast när muggen stod längre bort än så något korta. Gissningsvis är detta på grund av att funktionen avviker sig för mycket från det faktiska värdet på fördröjningen som krävs för att uppnå kastlängden. Trots att Pongbot är kapabel att träffa muggar på avstånd runt 60 cm, var detta då inte möjligt med den framtagna funktionen.

5.3 Framtida arbete

Ett vidarearbete på Pongbot skulle vara en liknande robot med längre arm som ger ett större moment. För att driva denna arm skulle en kraftigare stegmotor behövas. Med en kraftigare motor borde armen även gå att drivas vid ännu högre hastigheter vilket också medför en längre kastlängd. Utöver det är ett vidarearbete en mer exakt modell av kastet, exempelvis en numerisk modell som på något sätt behandlar luftmotståndet. Gissningsvis kan en sådan modell bli mer noggrann än en modell framtagen empiriskt, trots att den empiriska i stora lag fungerar väldigt bra.

5.4 Reflektioner

Gruppen är överlag nöjd med resultatet från projektet och hur Pongbot blev, trots att det huvudsakliga målet inte uppfylldes om att träffa 50% av kasten. Då en viss precision ändå är tydlig och då alla komponenter samarbetar väl samt att allt fungerar helt automatiskt känner gruppen att grundtanken kring tanken av hur Pongbot skulle uppföra sig ändå är uppfyllt. Projektet genomfördes inom den utsatta tidsramen utan att gruppen behövde stressa mer än nödvändigt, och gruppen klarade sig väl inom den givna budgeten med en stor marginal. Det som skulle kunnat gått bättre är att en högre precision skulle kunna uppnås samt att längre kast skulle kunna uppnås, men om Pongbot skulle vidarearbetas som föreslagit anser gruppen att dessa aspekter hade förbättrats.

Referenser

[1] Haider A. F. Almurib et al. *A review of application industrial robotic design. (2011)*

A review of application industrial robotic design | IEEE Conference Publication | IEEE Xplore

[2] Akira Takahashi et al. *Dynamic Compensation in Throwing Motion with High-Speed Robot Hand-Arm. (2021)*

https://ieeexplore.ieee.org/abstract/document/9560866

[3] John Davies. *MSP430 Microcontroller Basics*. (2008)

https://radfiz.org.ua/files/k3/kaf/medu/smt/ftp.univ.kiev.ua/.incoming/smt/doc/MSP430%20Microcontroller%20Basics.pdf

[4] "What is Arduino?" Arduino.cc. Accessed: Feb. 5, 2018. [Online.] Available:

https://www.arduino.cc/en/Guide/Introduction

[5] Alisher Shakirovich Ismailov et al. *Study of arduino microcontroller board*. (2022)

https://www.researchgate.net/profile/Alisher-Ismailov/publication/359502443_Study_of_arduino_microcontroller_board/links/62402fca8068956f3c50ea36/Study-of-arduino-microcontroller-board.pdf

[6] Matt Richardson & Shan Wallace. *Getting started with Raspberry Pi*. (2013)

https://learning.oreilly.com/library/view/getting-started-with/9781680452457/ch06.html

[7] Benetta Aranjo et al. *Stepper motor drives for robotic applications. (2012)*
Stepper motor drives for robotic applications | IEEE Conference Publication | IEEE Xplore

[8] Ammar Arif Ansari. *A Review of Different Motor Types and Selection of One Optimal Motor for Application in EV Industry. (2022)*

https://www.researchgate.net/profile/Ammar-Arif-Ansari/publication/361890047_A_Review_of_Different_Motor_Types_and_Selection_of_One_Optimal_Motor_for_Application_in_EV_Industry/links/62cada77d7bd92231fa720b9/A-Review-of-Different-Motor-Types-and-Selection-of-One-Optimal-Motor-for-Application-in-EV-Industry.pdf

