Arduino robotarm

Casimir Kristén

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för Automationsteknik och IT

Raseborg 2016

EXAMENSARBETE

Författare:	Casimir Kristén
Utbildningsprogram och ort:	Automationsteknik och IT, Raseborg
Inriktningsalternativ/Fördjupning:	Elplanering
Handledare:	Kim Roos

Titel: Arduino robotarm

Datum: 6.5.2016 Sidantal: 39 Bilagor: 3

Abstrakt

Syftet med detta arbete är att ge läsaren en grundläggande information om Arduino och hur en robotarm kan styras med hjälp av ett Arduino kort. Arbetet inleds med grundfakta om mikrokontroller och om olika kommunikationsprotokoll. Efter det introduceras läsaren till Arduino, vilket innefattar allmän information, hur det fungerar och de olika modellerna. I arbetet får läsare också bekanta sig med Arduino IDE, programmet som används i samband med Arduino och sensorer samt tillläggsmoduler, som används i samband med Arduino. Utöver det behandlas det också mer ingående om servon, eftersom de är en viktig del av en robotarms funktion.

Arbetet avslutas med en genomgång av hur en robotarm skapades, vilket inleds med en planering av armens konstruktion, hårdvara, val av servon och Arduino modell. För att sedan avslutas med en genomgång av programkoden som skrevs för styrning av robotarmen.

Språk: Svenska Nyckelord: Arduino, robotarm, servo

OPINNÄYTETYÖ

Tekijä:	Casimir Kristén
Koulutusohjelma ja paikkakunta:	Automationsteknik och IT, Raasepori
Suuntautumisvaihtoehto/Syventävät opinnot:	Sähkösuunnittelu
Ohjaaja:	Kim Roos

Nimike: Arduino robottikäsivarsi

Päivämäärä: 6.5.2016 Sivumäärä: 39 Liitteet: 3

Tiivistelmä

Tämän opinnäytetyön tarkoituksena on antaa lukijalle perustietoja Arduinosta ja siitä, miten robottikäsivartta ohjataan Arduino-kortin avulla. Työ alkaa mikro-ohjaimien ja tietoliikenne-protokollien perustiedoilla. Sen jälkeen lukija tutustutetaan Arduinoon kertomalla yleisiä tietoja siitä, miten se toimii ja mitä eri malleja on olemassa. Opinnäytetyössä lukija saa tutustua Arduino IDE:hen, Arduinon yhteydessä käytettävään ohjelmaan, sensoreihin, lisämoduuleihin, joita käytetään Arduinon yhteydessä, tämän lisäksi käsitellään perusteellisemmin servoja, koska ne ovat tärkeä osa robottikäsivarren toimintaa.

Opinnäytetyön lopussa käydään läpi, miten robottikäsivarsi luotiin, alkaen käsivarren suunnitellusta, laitteistosta, servojen ja Arduinon mallin valinnasta. Lopuksi käydään läpi käsivarren ohjaamiseen luotu ohjelmakoodi.

Kieli: Ruotsi Avainsanat: Arduino, robottikäsivarsi, servo

BACHELOR'S THESIS

Author:	Casimir Kristén
Degree Programme:	Automation Engineering and IT, Raseborg
Specialization:	Electrical System Design
Supervisor:	Kim Roos

Title: Arduino Robotic Arm

Date: 6.5.2016 Number of pages: 39 Appendices: 3

Summary

This thesis provides basic knowledge about Arduino and how you can control a robotic arm with an Arduino board. The first sentence of this thesis presents basic facts about microcontrollers and about different communication protocols. After that the reader will be introduced to Arduino, which includes what it is, how it works and the different boards used with it. The Reader will also get familiar with Arduino IDE, the program that is used, different sensors and shields that can be used with Arduino, as well as servos and how they work.

This thesis ends with a description of how a robotic arm was created, starting with the planning of the arm itself, hardware, servos and choosing an Arduino board, then the program code is explained.

Language: Swedish Key words: Arduino, robotic arm, servo

Innehållsförteckning

1 Inledning

Under de senaste åren har det dykt upp allt fler så kallade mikrokontroller kort, vilket har gjort det allt lättare före både amatörer och yrkesmän att skapa automationsbaserade projekt, till ett lägre pris. Projekten kan variera från att styra en lampa till mera avancerade projekt som 3D skrivare och automatiserade fordon i mindre skala.

Jag valde att planera och konstruera en robotarm som baserades på ett Arduino mikrokontroller kort. En robotarm valdes främst för att det var en bra grund att starta ifrån och för det ger goda möjligheter till att expandera med olika funktioner, som bland annat styrning med hjälp av sensorer, samt även kunna styra robotarmen via till exempel en hemsida med hjälp av en trådlösnätverksmodul eller via telefonen med hjälp av Bluetooth. Tanken bakom arbetet var att få en bättre praktisk erfarenhet av Arduino och för att samla grundläggande information om Arduino men också för att kunna pröva den teori som behandlas i arbetet i ett praktiskt sammanhang.

Arbetet baserades till en stor del på att kunna styra servomotorer med hjälp av Arduino, på både ett manuellt och automatiskt sätt. Utöver att kunna styra servomotorer valde jag också att använda mig av närvarosensorer, framför allt av säkerhetsskäl, men även för att kunna styra servona genom att skicka en signal till mikrokontrollen.

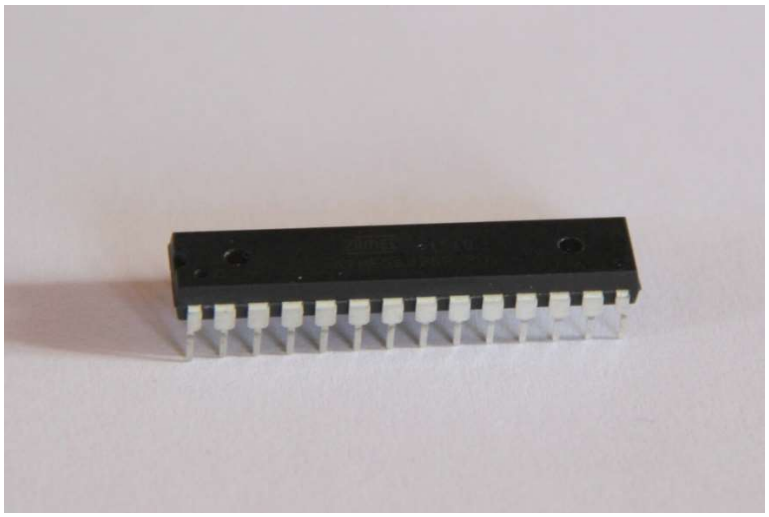
I det här arbetet behandlas den baskunskap som behövs för att kunna genomföra ett liknande projekt.

2 Mikrokontroller

En mikrokontroll kan beskrivas som en liten dator som är integrerat i ett system där den utför en färdigt definierad uppgift. Idag finns mikrokontrollers inbyggda i de flesta elektroniska apparaterna, både i hemmet och inom industrin. Man finner dessa i bland annat bilar, båtar, diskmaskiner, digitalklockor, och tangentbord. Listan kan göras lång med det ger en inblick i hur vanligt förekommande en mikrokontroll egentligen är. (Choudhary, 2012).

Redan år 1971 släpptes den första mikrokontrollen av Intel. Intel 4004 var en 4-bit mikrokontroll med en klockhastighet på 740 kHz. Ett år senare släppte Intel sin Intel 8008, som var en 8-bit mikrokontroll med klockfrekvensen 0,8 MHz. Under åren har mikrokontrollen utvecklats allt mer. I dagens läge finns det 4-bit, 8-bit, 16-bit och 32-bit mikrokontrollers med varierande klockfrekvens, som 4 KHz, 16 MHz, 48 MHz, 84 MHz. Till dem största tillverkarna av mikrokontrollers idag räknas: Atmel, Microchip, Texas Instruments, Freescale, Philips och Motorola. (Choudhary, 2012) (Wikipedia, 2009).

I figur 1 ser vi en mikrokontroll av Atmel och deras modell ATmega328P.



Figur 1. Atmel ATmega328P mikrokontroller. (Egen bild)

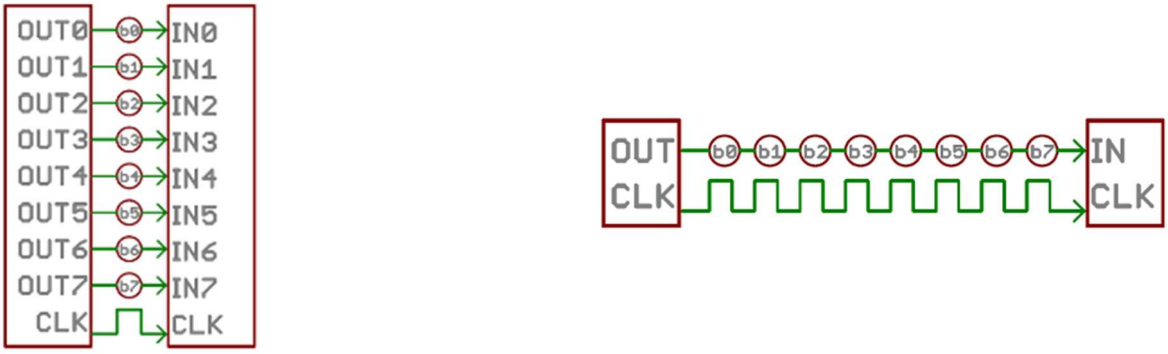
För att en mikrokontroll skall veta vad för uppgift den har, måste man programmera mikrokontrollen och definiera uppgiften. Beroende på tillverkare och modell varierar det vilket programmeringsspråk som används. Till de vanligaste programmeringsspråken som används för att programmera är: C/C++, Java, Actionscript och Assembler. Atmels AVR mikrokontrollers som finns integrerat på Arduino kretskort kan programmeras med Arduinos egna utvecklade språk/program. (O'Sullivan, u.å.).

En mikroprocessor skall inte blandas ihop med en mikrokontroll även om de kan visuellt se likadana ut. Mikroprocessorn har inte bland annat inbyggt minne, vilket en mikrokontroll har. För att en mikroprocessor skall fungera måste tillverkar lägga till det själv. Arbetsprincipen skiljer sig också åt från de både. En mikrokontroll gör en typ av definierad uppgift medan igen en mikroprocessor har utvecklats för att göra flera odefinierade uppgifter. (Choudhary, 2012).

3 Kommunikation

För att kunna skicka och ta emot data från en givare till ett Arduino kortet behövs någon form av kommunikation som bägge delarna kan läsa och tolka rätt. Det är också viktigt för användare att den vet skillnad mellan olika former av kommunikationssätt som finns. Att till exempel känna till skillnader mellan serie- och parallellkommunikation samt vad följande förkortningar betyder I^2C, SPI, 1-Wire osv.. Genom att vara medveten om skillnader kan man lättare välja bland lämpliga komponenter och skriva koden på rätt sätt, samt säkerställa sig att allting fungerar som beräknat.