[9] Robert Sheldon, Tech Target, *Definition, degrees of freedom (mechanics). (2022)*

https://www.techtarget.com/whatis/definition/degrees-of-freedom

[10] Python.org, The Python Tutorial, *6. Modules*. (2024)

https://docs.python.org/3/tutorial/modules.html

[11] Chris Liechti, PythonHosted.org, *pySerial 3.0 documentation. (2015)*

https://pythonhosted.org/pyserial/

[12] A. Ollero et al. *Mechatronics, robotics and components for automation and control: IFAC milestone report. (2006)*
Mechatronics, robotics and components for automation and control: IFAC milestone report - ScienceDirect

[13] Issam El Naga, Martin J. Murphy. *What is Machine Learning? (2015)*

What Is Machine Learning? | SpringerLink

[14] Michal Gabriel Sawczuk, *Design and control of a 3D printed, 6 DoF robot arm. (2021)*

FULLTEXT01.pdf (diva-portal.org)

[15] Chyon Hae Kim, Shigeki Sugano, *Executing optimized throwing on robot arm with free joint. (2016)*

Full article: Executing optimized throwing motion on robot arm with free joint (tandfonline.com)

[16] Tharangini Sankarnarayanan et al. *Training AI to recognize objects of interest to the blind and low vision community. (2023)*
Training AI to Recognize Objects of Interest to the Blind and Low Vision Community | IEEE Conference Publication | IEEE Xplore

[17] Computer vision engineer, *color-detection-opencv. (2023)*

https://github.com/computervisioneng?tab=repositories&q=opencv&type=&language=&sort=

[18] Stats,Blue, *Exponential Regression Calculator (u. å.)*
https://stats.blue/Stats_Suite/exponential_regression_calculator.html

Appendix A: Arduinokod I C++

```
#include <Servo.h>

// Define stepper motor connections and steps per revolution:
#define dirPin 2
#define stepPin 3
#define dirPinSpin 8
#define stepPinSpin 9
#define stepsPerRevolution 200

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position
int nummsg = -1;

void setup() {
  myservo.attach(5);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(stepPinSpin, OUTPUT);
  pinMode(dirPinSpin, OUTPUT);
  Serial.begin(115200);
  myservo.write(20);
}

void loop() {
  if (Serial.available() > 0){
    String msg = Serial.readString();
    Serial.print(msg);
    if (msg == "Close"){
      for (pos = 20; pos <= 38; pos += 1) { // goes from 0 degrees to 180 degrees
      // in steps of 1 degree
      myservo.write(pos);              // tell servo to go to position in variable 'pos'
      delay(5);     }                  // waits 15ms for the servo to reach the position
  }

  else if (msg == "Rightspin"){
    digitalWrite(dirPinSpin, LOW);

  }

  else if (msg == "Leftspin"){
  digitalWrite(dirPinSpin, HIGH);

  }
  else if (msg == "Spin"){
    while (Serial.available() == 0); {
    }
    String msg2 = Serial.readString();
    nummsg = msg2.toInt();
    Serial.print(nummsg);
    if (nummsg >= 5){
      for (int i = 0; i < nummsg; i++) {
        digitalWrite(stepPinSpin, HIGH);
        delayMicroseconds(16000);
        digitalWrite(stepPinSpin, LOW);
        delayMicroseconds(16000);
      } }
    else {
      for (int i = 0; i < 2; i++) {
        digitalWrite(stepPinSpin, HIGH);
        delayMicroseconds(16000);
        digitalWrite(stepPinSpin, LOW);
        delayMicroseconds(16000);
    } }
  }
```

```
  else if (msg == "Open"){
    for (pos = 38; pos >= 20; pos -= 1) { // goes from 180 degrees to 0 degrees
      myservo.write(pos);              // tell servo to go to position in variable 'pos'
      delay(5);     }                  // waits 15ms for the servo to reach the position
  }
  else if (msg == "Throw"){
    digitalWrite(dirPin, LOW);
    while (Serial.available() == 0); {
    }
    String msg3 = Serial.readString();
    int maxval = msg3.toInt();
    Serial.print("Maxval is ");
    Serial.print(maxval);
    int t = 4000;
    int k = (3500-maxval)/10;
    Serial.print(k);
    int count = 0;

    digitalWrite(stepPin, HIGH);
    delayMicroseconds(5000);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(500000);

   for (int i = 0; i < stepsPerRevolution/2; i++) {
     // These four lines result in 1 step:
     if (i >= 50){
       t = t+100;}
     else if (t <= maxval){
       }
     else if (i <= 10){
       t = t - 50;
      }
     else if (count >= 1){
       count = 1;
       t = t-k;}
     else{
       count++;}
     Serial.print(t);
     Serial.print(", ");
     digitalWrite(stepPin, HIGH);
     delayMicroseconds(t);
     digitalWrite(stepPin, LOW);
     delayMicroseconds(t);
       if (i == 20){
         for (pos = 38; pos >= 20; pos -= 1) { // goes from 180 degrees to 0 degrees
           myservo.write(pos);              // tell servo to go to position in variable 'pos'
            }
           i++;}
}}
   else if (msg == "Reset"){
     digitalWrite(dirPin, HIGH);
     for (int i = 0; i < 100; i++) {
       // These four lines result in 1 step:
       if (i < 70){
       digitalWrite(stepPin, HIGH);
       delayMicroseconds(5000);
       digitalWrite(stepPin, LOW);
       delayMicroseconds(5000);
     }
      else{
        digitalWrite(stepPin, HIGH);
        delayMicroseconds(10000);
        digitalWrite(stepPin, LOW);
        delayMicroseconds(10000);}
      }
      digitalWrite(dirPin, LOW);
  }

  }
}
```