Man kan indela de kommunikationsmetoder som används i två huvudgrupper, serie- och parallellgränssnitt, som sedan i sin tur kan indelas i undergrupper. Genom att se på figur 2 kan man snabbt se att det förekommer skillnader mellan serie- och ett parallellgränssnitt. Framförallt kan man se att antal kablar som behövs för att kunna skicka data varierar. Figur 2 illustrerar också hur data skickas i de båda fallen. Till höger ser vi att vid ett parallellt gränssnitt skickas flera bits samtidigt per klockslag och av den orsaken behövs det flera kablar. Det betyder också att vi kan snabbare skicka data. Vid ett seriegränssnitt till vänster, kan vi se att vi klara oss med endast två kablar för att kunna skicka samma mängd data. Till skillnad från ett parallellgränssnitt skickas endast en bit per klockslag, vilket betyder att varje bit levereras i tur och ordning men långsammare. (Sparkfun, 2012).



Figur 2. Skillnad mellan ett parallellt- och ett seriegränssnitt. (Sparkfun, 2012)

På Sparkfun (Sparkfun, 2012) använde de sig av ett mycket bra beskrivande exempel angående skillnaden mellan ett parallell- och seriegränssnitt. Man kan se ett parallellgränssnitt som en väg med flera filer, som en motorväg och ett seriegränssnitt som en gammal landsväg. Både vägarna har samma funktion, skillnaden är givetvis att på en motorväg ryms det mera bilar samtidigt och med en högre hastighet, till skillnad från en landsväg. Fördelen med landsvägen är att kostanden är betydligt lägre än vad den är för en motorväg, likaså är också seriegränssnittet. I och med att det krävs mindre antal kablar och till fördel för kostanden används seriekommunikation mera med Arduino kort. Därför behandlas seriekommunaktion djupare i detta kapitel, samt även olika varianter av seriegränssnitt.

3.1 Serie

Seriekommunikation är inte enbart något som hör ihop med Arduino utan seriekommunikation används i stor utsträckning. De mest kända formerna av seriekommunikation är troligtvis USB (Universal Serial Bus) och Ethernet, som används bland annat i datorer och mobiltelefoner. Utöver de både nämna exemplen finns det även andra serieprotokoll som används. Till dem hör bland annat I^2C, UART och SPI. Alla de protokoll kan i sin tur indelas i två huvudgrupper, asynkront och synkront seriegränssnitt.

Då man talar seriekommunikation är det vanligtvis ett asynkront gränssnitt man tänker på. Detta gränssnitt kännetecknas av TX, RX och GND märkningar vid ut- och ingångarna. I ett asynkront seriegränssnitt levereras data utan en klocksignal, vilket betyder att det behövs färre kablar men risken för att data inte levereras eller störningar ökar. Vid ett synkront seriegränssnitt är alla delar synkroniserade med samma klockfrekvens, vilket betyder att data

levereras i ordning, det leder i sin tur till snabbare överföring. Nackdelen med ett synkront gränssnitt jämför med ett asynkront gränssnitt är att det krävs en extra kabel för klocksignalen. (Sparkfun, 2012).

Ett seriegränssnitt kan vara full-dublex, half-dublex eller simplex. Vid ett full-duplex seriegränssnitt kan både mottagaren och sändare skicka data till varandra samtidigt. Medan igen vi ett half-duplex gränssnitt måste data skickas i tur och ordning. Det betyder att den mottagande delen måste vänta på att den sändande delen har sänt informationen före data kan skickas på nytt. Vid ett simplex gränssnitt behövs det bara en kabel där information skickas eftersom data endast skickas åt ena hållet, från sändaren (TX) till mottagaren (RX). (Sparkfun, 2012).

Hastigheten för ett seriegränssnitt kallas baud rate, vars enhet är bit per sekund (bps). Hastigheten för ett seriegränssnitt kan variera så länge både den mottagande och sändande delen använder sig av samma hastighet. Vanligtvis används hastigheterna 1200, 2400, 4800, 9600, 19200, 38400, 57600 och 115200 bps, där 9600 bps är den vanligaste hastigheten som används. Det är rekommenderat att högst använda sig av 115200 bps eftersom högre hastighet än det ökar risken för störningar i sändningen. (Sparkfun, 2012).

UART (Universal Asynchronous Receiver and Transmitter) kan beskrivas som en blandning av både ett parallell- och ett seriegränssnitt. På ena sidan om UART gränssnittet finns vanligtvis en 8-bit data bus (parallell) och på andra sidan finns det ett asynkront seriegränssnitt. Eftersom UART är ett asynkront gränssnitt behövs endast två kablar, en för att skicka data och en annan för att ta emot data. UART är också det mest använda seriegränssnitt. I de flesta Arduino modellernas mikrokontroller finns det inbyggt en eller flera UART gränssnitt som kännetecknas av TX och RX märkta utgångar. (Sparkfun, 2012) (Karthik, 2016).

SPI (Serial Peripheral Interface) är ett synkront seriegränssnitt som utvecklades av Motorola. SPI kännetecknas av märkningarna, SCLK, MOSI, MISO och SS. SCLK är klocksignal eftersom gränssnittet är synkront. MISO och MOSI används för att skicka och ta emot data. Ifall det finns flera komponenter kopplade behöver mottagaren känna igen vilken som är vilken. Till det används SS utgången, som står för Slave Select. Fördelen med SPI över I^2C är att data levereras snabbare vid SPI än vid ett I^2C gränssnitt. (Sparkfun, 2012) (Karthik, 2016).

Bland de mest vanliga seriegränssnitten som används med Arduino för att skicka och sända data är I^2C (Inter-Integrated Circuit). I^2C är ett synkront seriegränssnitt. Gränssnittet utvecklades redan år 1982 av Philips men tillskillnad från idag var det en mer begränsat och långsammare. Ursprungligen används I^2C gränssnittet av IC kretsar för kommunikation på moderkort till datorer. I I^2C har man tagit fördelarna från ett asynkront gränssnitt, vilket innebär färre kablar och fördelarna med SPI, det vill säga synkroniserad kommunikation. Data som skickas går via SDA märkta pins och den synkroniserade klocksignalen går via SCL. Endast de här två portarna, SDA och SPI behövs, vilket också kännetecknar ett I^2C gränssnitt. Det är möjligt att sammankoppla upp till 127 olika enheter på samma bus. Nackdelen bland annat med I^2C är att den är gjord för kommunikation med korta avstånd. (Sparkfun, 2012) (Karthik, 2016).

Tabell 1. En översikt över de olika seriegränssnitten. (Karthik, 2016)

Standard	Tx Type	# Signal Wires	Data Rate & Distance	Hardware $	Scalability	Application Example
UART	Asynchronous	2	20kbps @ 15m	Medium (transceiver)	Low (point-to-point)	Diagnostic display
LIN	Asynchronous	2	20kbps @ 40m	Medium (transceiver)	High (identifier)	Washing machine subsystem network
SPI	Synchronous	4+	25Mbps @ 0.1m	Low	Medium (chip selects)	High speed chip to chip link
I2C	Synchronous	2	1Mbps @ 0.5m	Low (resistors)	High (identifier)	System sensor network

I tabell 1 ser man grundläggande information om de olika seriegränssnitten som har behandlats. Där kan man se bland annat vilken typ, hur många kablar det behövs, samt hastighet och räckvidd.

4 Arduino

Arduino kan beskrivas som ett prototypkretskort där man har monterat en programmerbar mikrokontroll. I det här kapitlet behandlas det mera ingående vad Arduino är och vad det innebär. Utöver det behandlas också de olika modellerna som finns på marknaden. De modeller som jag övervägde till mitt arbete valde jag att gå djupare in på. På Arduinos hemsida finns det bland annat mera information, guider, nätbutik och exempel.

Ibland kan man också stöta på namnet Genuino, vilket är ett annat namn för Arduino. Skillnaden mellan Arduino och Genuino är marknadsföring. Utanför USA marknadsför Arduino som Genuino. Även om det är vanligare att man stöter på namnet Arduino vid nätbutiker i Europa. (Arduino, 2016k).

4.1 Framgång

Arduino är ett familjenamn för både olika modeller av mikrokontrollerkort men också för mjukvaran som används för att programmera dessa kretskort. Arduino innebär också en typ av certifiering. Det betyder att andra tillverkar också kan tillverka mikrokontrollerkort och få de certifierade, vilket innebär att de är kompatibla med samma hård- och mjukvara som de övriga Arduino märkta modellerna. Idag används Arduino inom många olika projekt, allt från enkla till avancerade vetenskapliga projekt. Dessa projekt är skapade av elever, programmerare men även av människor som har det som hobby. Dessutom används det också inom utbildning, med andra ord är det väldigt utbrett. Men vad är det som har gjort att Arduino har blivit så populärt? (Arduino, 2016l).

Följande saker kan man räkna som de mest bidragande orsakerna till Arduinos växande popularitet:

- Öppen källkod
- Pris
- Lätt och tydligt
- Utbud av hårdvara
- Stöd för olika operativ system

Öppen källkod betyder att vem som helst har möjlighet att både använda, modifiera och vidaredistribuera koden. I det här fallet betyder det att det är möjligt för vem som helst att expandera programmeringsspråket som används, samt även skapa sitt egna kort utgående från ritningar, som erbjuds på Arduinos hemsida. (Arduino, 2016l).

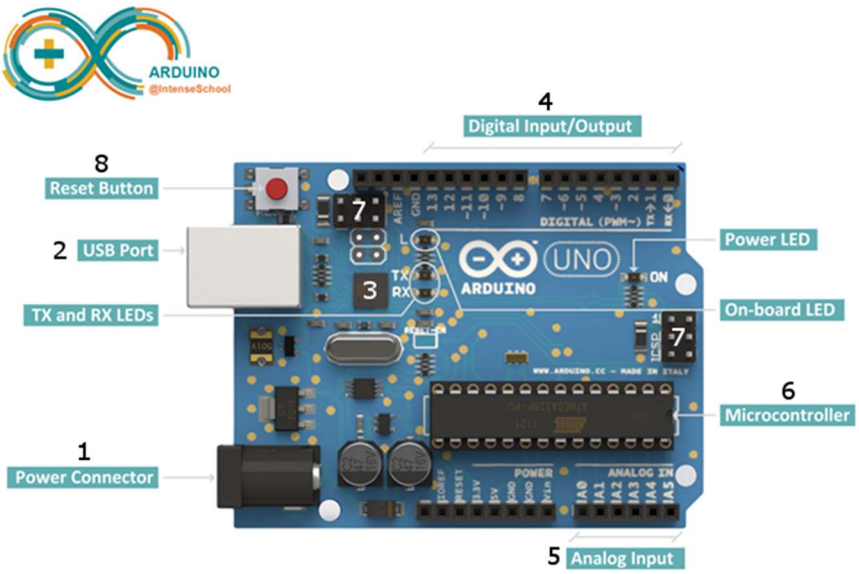
Det relativa låga priset för de olika modellerna är en bidragande orsak. Priset varierar från 25$ för de mindre modellerna och upptill 50$ för de mer avancerade korten. En annan bidragande orsak är också stödet för ett stort utbud av hårdvara. Då korten har stöd för både

analoga ingångar och digitala in- och utgångar, samt även stöd för olika kommunikations protokoll som exempel I^2C och SPI, vilket ger möjlighet till att motta värden från sensorer, styra motorer och så vidare. Arduino IDE är programmet som kan användas för att skriva och ladda upp kod med. I och med att programmet har stöd för både Windows, OSX samt Linux är användare inte tvungen att låsa sig till ett visst operativsystem för att kunna använda sig av Arduino IDE. (Arduino, 2016l).

Den kanske tyngsta orsaken till framgången är hur lätt det är att skapa ett eget projekt. Arduino använder sig att ett eget programmeringsspråk som är baserat på C-språk. Skillnaden är att en stor del har förenklats och gjorts färdigt för användaren. Det betyder att många funktioner som kan användas redan är färdigt skapade och användare behöver inte själva skriva funktionerna. Det i sin tur gör att koden blir kortare och bättre strukturerade, vilket i sin tur bidrar till att den blir lättare att tolka och lära sig av någon annans kod. (Arduino, 2016l).

4.2 Uppbyggnad i allmänhet

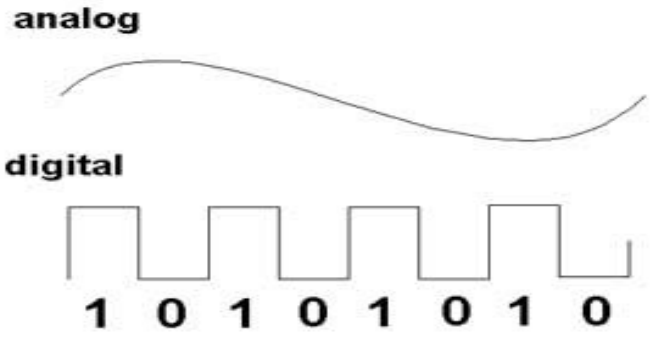
I det här kapitlet används en Arduino UNO som exempel på hur ett kort överlag är uppbyggt. Det förekommer visserligen variationer mellan de olika modellerna, men alla kort följer överlag samma grundprincip. I kapitlet förklarar jag också olika begrepp och termer som används i sammanband med kapitlet.



Figur 3. Hur ett Arduino kort kan se ut. (Furqan, 2016)

I figur 3 ser vi en Arduino UNO där olika delar är märkta med siffror för att det skall vara lättare till att hänvisa till de olika delarna. Del märkt med 1 är ett strömuttag som kan användas för att strömförsörja kortet och bredvid den finns en spänningsregulator. Ett strömuttag finns inte på alla modeller. Ifall modellen har ett USB uttag, märkt med 2, kan man med största sannolikhet även strömförsörja modellen via det. USB uttaget används också för att sammankoppla kortet med en dator för att kunna ladda upp kod. I samband med USB uttaget hittar vi också en IC krets som omvandlar USB till UART kommunikation, märkt med 3. Om det inte finns en USB port på modellen används vanligtvis UART protokoll för att sända och ta emot data och iså fall behövs en extern USB omvandlar för att kunna sammankoppla kortet med en dator. (Sparkfun, 2013d).

Olika modeller är försedda med varierande antal in- och utgångar, både analoga och digitala, beroende på vad för modell av mikrokontroller kortet är försett med. Vanligtvis använder man också ordet Pins eller I/O för att beskriva en in- eller utgång. I Figur 3 är in- och utgångarna märkta med 4 och 5. De som är märkta med 4 är digitala och de med 5 är de analoga in- och utgångarna. (Sparkfun, 2013d).



Figur 4. Skillnad mellan en analog och en digital signal. (Sparkfun, 2013a)

Skillnaden mellan en digital och en analog signal kan man se i figur 4, hur de skiljer sig åt. För en analog signal är kurvan mjuk, som en våg och för en digital är den kantig och exakt. De som man kan se är att en analog signal kan anta varierande värden inom begränsat område, till skillnad från en digital signal varierar värdet vanligtvis mellan två olika värden. En digital signal kan givetvis också variera mellan flera olika nivåer eller värden. Man kan tänka en digital signal som en av/på avbrytare, 1/0 eller LOW/HIGH. I praktiken för Arduino betyder det att om en analog ingång används kan man avläsa ett varierande spänningsvärde och sedan skapa olika funktioner beroende på värdets storlek medan vid en digital ingång är värdet 0 eller 5V. (Sparkfun, 2013a).

Ett praktiskt exempel är om man använder sig av en ultraljudsavståndmätare och en närvarosensor. Avståndsmätare är en analog givare medan närvarosensorn är digital. Närvaro sensorn skickar en signal som är LOW eller HIGH, det vill säga 0V eller så 5V ifall det finns något föremål inom området. Den analoga avståndsmätare skickar i sin tur en varierande spänningssignal beroende på hur nära objektet är och beroende på hur hög spänningen är kan man kan tolka avståndet.

Nummer 6 är kortets huvud IC (Integrated Circuit) krets, även kallat mikrokontroller. Mikrokontroller kan beskrivas som själv kortets hjärna eftersom Arduino kretskortet egentligen är en prototyp monteringsbräde för mikrokontroller. De flesta Arduino kort är försedda med mikrokontroller av företaget Atmel och deras megaAVR series IC. (Arduino, 2016l).

De flesta modeller har också försätts med en eller flera ICSP uttag. ICSP är en förkortning av In-Circuit Serial Programmer. Uttaget används för att det skall vara möjligt att byta firmware i IC kretsen efter att den har fästs på kretskortet. ICSP använder sig av SPI kommunikation, var av pinsen är märkta med MISO MOSI SCK. ICSP är märkt med nummer 7 i figur 3. (Sparkfun, 2013d).

Utöver de mest väsentliga delarna på kortet har också det flesta kort en resetknapp, märkt med 8, som kan översättas till en återställningsknapp även om det engelska ordet är mer förekommande. Som namnet antyder så återställs programmet i mikrokontroller varefter det körs från början igen. Arduino har inte stöd för en så kallad Hard Reset knapp, vilket betyder att även programmet raderas från minnet. (Sparkfun, 2013d).

4.3 Arduino UNO

Arduino UNO är den mest allmänna modellen som används speciellt för personer som vill lära sig att jobba med Arduino, därför kan UNO beskrivas som en introduktionsmodell. Namnet UNO kommer från italienska och översätts till ett. Namnet valdes eftersom UNO var det första kortet med en USB port och i samband med lanseringen, lanserades även Arduino IDE, vilket gjorde att det blev betydligt lättare att skriva och ladda upp kod till mikrokontrollen. (Arduino, 2016f).

Tabell 2. Grundläggande information om Arduino UNO. (Arduino, 2016f)

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

I tabell 2 kan vi se att UNO har försätts med totalt 14 digitala in- och utgångar, samt sex analoga ingångar. Sex av de 14 digitala in- och utgångarna kan även användas som PWM utgångar. De analoga ingångarna är märkta med A0 till A5 och de digitala in- och utgångarna är märkta från 0-13. De in- och utgångar som även är märkta med ~ har stöd för PWM signal. Pins 0 och 1 som även är märkta med RX och TX, har stöd för seriekommunikation för att skicka och ta emot data. UNO har en inbyggd ATmega16U2 krets, som fungerar som en UART till USB omvandlare. UNO har även stöd för både I^2C och SPI kommunikation. (Arduino, 2016f).

Kortet har också en 5V och en 3,3V utgång som kan användas för att strömförsörja komponenter. Ifall strömförbrukning överstiger 500mA är det rekommenderat att istället använda sig av en extern strömkälla för att undvika skada. UNO har även en inbyggd säkring som utlöses om strömmen överstiger 500mA. Utöver det finner vi även ett USB uttag, strömuttag, 16 MHz kvarts kristall, ISCP uttag och en återställningsknapp, även kallat reset. (Arduino, 2016f).

Kortet kan strömförsörjas direkt via USB porten eller via strömuttaget. Den rekommenderade spänningen är mellan 7-12V ifall strömuttaget används. Spänningen måste vara minst 7V, om ej, finns det risk att spänningen som matas till de digitala utgångarna sjunker under 5V. Den rekommenderade strömförbrukningen per I/O pin är 20mA ifall strömstyrkan överstiger 50mA finns det risk att IC kretsen skadas. (Arduino, 2016f).

Arduino UNO är försedd med Atmel ATmega328P, en 8-bit mikrokontroller. En stor del av UNO specifikationer baseras på mikrokontrollens specifikationer. Av den här orsaken kan man kalla mikrokontroller kortets hjärna. Läser man databladet för ATmega328P kan man hittar mera ingående information om mikrokontrollen. Bland annat rekommenderar tillverkaren en driftspänning mellan 1,8-5,5V för kontrollen. De utlovar också att det är möjligt att läsa och skriva till flash minnet 10000 gånger och att ATmega328P klarar av att spara data i 100 år om rumstemperaturen är normal. (Arduino, 2016f) (Atmel, 2015a).

Arduino UNO kan sammanfattas som en introduktionsmodell, eftersom den erbjuder tillräckligt med in- och utgångar till de flesta projekt och till ett lägre pris.

4.4 Mega2560

Arduino Mega2560 kan ses som UNOs storebror eftersom kortet har många likheter med UNO men också erbjuder flera in- och utgångar, vilket ger möjlighet till mera avancerade projekt. Mega2560 kan bland annat användas till 3D skrivare eller robotprojekt där det krävs mer än sex stycken servon. Modellnamnet Mega2560 kommer från mikrokontrollens modellnamn, ATmega2560, som kortet har försetts med. Arduino Mega2560 är också en uppdaterad version av den äldre Mega modellen, som var försedd med ATmega1280 mikrokontroller. (Arduino, 2016d).

Tabell 3. Grundläggande information om Arduino Mega2560. (Arduino, 2016d)

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Jämför man tabell 1 för UNO och tabell 2 för Mega2560 kan man se likheter och framförallt att det skiljer sig åt hur många analoga och digitala in- och utgångar det finns på Mega2560. Mega2560 har försetts med totalt 54 digitala I/O, var av 15 har PWM stöd jämfört med UNO som har totalt 14 digitala I/O och endast 6 PWM utgångar. Antalet analoga ingångar har också ökat från sex till 16 på Mega2560 från UNO. Till skillnad från UNO har även flash minnet för Mega2560 ökat från 32KB till 256KB, vilket betyder att det finns mera utrymme för ett längre program. Mega2560 har också totalt fyra portar som är märkta med RX och TX, vilket betyder att de kan användas för seriekommunikation. (Arduino, 2016d).

Angående strömförsörjning av modellen och den rekommenderade driftspänningen följer Mega2560 samma grund som UNO. Det vill säga att modellen kan strömförsörjas antingen via USB porten eller via strömuttaget och med den rekommenderade spänningen mellan 7-12V. (Arduino, 2016d).

Mega2560 är försedd med Atmels ATmega2560 8-bit mikrokontroller. Specifikationerna för Arduino Mega2560, så som antal utgångar och minne baserar sig på samma specifikationer som för ATmega2560 mikrokontrollen. Utöver den mest nödvändiga information för mikrokontrollen finner man också i databladet bland annat att det är möjligt att läsa och skriva till flashminnet 10000 gånger och att ATmega2560 klarar av att lagra programmet i 100 år vid normal rumstemperatur. (Atmel, 2015b).

Arduino Mega2560 kan sammanfattas som en större modell av UNO då den erbjuder mera in- och utgångar, samt även större minnesmängd, vilket betyder att den är lämplig för mera ingående projekt där det behövs många in- och utgångar och större minnesmängd för mera avancerade program.