Appendix B: Pythonkod för kamera och beräkningar

```
# pip install pyserial
import serial.tools.list_ports
import cv2
from PIL import Image
from util import get_limits
import time
import math

ports = serial.tools.list_ports.comports()
serialInst = serial.Serial()

serialInst.baudrate = 115200
serialInst.port = "COM5"
serialInst.open()


def skicka():
    while True:
        command = input("What activity do you want to perform? \n")
        serialInst.write(command.encode('utf-8'))

        if command == "exit":
            exit()

def prata():
    while True:
        command = input("What activity do you want to perform? \n")
        serialInst.write(command.encode('utf-8'))

        if command == "Spin":
```

```
                time.sleep(1.1)
                command2 = "10"
                serialInst.write(command2.encode('utf-8'))

def outputs():
    if camerafunction():
        serialInst.write("Close".encode('utf-8'))
        exit()
    outputs()

def throw(maxval):
    serialInst.write("Close".encode('utf-8'))
    time.sleep(1)
    serialInst.write("Throw".encode('utf-8'))
    time.sleep(1.1)
    serialInst.write(str(maxval).encode('utf-8'))
    time.sleep(5)
    serialInst.write("Reset".encode('utf-8'))
    time.sleep(4)
    camerafunction()


def camerafunction():
    yellow = [0, 255, 255]  # yellow in BGR colorspace
    cap = cv2.VideoCapture(0)

    ifcenterlist = []
```

```python
while True:
    ret, frame = cap.read()

    hsvImage = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lowerLimit, upperLimit = get_limits(color=yellow)

    mask = cv2.inRange(hsvImage, lowerLimit, upperLimit)

    mask_ = Image.fromarray(mask)

    bbox = mask_.getbbox()

    if bbox is not None:
        x1, y1, x2, y2 = bbox

        frame = cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 5)

        cupsize = (y2-y1)
        #cupdistance = 84.2049*0.9956**cupsize
        a = 2.98
        b = 1000.01

        maxval = 1101.2146 * 1.0018**cupsize
```

```python
    cupcenter = ((x1+x2)/2) - 330
    #print("x1 = "+ str(x1))
    #print("x2 = " + str(x2))
    print(cupcenter)
    tol = 7
    if abs(cupcenter) < tol:
        ifcenterlist.append(cupcenter)
    else:
        ifcenterlist = []

    if len(ifcenterlist) == 10:
        throw(maxval)
        print(f"Cupsize is {cupsize}")
        time.sleep(5)

    # Centrering

    if abs(cupcenter) >= tol:
        if cupcenter > 0:
            serialInst.write("Rightspin".encode('utf-8'))
        else:
            serialInst.write("Leftspin".encode('utf-8'))
        time.sleep(1.1)
        print(f"cupcenter är {cupcenter}")
        cupcenter1 = str(round(abs(cupcenter/8)))
        serialInst.write("Spin".encode('utf-8'))
        time.sleep(1.1)
        serialInst.write(cupcenter1.encode('utf-8'))
        time.sleep(1.1)
```

```
        cv2.imshow('frame', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()

    cv2.destroyAllWindows()

if __name__ == "__main__":
    camerafunction()
```