4.5 Zero

Bland de nyaste modellerna i Arduinos utbud är Zero, som släpptes i juni 2015. Zero kan beskrivas som en 32-bits version av UNO, samt en vidareutvecklad modell för att bland annat passa in i projekt som kretsar runt Internet of Things trenden. (Arduino, 2016h).

Tabell 4. Grundläggande information om Arduino Zero. (Arduino, 2016h)

Microcontroller	ATSAMD21G18, 32-Bit ARM Cortex M0+
Operating Voltage	3.3V
Digital I/O Pins	20
PWM Pins	All but pins 2 and 7
UART	2 (Native and Programming)
Analog Input Pins	6, 12-bit ADC channels
Analog Output Pins	1, 10-bit DAC
External Interrupts	All pins except pin 4
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	None. See documentation
Clock Speed	48 MHz

Om vi jämför Zero specifikationer i tabell 4, som med Mega2560 mot UNO kan man se att de skiljer sig från varandra. Den största skillnaden för Zero och de övriga modellerna är att spänningen som matas till utgångarna endast är 3,3V, till skillnad från 5V som används på bland annat UNO och Mega2560. Det betyder att man bör kontrollerna att de givare, sensorer och övrig hårdvara man har tänkt använda sig av fungerar vid 3,3V. (Arduino, 2016h).

Zero är försedd med totalt 20 digitala in- och utgångar var 18 stycken har stöd för PWM signal. Vilket betyder att Zero även har tre stycken flera PWM utgångar än Mega2650. Precis som på UNO är Zero också försedd med sex stycken analoga ingångar men till skillnad från UNO och Mega2560 har Zero en analog utgång. Zero har försetts med lika mycket minne som Mega2560, det vill säga 256 KB. (Arduino, 2016h).

Zero har också försetts med en 32-bit ARM mikrokontroller, tillskillnad från UNO som har en 8-bit mikrokontroller. Det har lett till att Zero är fyra gånger snabbare än UNO. I och med att Zero är en 32-bit plattform betyder det att den också är lämplig till att lära sig mer om hur 32-bit applikationer fungerar. Mikrokontrollen på kortet är Atmel ATSAMD21G18. I tillverkarens datablad för mikrokontrollen finns det mer ingående information. Det som kan lyftas ur databladet fram är att ATSAMD21G18 har inbyggt en 32-bit RTC (Real Time Counter), vilket innebär en klocka och kalender funktion, som kan används på lämpligt sätt i projekt. (Atmel, 2016).

4.6 Övriga

Arduino erbjuder också betydligt flera intressanta modeller än de som jag valde att gå djupare in på. De övriga modellerna som finns på marknaden idag är; Arduino 101, Arduino PRO, Arduino DUE, Arduino YUN, Arduino GEMMA, Arduino Lily PAD. Utöver de nuvarande modellerna finns det också en stor mängd utgångna modeller.

Både Arduino PRO och Arduino 101 klassas som introduktions modeller på Arduinos hemsida. Arduino PRO påminner mycket om UNO eftersom man kan sen den som en avskalad UNO modell där tanken är att man skall permanent montera PRO modellen i något projekt. PRO finns att få i både 5V och 3,3V version, var av den sistnämnda är tänkt att strömförsörjas med hjälp av batterier. Precis som PRO påminner 101 också om UNO men till skillnad från UNO är 101 försedd med 6-axlat accelerometer och gyroskop, samt Bluetooth modul. Mikrokontrollen på 101 är inte en Atmel modell utan istället Intels Curie. Eftersom 101 är försedd med både Bluetooth och gyroskop är den lämplig att användas till olika projekt som baseras på styrning och självgående fordon. (Arduino, 2016a) (Arduino, 2016e).

Arduino Due kan beskrivas som en kombination av både Arduino Mega2560 och Zero. Fysiskt påminner Due om Mega2560 dessutom har de både modellerna 54 digitala in- och utgångar. Due har fyra stycken färre analoga ingångar än Mega2560 men istället är Due försedd med två analoga utgångar. Flash minnet på Due har också fördubblats till 512 KB. Precis som Zero är Due också försedd med en 32-bit mikrokontroller. Det som bör observeras är att den högsta tillåtna spänning till in- och utgångarna är 3,3V. (Arduino, 2016b).

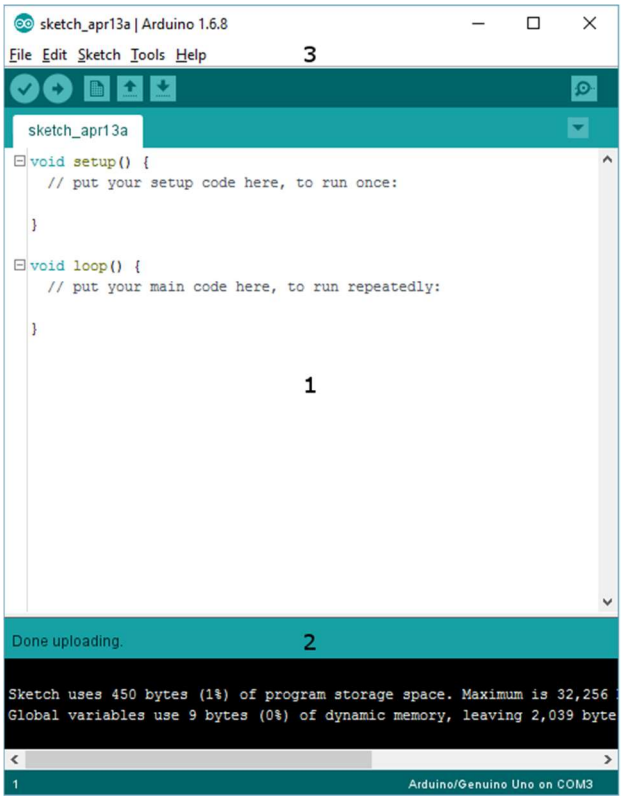
Arduino Yún är mycket lämplig till projekt som baseras sig på kommunikation över nätet, det vill säga Internet of Things baserade projekt. Till skillnad från andra Arduino modeller är Yún försedd med både Ethernet port, WiFi modul, USB A port och en mikro SD korts hållare. Yún är också försedd med Atheros AR9331 processor vilket gör det möjligt att kommunicera med Linux och skriva kod baserat på python. (Arduino, 2016g).

Arduino GEMMA och Lily PAD är bland de minsta modellerna eftersom de är tänkt för att användas i textiler och för att bäras med sig på kroppen. Eftersom de både modellerna är fysiskt mindre finns det också begränsat med in- och utgångar. GEMMA är försedd med endast tre digitala I/O, tillskillnad från Lily PAD som har nio. Både modellernas

driftspänning är 3,3V eftersom det är tänkt att modellerna skall strömförsörjas en 3,7V LIPo ackumulator. (Arduino, 2016c) (Arduino, 2016j).

5 Arduino IDE

Arduino Integrated Development Environment (IDE) är mjukvaran som kan installeras på en dator med något av följande operativsystem; Windows, Linux eller Mac OS. Programmet har skapats för att det skall vara lätt skriva kod och kommunicera med hårdvaran. I programmet finns det bland annat en inbyggd text behandlar som används för att skriva och redigera kod, funktioner för att uppladda kod till kortet, samt även en inbyggd funktion som visar meddelande och felmeddelanden. Arduino IDE är inte heller ett måste för att kunna programmera sitt Arduino kort man kan också på ett traditionellt sätt programmera Arduino kortet, eller rättare sagt mikrokontrollen men Arduino IDE underlättar speciellt för någon med begränsade kunskaper.



Figur 5. Översikt över Arduino IDE. (Skärmdump från Arduino IDE)

Det första man möts av då man öppnar Arduino IDE är den inbyggda textbehandlaren (1), som man kan se i figur 5. Den kod som skrivs här är också den kod som kommer att laddas

upp till mikrokontrollen. Programmerings språk som används i Arduino IDE är ett eget utvecklat språk som baserar sig på både C/C++. Skillnaden mellan att traditionellt programmera med C till en mikrokontroll och med Arduinos egna språk är att före koden kompileras med en C/C++ kompilator så undergår koden en behandling och den konvertering som behövs för att kunna styra och kontrollera mikrokontrollen. (Arduino, 2016i).

Det här syns i exemplet nedan. I och med att funktioner genereras automatiskt av programmet har det lett till att språket är både lätt att skriva och förstå då man läser av kod eftersom koden skrivs på ett beskrivande sätt. Följande exempel visar skillnaden hur det ser ut då man skriver till digital pin 13 i Arduino IDE och motsvande i C. Med följande kommando kan vi tända en LED som är kopplad till digitala utgången; digitalWrite(13, HIGH), medan om vi skulle vilja tända samma LED i C skulle det se ut på följande sätt; PORTB |= (1<<PB5). Som man kan se ur exemplet kan man direkt läsa ur koden vad kommandot gör i Arduinos fall.

Under om textbehandlare finns en ruta där meddelande visas (2), i figur 5. I rutan syns både felmeddelanden och meddelanden om något har lyckats. Vilket är till stor nytta ifall det skulle inträffa problem. De vanligaste meddelanden som syns är om det finns något fel i koden, som gör att det inte går att kompilera koden eller variabler som inte är definierade rätt. I rutan syns om kompileringen av koden har lyckats. Ifall något fel har inträffat försöker programmet visa och förklara så noggrant som möjligt var felet finns. Om programmet inte får kontakt med kortet över serie porten syns det också ett felmeddelande.

Över om textbehandlare finns det fem menyer (3). I menyerna finns det framför allt följande funktioner; uppladdning, val av kort modell och serie port, serie monitor och möjlighet att bränna en. starthanterare. För att kunna ladda upp koden till kortet måste man först välja rätt Arduino modell från menyn och sedan välja över vilken serieport man vill skicka data. Efter att koden har kompilerats är det möjligt att ladda upp koden till kortet.

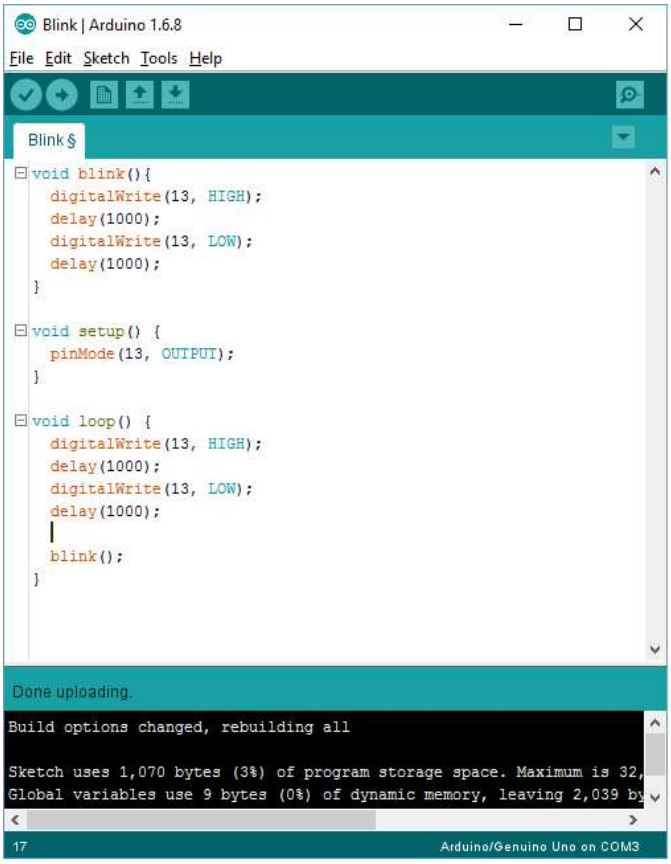
Inbyggt i Arduino IDE finns också en seriemonitor. Som namnet antyder kan seriemonitorn användas till för att övervaka den data som skickas via serieporten, det vill säga USB porten som används för att sammankoppla Arduino kortet. Seriemonitorn kan också användas till att visa värden som sensorer skickar eller för att se om variabler antar rätt värden. Utöver det kan också seriemonitorn användas för att kontrollera att koden genereras genom att lägga in funktioner som skickar text till seriemonitorn.

Inbyggt i programmet finns också bibliotek och färdiga exempel. Bibliotek används för att lägga till extra funktion i programmet utan att man själv behöver skriva dessa. I Arduino IDE finns det bland annat färdiga bibliotek för servo motorer, SD kort, I^2C och SPI. Det är också möjligt att skapa ett eget bibliotek eller modifiera något existerande. Utöver bibliotek finns det också färdiga exempel med kod för olika saker. Exemplen varierar allt från att få en LED att blinka till hur man skapar en webbserver.

På en del Arduino modeller är det också möjligt att avlägsna mikrokontrollen och ersätta den ifall skada har skett. Ifall man väljer att köpa en ny mikrokontroll är dessa inte alltid färdigt programmerade för att kunna tolka Arduino språk och därför är man tvungen att bränna en s.k. starthanterare, på engelska bootloader, vilket Arduino IDE har stöd för att kunna göra. Starthanteraren kan i sin enkelhet förklaras som en form av en koordinator då mikrokontrollen startar upp. Den granskar ifall du försöker ladda upp kod eller om du kör kod. (Sparkfun, 2013c).

5.1 Exempel

I figur 6 kan man se två funktioner, void setup(); och void loop();. De är endast de här två funktionerna som behövs för att få något att fungera, om det sedan är en LED eller en sensor som skickar ett värde. Utöver de nödvändiga funktionerna tillkommer det givetvis variabler, konstanter och import av olika bibliotek.



Figur 6. Exempel kod för att få en LED att blinka. (Skärmdump från Arduino IDE)

I void setup(); funktionen placerar man vanligtvis de kommandon som man vill köra endast en gång eftersom dessa kommandon läses av endast då Arduino modellen startas upp. För att köra kommandon i void setup(); på nytt måste man återställa Arduino kortet. Kod som placeras i void setup(); är vanligtvis definiering av vilken hårdvara som använder vilken I/O, uppstart av sensorer och annan hårdvara. I figur 6 ser vi exempel på hur det kan se ut för att få en LED att blinka med ett visst intervall. Ur figuren kan man se att man har definierat i void setup(); vilken utgång LED är kopplad till och om det är en ingång eller utgång.

Huvudprogrammet placeras i void loop(); efter som den kod har placerats där körs om och om igen. Koden läses av rad för rad i tur och ordning. I figur 6 i void loop(); funktionen kan vi se att vi först förser den digitala utgången 13 med 5V, vilket leder till LED tänds. Efter det står programmet stilla i en sekund eftersom vi har placerat en fördröjning (delay), vars enhet är millisekunder. Efter fördröjningen slocknar LED för att sedan tändas igen efter en sekund och på följande sätt körs programmet om och om igen.

I void loop(); ser vi också en annan funktion void blink();, som även är definierad över om void setup();. Som man kan se ur figur 6 innehåller void blink(); samma kod rader som det finns i void loop();. Kör vi programmet med endast void blink(); i void loop(); kommer även

LED att börja blinka på samma sätt. Då programmet blir längre är det lättare att endast kalla på olika funktioner i void loop(); och inte skriva alla kod där. Det leder till att programmet blir mer överskådligt.

6 Sensorer

Då man vill utöka ett projekt eller automatisera något görs det vanligtvis via att man lägger till en eller flera sensorer. Sensorerna kan användas på olika sätt beroende på vilket typ av projektet.

Går man in på en nätsida som säljer olika typer av sensorer och givare, som referens använder jag mig av sparkfun.com. Då kan man snabbt konstatera att det finns sensorer som kan mäta det mesta. Olika typer av sensorer behövs för olika ändamål, beroende på typ av projekt. I ett projekt kanske man enbart vill avläsa värdet för att sedan spara värdet i någon form av databas, som en väderstation till exempel. I ett annat projekt kan man använda sig av sensorer för att automatisera projektet, som en miniatyrbil. Där sensorer används för att undvika kollision med hinder, som exempel.

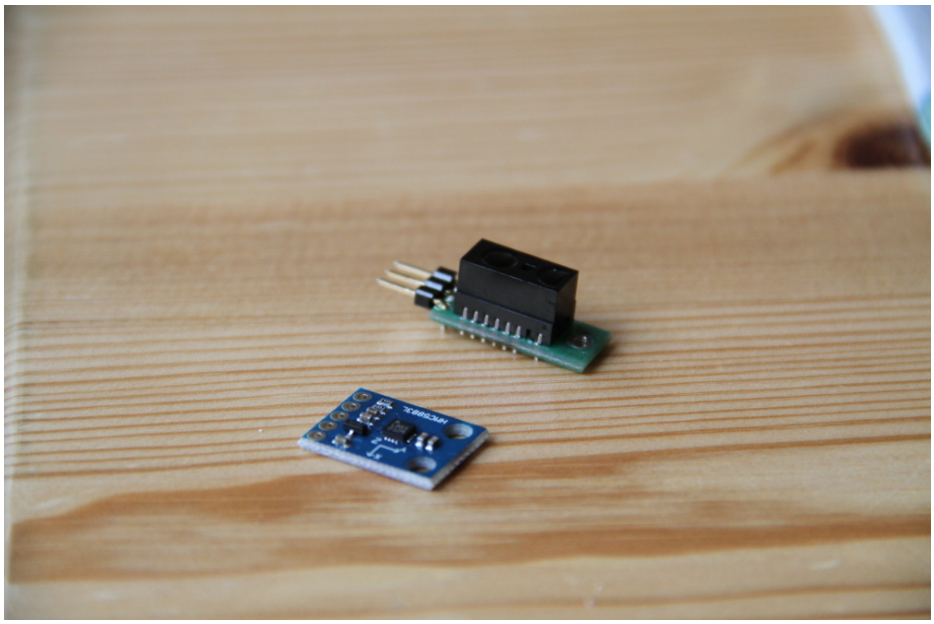
I listan nedan kan man se exempel på olika sensorer, både vanliga och mer ovanliga.

- TMP006 temperatur sensor
- BMP180 lufttryck sensor
- ADXL345 3-axlad accelerometer
- HMC5883L kompass
- GP2Y0D810 IR närvarosensor
- HC-SR04 ultraljuds avståndsmätare
- AD8234 hjärtfrekvens sensor
- GT-511C1R fingertrycks avläsare

Då man väljer en sensor till projektet bör man vanligtvis granska de två följande sakerna, kommunikationsgränssnitt och driftspänning. I och med att det finns ett stort utbud av sensorer, finns det också en del variationer av dessa. Då Arduino använder sig av 5V eller 3,3V är de flesta sensorer anpassade till det området. Bland annat närvarosensorn GP2Y0D810 fungerar med en likspänning som ligger mellan 2,7-6,2V medan igen ultraljuds

avståndsmätare, HC-SR04 enbart fungerar vid 5V, tillskillnad från kompassen HMC5883L där max spänningen är 3,6V. Det betyder att man bör noggrant kontrollerar specifikation för sensorn före man tar den i bruk. (Sparkfun, 2016).

I kapitel tre behandlas seriekommunikation i syfte om att man skall ha en bättre förståelse om tekniken då man väljer sensor, eftersom olika sensorer använder sig av olika seriegränssnitt. Alla sensorer använder sig inte heller av seriegränssnitt, till exempel närvarosensorn som användes vid robotarmen skickar endast ut en digitalsignal, var av den kopplas till en digital ingång på Arduino kortet. Ser man på de sensorer som sparkfun.com erbjuder kan man konstatera att de flesta sensorer använder sig antigen av I^2C eller SPI gränssnitt. Utan att se i tillverkarens beskrivning av sensorer kan man vanligtvis avgöra om den använder sig av I^2C eller SPI. Eftersom I^2C kännetecknas av SDA och SCL märkta pins medan SPI kännetecknas av MISO MOSI märkta pins. I figur 7 nedanför ser man två sensorer, en digital närvarosensor och under om den ett HMC5883L kompass med ett I^2C gränssnitt.



Figur 7. En digital närvarosensor och en kompass, HMC5883L. (Egen bild)

Hur koden utformas i Arduino IDE för att få sensor att fungera beror till stor del på vad för typ av sensor det är. Ifall sensorn använder sig av I^2C eller SPI protokoll behöver man vanligtvis först importera ett nödvändigt bibliotek. Efter det lägger man vanligtvis det kommando som startar upp sensorn i void setup();, eftersom vi endast behöver starta sensorn engång. I void loop(); skrivs sedan den kod som behövs för att få värdet från sensorn som sedan i sin tur kan visa i serie monitor i Arduino IDE eller på en skärm.

I figur 8 ser man hur koden kan se ut för en digital IR sensor. Sensorns kod utformas på liknande sätt som för en avbrytare genom att inleda med att definiera vilken ingång sensorn är kopplad till (int sensor_Pin = 12;). I void setup(); definieras I/O som används som en ingång. Med kommandot sensor_Reading = digitalRead(sensor_Pin); läses sensor av, ifall den digitala signalen är LOW eller HIGH. I figuren har sensorn använts för att tända en LED om signalen är HIGH.

```
int sensor_Pin = 12;
int sensor_Reading = 0;
int led_Pin = 13;

void setup() {
  pinMode(sensor_Pin,INPUT);
  pinMode(led_Pin,OUTPUT);
}

void loop() {
  sensor_Reading = digitalRead(sensor_Pin);

  if (sensor_Reading == LOW){
    digitalWrite(led_Pin,HIGH);
  }
  else{
    digitalWrite(led_Pin,LOW);
  }
}
```

Figur 8. Exempelkod på en digitalsensor. (Skärmdump från Arduino IDE)

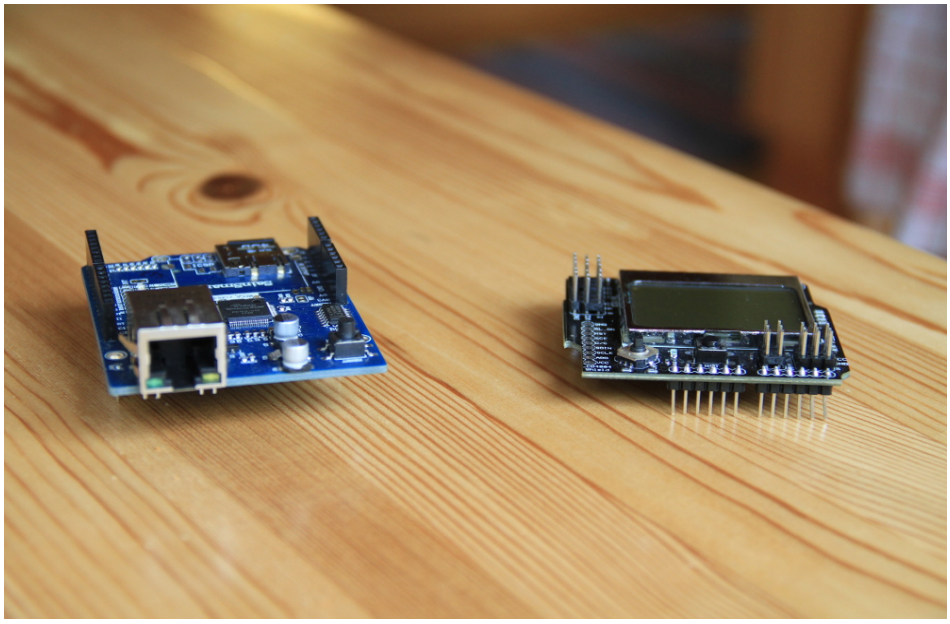
7 Shields

Det är också möjligt att lägga till flera funktioner eller expandera ett Arduino kort för att kunna till exempel styra motorer, skicka data till en webbserver, spara samma data på ett SD kort eller olika former av trådlöskommunikation. Det kan man göra genom att montera en sköld, även kallat Shields ovan på Arduino kretskortet.