Appendix C: Tabell över resultat från test av Pongbot

Kast	I	U	Ungefärlig distans	Kommentar
1		1	40	Kort
2		1	41	Lite lång, väldigt nära
3		1	45	Kort
4		1	35	Studsade ur, lite sne
5		1	30	Kort, studsade ur
6		1	35	Jättenära, lite långt
7	1		25	
8		1	45	Bakkant lite långt
9		1	50	Ganska kort
10		1	50	Stolpe ut, lite sned
11		1	35	Var i men studsade ur på kanterna
12		1	30	Lite kort, träffade främre kant
13	1		20	
14	1		45	
15		1	40	Fick inte iväg bollen
16		1	35	Kort, träffade framsidan
17		1	35	Höll inte i bollen
18		1	35	Skippade steg
19	1		25	
20		1	30	Lite kort
21		1	25	Lite lång
22	1		40	
23		1	35	Lite lång, träffade bakkant
24		1	35	Kort, träffade framsida
25	1		25	
26	1		25	
27		1	60	Ganska mycket för kort
28		1	60	Ganska mycket för kort
29		1	55	Framkant
30		1	50	Tappade bollen i kastet
31		1	50	Ganska mycket för kort
32	1		45	Med studs i bordet, (tappade bollen)
33	1		45	
34		1	40	Kant kant ur
35		1	40	Främre kant, lite kort
36		1	35	Främre kant, lite kort
37		1	30	Bakkant, lite lång
38		1	25	Fämre kant
39	1		25	
40		1	20	Lite sne, studsade på insidan
41	1		20	
42		1	30	Bakre insida kant, lite lite lång
43		1	35	Bakre kant, lite lång
44		1	40	Bakre kant, lite lång
45	1		40	
46		1	40	Främre kant, lite kort
47		1	40	För kort, spökade i kameran
48	1		45	
49		1	45	Främre kant, lite kort
50		1	45	Framsida kopp, kort

51		1	50	Ganska kort
52		1	50	Framsida kopp, kort
53		1	20	Studsade ur
54	1		20	
55	1		20	
56		1	25	Främre kant, lite kort
57	1		25	
58	1		25	
59	1		30	
60	1		30	
61	1		30	
62		1	40	Främre kant, lite kort
63		1	40	Bakre inre kant
64		1	40	Framsida kopp, kort
65		1	45	Främre insida kant, lite kort
66	1		45	
67	1		45	
68	1		45	
69		1	30	Bakre kant, lite långt
70	1		35	
71		1	35	Framkant, lite kort
72		1	30	Bakkant, lite lång
73		1	30	Bakkant, lite lång
74	1		30	
75	1		30	
76		1	40	Framkant, lite kort
77		1	40	Framkant, lite kort
78		1	40	Framkant, lite kort
79		1	45	Ganska mycket för kort
80		1	45	Framkant, lite kort
81	1		25	
82	1		25	
83	1		20	
84	1		20	
85	1		45	
86	1		45	
87	1		45	
88		1	52	Väldigt kort
89		1	52	kort
90		1	40	Bakre kant lite lång
91	1		47	
92	1		50	
93		1	55	Kort, framsida kopp
94		1	55	Framkant, väldigt nära
95		1	55	Kort
96		1	30	Kort, lite sne
97		1	30	Bakre kant
98		1	25	Bakre kant, lite långt
99	1		25	
100	1		25	

Statistik	
I	37
Utanför	63
Totalt	100
Rate	0,37
Medellängd	36,92

TRITA – ITM-EX 2024:52

www.kth.se