Följande lista innehåller exempel på olika sköldar som finns och vilka funktioner dessa har:

- Ethernet och WiFi sköld — möjlighet till nätverks kommunikation
- Relä sköld — möjlighet till att styra en eller flera relä
- Motor/Servo Sköld — för att styra motorer eller flera servon
- LCD skärm — kan visa information eller värde från sensorer
- GPS sköld — möjliggör GPS positionering

Listan kan göras betydligt längre då de som nämns är bara ett fåtal. Enligt shieldlist.org finns det 518 stycken sköldar, Totalt finns det mera noggrannare information om 317 stycken, som till exempel vilka ut- och ingångar de använder på Arduino kortet och om det är möjligt att kombinera flera sköldar ovanpå varandra. Eftersom skölden monteras ovan på Arduino kortet är det viktigt att först och främst kontrollerar att skölden fysiskt passar eftersom det förekommer variationer bland de olika modellerna. Beroende på vilken funktion skölden har kan det vara att den också tar upp in- och utgångar, vilket betyder att det är viktigt att kontrollerar att de angivna in- och utgångarna är lediga. Ifall man planerar på att använda sig av flera sköldar ovan på varandra, vilket är möjligt, lönar det sig att kontrollerna att det är möjligt att montera en annan sköld ovanpå, vilket inte alla är. (Sparkfun, 2013b).



Figur 9. Olika sköldar för Arduino. (Egen bild)

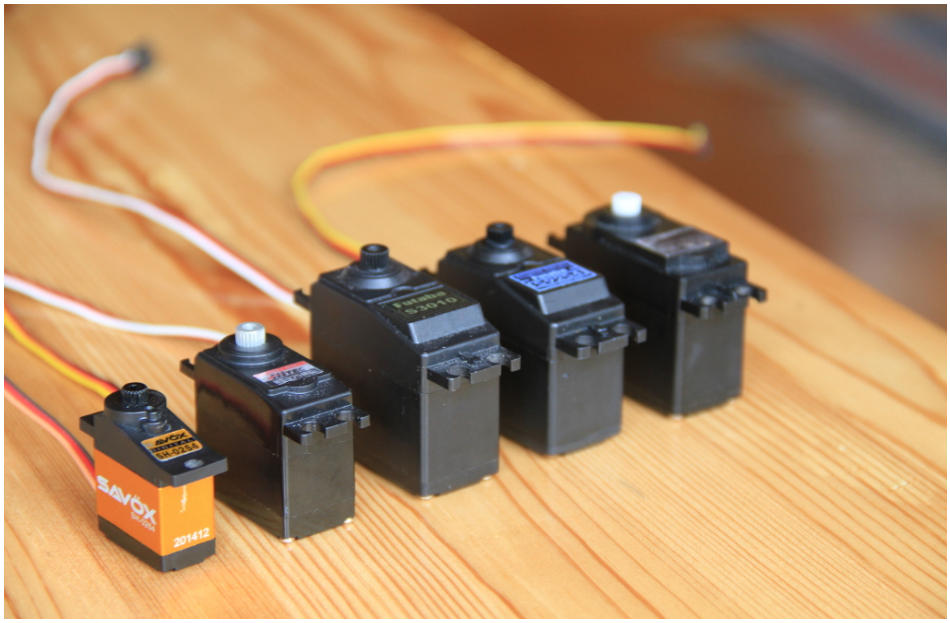
I Figur 9 kan man se två exempel på hur sköldar kan se ut. Till vänster i figuren finns en Ethernet sköld och bredvid den en LCD sköld med inbyggda styrknappar. Som man kan se ur figuren är det inte möjligt att montera en annan sköld ovanpå LCD skölden, vilket man kan göra på Ethernet skölden.

8 Servo

Ett servo kan på ett kort och lätt sätt beskrivas som ett föremål som omvandlare elektriska impulser till mekanisk rörelse med god precision. På grund av den här funktionsprincipen kan vi idag hitta servo i en stor del av produkter. Servon förekommer i allt från Cd-läsare i en dator, i fordon och i robotar av olika typer. Genom att använda sig av servon har man underlättat och minskat på den kraft som behövs för att kunna röra på något. Ett mera beskrivande exempel kan vara en lucka som är servostyrd. Utan ett servo skall en person fysiskt öppna och stänga luckan, dessutom måste personen i fråga vara fysiskt närvarande vid luckan. Men med hjälp av att servo behöver personen bara trycka på en fritt placerade knapp för att kunna öppna och stänga luckan.

Servon används mycket inom hobbyverksamhet vilket har lett till att det finns ett stort utbud av servon i olika storlekar och typer. Vilket i sin tur har lett till att det är populärt att även använda servon i olika typer av projekt, främst inom robotområdet, som baseras på till exempel Arduino. I det här arbetet valde jag att inrikta mig endast på hobby servon eftersom de användes i robotarmen

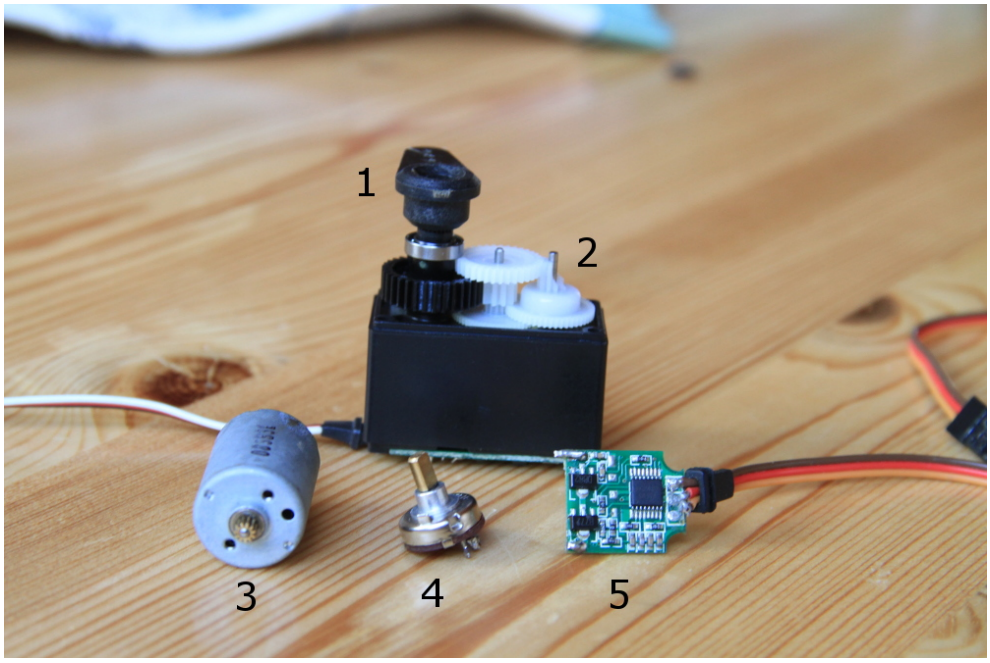
Figur 10 sammanfattar de vanligaste storlekarna och typerna av hobbyservon som det finns. Beroende på storlek indelas servona i följande storleksklasser; mikro, mini och standard. Sett från vänster i figuren kan vi se de olika storlekarna. Servon kan också delas in i tre typgrupper beroende på om de är; analoga, digitala och fritt roterande servon



Figur 10. Olika servo storlekar och typer. (Egen bild)

8.1 Uppbyggnad

Ett enkelt servo består av följande saker; en rotationsaxel, växellåda, elmotor, potentiometer och en styrkrets, som i figur 11. Alla dessa delar är i sin tur inkapslade i ett ytterskal av plast eller på de mera påkostade modellerna, aluminium. Delen som är märkt med 1 i figur 11 är en drivaxel och följande del i nummerordningen är växellådan. Del 3 är servomotorn och del nummer 4 är en potentiometer. Den sista delen märkt med 5 är styrkretsen.



Figur 11. De olika delarna i ett servo. (Egen bild)

Drivaxeln i servot är den sista länken i hela drivkedjan och i den här axeln fästes sedan en hävarm. Precis som namnet beskriver roterar axeln. I de mindre påkostade modellerna av servon förekommer vanligtvis endast glidlager som axeln roterar runt men i de mera påkostade modellerna har man ersatt glidlagren med ett eller två kullager för att minska på friktion och glapp. Drivaxeln är också försett med ett kugghjul eftersom den är förgrenad med resten av växellådan i servot. Växellådans uppgift baserar sig på den mekanikens gyllene regler; "*Det man vinner i kraft förlorar man i väg*". Ifall drivaxel skulle vara sammankopplad direkt med servomotorn utan någon utväxling i förhållande 1:1 skulle drivaxeln rotera lika många varv som motorn, vilket skulle betyda att servots styrka skulle vara lika stor som elmotorns rotations moment, det vill säga begränsat. Ifall det finns en utväxling mellan drivaxeln och motorn kommer servots styrka öka betydligt till förlust av att drivaxel roterar betydligt långsammare än vad motoraxeln gör. (Eglowstein, 2012).

Drivaxeln och kugghjulen i växellådan förekommer i olika material beroende på servots specifikation, samt pris. Det vanligaste förekommande materialet i servo av lägre prisklass är nylon. Vanligtvis används nylon kugghjul i servon med mindre kraft, vars moment är under 6 kg-cm. Då servots styrka ökar har man istället valt att använda sig av kugghjul och drivaxel i metall för att säkerställa att de klara av en större belastning. En nackdel med att använda sig av metallkugghjul gentemot nylon är ifall servot utsätts för vibrationen kommer glappet mellan kuggarna öka, vilket leder till att precisionen förblir sämre. Titan och karbonite förekommer också i servon. Titan används främst i servon som utsätts för stor belastning, i servon som är specificerade för 20 kg-cm och uppåt. Karbonite är förstärkta nylon kuggar som uppskattas till att klara av fyra gånger högre belastning än nylon och med mindre slitage. (Society of Robots, 2014).

Servon drivs vanligtvis av en DC (Direct Current) motor, som översätts till svenska som likströmsmotor. AC (Alternating Current) motorer, det vill säga växelströmsmotor används också i servon men de används främst i större servon inom industrin. Motorer av typ borst och bortslösa används i servon idag, den sist nämna har blivit allt vanligare under de senaste åren. Borstmotorer kan även indelas i två grupper, normal och coreless. I en coreless motor har man ersatt stålkärnan, där koppartråden är runt lindad med ett trådnät som istället roterar runt magneterna. Det har lett till att en coreless motor har mera moment och mjukare gång samt även högre acceleration. (Society of Robots, 2014).

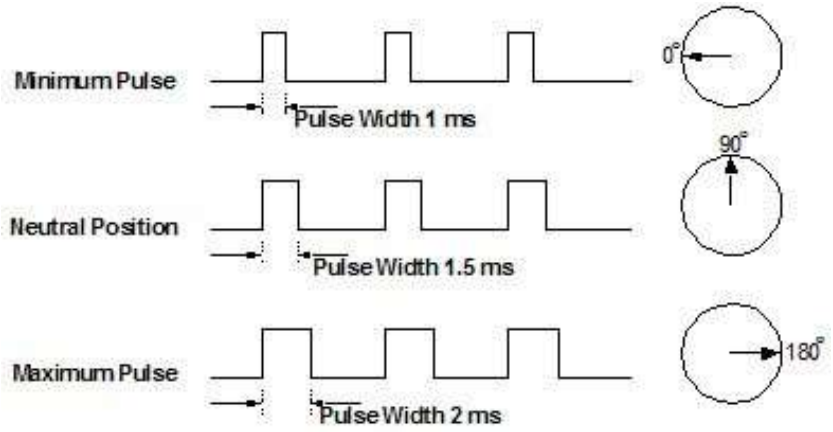
En potentiometer inom elektroniken är ett justerbart motstånd. Det mest förekommande värdet för potentiometrar som används i servon är 5K ohm. I servon används potentiometern för att beskriva i vilken position servon befinner sig som bäst efter som resistansen förändras då man vrider på potentiometern. För att den skall kunna roterar och resistansen skall kunna förändras har potentiometern direkt kontakt med växellådan. (RC Helicopter Fun, u.å.).

8.2 Funktion

För att hobbyservon skall fungera behövs tre saker, spänning, jord och en signal. Ett servo har ett spänningsbehov mellan 5-6V för att fungera optimalt och orka leverera den kraft som har angivits i specifikationerna. Så kallade High Voltage servon har blivit allt mer vanligare eftersom de klarar av spänningar upp till 8,5V. Dessa spänningsvärde används främst för att i de flesta fall drivs servon av ackumulatorer av typen nickel-metall hybrid och litium

polymer, vars nominella spänning är 4,8V eller 6V för NiMH och 7,4V för LIPo. (RC Helicopter Fun, u.å.).

Att enbart tillföra en spänning till servot gör inte så mycket, i de flesta fallen rör sig inte ens servot på sig och det går fritt att roterar axel eftersom servomotorn inte har någon spänning. För att servo skall kunna röra på sig behövs en signal av typen PWM, som är förkortat från Pulse Width Modulation. Som namnet antyder är det en pulserad signal med frekvensen 50 Hz för analoga servon och upptill 300 Hz för digitala servon. Det kan man se i figur 12, som beskriver visuellt hur en PWM signal ser ut för ett servo. (Society of Robots, 2014).



Figur 12. Beskrivning av PWM styrning. (Society of Robots, 2014)

Dessa intervaller i sin tur styr sedan servot till en viss position beroende på hur långt intervallet är. Då ett vanligt servo som roterar 180 grader befinner sig in sin mitt position, det vill säga 90 grader är intervallets längd 1500µs, som man kan se i figur 12. Genom att ändra intervallets längd till antagligen 1000µs eller 2000µs kommer servo att förflytta sig till sitt ändläge, som är 0 eller 180 grader. Ifall intervallet skulle vara 1250µs betyder det att servo skulle förflytta sig till sin 45 graders position eftersom;

$$\frac{45°}{180°} = {}^{1}/_{4} = 0{,}25 = 25\%$$

$$1000\mu s \times 1{,}25 = 1250\mu s$$

Eller på motsvarande sätt kan man ta reda på vad 165 grader motsvara i mikrosekunder för servot;

$$\frac{165°}{180°} = 0{,}916 = 91{,}6\%$$

$$1000\mu s \times 1{,}916 = 1916\mu s$$

Det kan vara bra att känna till hur man kan på ett lätt sätt omvandlar grader till mikrosekunder ifall man väljer att istället använda sig av mikrosekunder i koden, som styr ett servo.

Ett servo som är fritt roterande går inte till en viss position utan beroende på intervallets längd roterar servot med en varierande hastighet istället. Om intervallet är 1500µs står servot stilla och är intervallet 1000µs roterar servot åt ena hållet med full hastighet.

8.3 Servo i Arduino

Att styra servon med hjälp av ett Arduino kort är relativt lätt eftersom biblioteket som behövs finns redan implementerat i Arduino IDE, inget utomstående biliotek behövs nämligen. Utöver det behöver man definiera vilka utgångar man vill använda till servona. Det som behövs anmärkas är att utgångarna måste vara digitala utgångar med möjlighet till PWM signal. För att sedan kunna styra kan man använda sig av två olika kommando beroende på om man vill använda sig av enheten grader eller mikrosekunder. Skillnader förekommer även mellan de båda kommandon. Nedanför i figur 13 kan man se hur koden ser ut i Arduino miljön för att styra ett servo med två olika kommandon.

```
#include <Servo.h>

Servo futaba;

void setup(){
  futaba.attach(3,1000,2000);
  futaba.attach(3);
}

void loop(){
  futaba.writeMicroseconds(1250);
  futaba.write(45);
}
```

Figur 13. Styrning av servon i Arduino IDE. (Skärmdump från Arduino IDE)

För att kunna styra ett servo behövs först ett bibliotek som innehåller de funktioner vi behöver. Med kommandot #include <Servo.h>, importerar vi det biblioteket. Nästan steg är att definiera en variabel som ett servo. I Figur 13 kan vi se att variabel futaba har definieras som ett servo med hjälp av Servo;. Sedan behöver vi definiera vilken eller vilka utgångar som servot har anslutits till, observera att de måste vara digitala utgångar. Detta görs genom att använda sig av attach();, i figur 13 har jag valt digital utgång 3. Ifall man vill använda sig av mikrosekunder måste man även definiera ett intervall med hjälp av attach();,som vi kan se i figuren har utgång 3 valts men även att intervallets minimum och maximum värdet är 1000 och 2000.

För att slutligen sedan kunna styra servo till en position kan man använda sig av två kommandon write(); eller writeMicroseconds();. Med kommandot write(); använder man sig av grader, det vill säga man berättar till vilken position servot skall flytta sig till i grader. På motsvarande sätt om man använder sig av writeMicroseconds(); berättar man till vilket läge servo skall röra sig. I figur 13 ser man att servot som har definierat som futaba skall rör sig till sin 45 graders position eller som motsvara 1250µs.

Både write(); och writeMicroseconds(); gör samma sak, flyttar servot till en position men av egen erfarenhet har det visat sig att write(); kommandot inte fungerar med alla servon. Istället för att servo går till den definierade positionen börjar servot istället hacka och röra sig ojämnt. Med kommandot writeMicroseconds(); har dessa problem inte uppstått.

9 Robotarm

Utöver att få en djupare inblick i Arduino valde jag också att själv göra ett projekt baserat på Arduino för att bland annat kunna tillämpa den teori jag har lärt mig under arbetets gång. Valet föll på att konstruera och programmera en robotarm. Valet av en just en robotarm berodde på att jag ville göra något som skulle kunna användas i praktiskt sammanhang och att ha något som går att vidareutveckla.

Till projektets målstolpar hade jag valt följande skeden; val av servo, val av lämplig Arduino modell, automat- och manuellstyrning och någon form av sensor av säkerhets skäl eller som kan utnyttjas på annat sätt i programmet.

9.1 Planering och val av komponenter

Arbetet på robotarmen inleddes genom att göra en ritning och planering av robotarmen med hjälp av AutoCAD. Målet med själva robotkonstruktionen var att skapa en flexibel konstruktion som inte skulle begränsas av servoval eller montering av tilläggsutrustning i efterhand. Som materialval valde jag 3mm plywood eftersom det gjorde det möjligt att fräsa ut alla delar jag behövde med hjälp av en CNC fräs. För att kunna fräsa ut delarna konverterade cad ritningen manuellt till G-kod, som fräsen kunde läsa och tolka. Efter att alla delar har limmats ihop, laminerades hela konstruktion med hjälp av glasfiber och epoxi, vilket förstärkte konstruktionen. I bilaga 1 finns den ursprungliga ritningen på robotarmen.

Efter att designen av armen började ta form var nästa steg att välja lämpliga servon för ändamålet. Följande faktorer påverkade stort valet av servon; styrka, snabbhet och precision Problemet var att hitta servon som var både precisa, snabba och starka till ett lämpligt pris. Av egen erfarenhet med hobbyservon visste jag att det var svårt val att hitta lämpliga servon eftersom servon som är både starka och snabba saknar precision, om kostanden är en faktor. Servon som i sin tur är precisa och snabba saknar styrka. Valet föll på Savöx och Hitec, vars servon har visat sig vara bra eftersom de kombinera de tre faktorerna till ett mera anpassat pris. I följande lista kan man se vilka modeller jag vald, samt deras styrka och snabbhet vid 6 V:

- Savöx SC-1283SG (30 kg-cm/0.13 sek/60°) för axel 1
- Savöx SC-1256TG (20 kg-cm/0.15 sek/60°) för axel 2
- Savöx SC-0254MG (7,2 kg-cm/0.14 sek/60°) för axel 3 och gripklo
- Savöx SH-0254 (3,9 kg-cm/0.13 sek/60°) för verktygs rotation
- Hitec HS-645MG (9,6 kg-cm/0.20 sek/60°) för rotations axel

Valet av lämplig Arduino modell påverkades bland annat av att det behövdes sex stycken digitala I/O med stöd för PWM och sex stycken analoga ingångar för manuell styrning samt överlopps in- och utgångar för övrig hårdvara. Första valet var Arduino Zero främst för den var bland de nyaste modellerna och den hade tillräckligt med in- och utgångar för ändamålet. Orsaken till varför Zero inte valdes berodde främst priset och att driftspänning till in- och utgångarna var 3,3V. Som ett andra val var Arduino Mega2560 för att kunna försäkra sig om att det fanns tillräckligt med in- och utgångar för att även i senare skede kunna utveckla projektet efter behov. Valet föll sedan på Arduino UNO eftersom på UNO fanns det sex stycken PWM utgångar och sex stycken analoga ingångar samt även överlopps in- och

utgångar. Kostnaden för UNO var också lägst av vilket var en bidragande faktor till det slutgiltiga valet.

För att kunna styra robotarmen manuellt använde jag mig av sex stycken potentiometrar, var av två användes som skruvpotentiometer och resten används i form av två joystickar. Utöver en manuell styrnings funktion hade jag också valt två stycken digitala IR närvarosensorer närmare bestämt Pololu Sharp GP2Y0D810, som upptäcker föremål i området 2 till 10 cm. Tanken bakom sensorerna var att använda de som en säkerhetslösning för att kunna stanna robotarmen för att undvika kollision. Ett annat ändamål var också att använda de för att kunna starta ett visst program.

9.2 Kostnad

I följande lista kan man se ungefär vad kostnaderna var för de olika komponenterna, som användes i robotarmen:

- Servon: 240,00€
- Arduino UNO: 30,00€
- Pololu Sharpsensors: 10,00€
- Strömkälla och övrigt: 25,00€

Kostnaden för en Arduino robotarm utgörs till stor del av servon. Valet av servon beror på robotarmens storlek och hur mycket den skall klara av att lyfta, samt även hur den är mekaniskt utformad. Det är också möjligt att använda sig av två servon på samma axel för att kunna minska på servots styrka eller för att fördubbla styrkan. Det som bör observeras är att alltid halveras inte priset på servot bara för att styrkan halveras. Servots precision och snabbhet påverkar också kostnaden för ett servo. Genom att använda sig av så kallade kinesiskaservon kan man snabbt halvera totalkostnaden för servon men med risk för sämre kvalité.

Priset för Arduino UNO, som användes i projektet var 29,90€. Vill man ta steget till nästan modell för att försäkra sig om att in- och utgångarna räcker till kan Mega2560 vara ett alternativ men då stiger priset också till 44,90€. Ett alternativ är också att använda sig av en

servotilläggsmodul, även kallat Shield, till UNO för att ökat antalet servoutgångar med 16. Priset för den tilläggsmodulen är runt 15,00€.

Övrig utrustning som används, så som sensorer, brytare, potentiometers och strömkälla kan köpas på ebay av olika kinesiska försäljare till ett väldigt lågt pris. Priset för sensorer varierar vanligtvis mellan 1,00–5,00€, likaså för brytare, kontakter och kablar, som kan behövas. Kostnaden för strömkällan som användes var runt 15,00€.

Prisuppgifterna var hämtade från följande nätbutiker; Verkkokauppa, Adafruit, Ebay och Lindinger, i april 2016.

9.3 Utförande

Komponenterna kopplades enligt kopplingsschema, som finns i bilaga 2. För att strömförsörja både servon och Arduino kortet valdes en 6V/20A strömkällan. Strömkällan dimensionerades enligt hur stor den totala strömmen blir om alla servon skulle stalla. Servots stall ström finns dokumenterat i tillverkarens datablad för servot. Den totala strömmen blev ca 15A. Strömkällan överdimensionerades en aning för att även kunna strömförsörja de övriga komponenterna och för att minska på värmeutvecklingen eftersom kostnaden var minimal mellan en 15A och en 20 A strömkälla.

Alla servon strömförsörjdes direkt av strömkällan och servosignalkablarna kopplades till de digitala PWM utgångarna (3, 5, 6, 9, 10, 11) på Arduino UNO, enligt kopplingsschemat. Sharp sensorerna kopplades till Arduinos 5V utgång (5V), GND och till två digitalingångar (12, 13) för utgående signal från sensorn. Potentiometrarna som användes för manuellstyrning kopplades också till 5V, GND men även till de analoga ingångarna (A0 - A5).

Efter att alla komponenter var kopplade kunde jag inleda programmeringen av robotarmen. Grundtanken bakom hur jag utformande programmet var att skapa en botten som innehåll färdigt manuellstyrning, tolkning av signal från sensorerna och utrymme för huvudprogrammet för robotarmen. Genom att ha ett färdig botten kan jag lätt byta ut huvudprogrammet som körs automatiskt beroende på vad jag vill att armen skall göra.

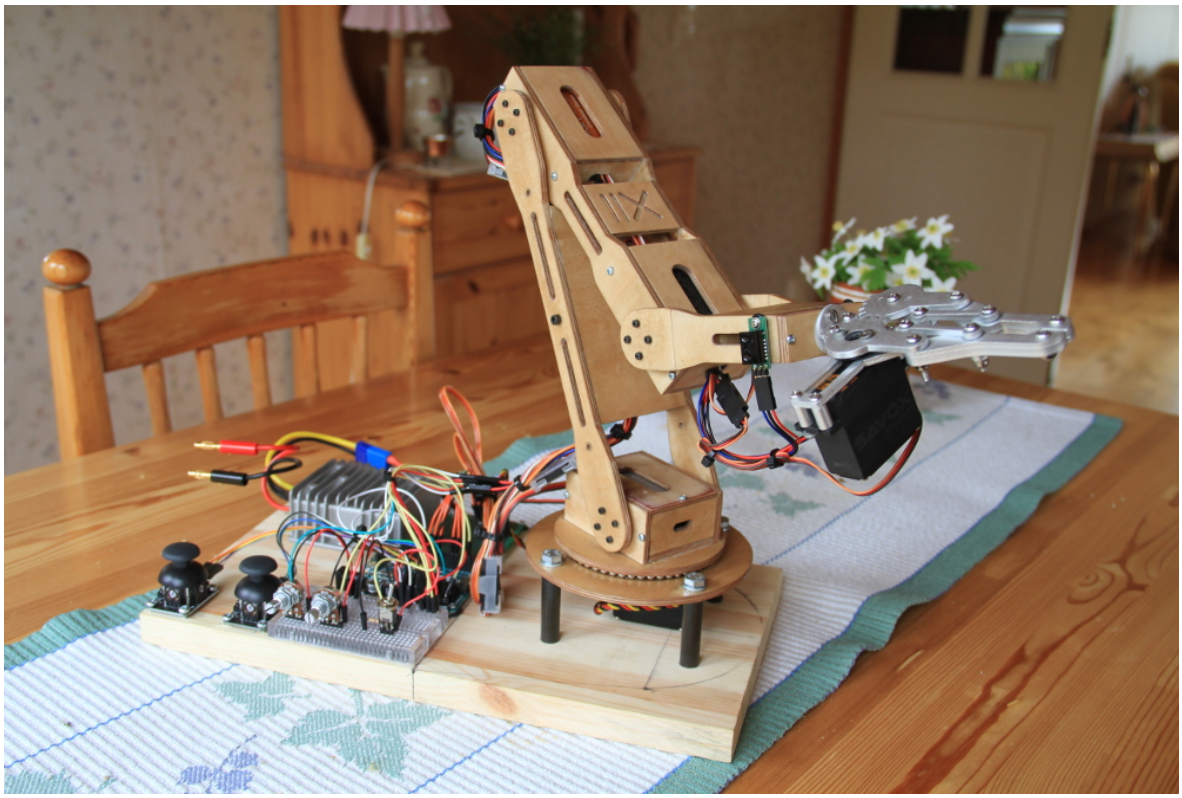
I bilaga 3 har jag inkluderat koden för robotarmen med ett automatiskt program som går ut på följande sätt: Roboten plockar upp ett föremål, som exempel användes en USB minnessticka och flyttar på den för att sedan plocka upp nästa föremål för att sedan placerar den det ena föremålet ovanpå den andra. Efter det väntar roboten fem sekunder för att sedan flytta tillbaks föremålen med ombytta platser. På så sätt kan programmet upprepas flera gånger.

Programkoden inleds med att importera servo biblioteket som behövs med hjälp av #include <Servo.h>. Efter det definieras alla variabler till INT, det vill säga heltal, inte decimal tal, som man kan se i bilaga 3. Då alla variabler var skapade kunde jag göra de funktioner som jag ville ha i grundkoden, vilket innefattade void control(); void sharpsensor(); och void rest();. Funktionen void control(); innehåller koden för manuell styrning av robotarmen. De två joystickarna som används programmerades så att då de befinner sig i deras ändpositioner rör sig respektive servo en grad var 25ms. Skruvpotentiometrarna som användes för att rotera och öppna gripklon skalades i programmet så att servot rör sig linjärt med dem. I funktionen void sharpsensor(); läses värdet av som sensorerna skickar. Värdet är LOW (0 V) eller HIGH (5V) som sedan i sin tur kan användas i programmet för olika ända mål. För att kunna köra roboten till en viloposition skapades funktion void rest(); som kan användas till att bland annat avsluta ett program eftersom robotarmen viker ihop sig och lutar mot plattformen den står på.

Huvudprogrammet som körs är placerat i void loop();. Programmet inleds genom att vi kallar på funktion void sharpsensor(); för att se vilket värde sensorerna skickar. Ifall värdet är LOW (0V), det vill säga det finns något föremål i närheten står programmet stilla tills det är fritt utrymme. Om det inte finns något föremål i närheten kollar programmet till följande ifall den manuella avbrytaren är intryckt, det vill säga spänningen till den digitala ingången (2) är 5V. Om avbrytaren är intryckt körs det automatiska programmet ända tills avbrytare släpps upp och spänningen till den digitala ingången är 0V. Då kan man manuellt kontrollera robotarmen.

Det som bör nämnas angående det automatiska programmet som är uppdelat i tre funktioner, void move_objectOne();, void move_objectTwo(); och void move_objectBack();. Är att istället för att använda sig enbart av write(); eller writeMicroseconds(); för att styra servona användes främst en så kallad for loop. Fördelen med att använda sig av en for loop är att servot rör sig saktare och mjukare eftersom man kan definiera hur många steg servot skall

röra sig på en viss tid. Om man enbart använder sig av write(); rör sig servot till den definierade positionen med full hastighet.



Figur 14. Arduino robotarm. (Egen bild)

Arbetet kan sammanfattas med figur 14. I figuren har själva robotarmen monteras på en träskiva där också de övriga komponenterna har monterats, vilket gör det lättare att byta ut och pröva nya komponenter för framtida planer.

Det här var bara ett exempel på alla de olika program och rörelse mönster robotarmen kan utföra. Det här exemplet valdes främst för att visa att robotarm kunde placera föremålen med tillräckligt hög precision för att kunna göra det om och om igen.

10 Avslutning

Då jag inledde arbetet hade jag grunduppfattning om Arduino och om att skriva programkod. Under mitt sista skolår hade jag två projekt som innehåll programmering, det gjorde att jag fick en bättre förståelse för hur programmering går till. Men jag upplevde att jag inte förstod mig på allt som jag gjorde, vilket var bland annat hur sensorer kommunicera, vad de olika förkortningarna betydde och vilken uppgift de olika komponenterna hade på ett Arduino kort. Genom att göra en robotarm som styrdes av ett Arduino kort och dokumentera arbetet fick jag en möjlighet att lära mig mer ingående om det jag inte visst och det jag var osäker på.

Detta arbete har gett mig en bra grund och förståelse för att kunna utveckla robotarmen eller inleda ett nytt projekt.

Källhänvisning

Arduino, 2016a. *Arduino 101.* [Online]
http://www.arduino.cc/en/Main/ArduinoBoard101 [hämtat: 6.4.2016]

Arduino, 2016b. *Arduino Due.* [Online]
http://www.arduino.cc/en/Main/ArduinoBoardDue [hämtat: 6.4.2016]

Arduino, 2016c. *Arduino Gemma.* [Online]
http://www.arduino.cc/en/Main/ArduinoGemma [hämtat: 6.4.2016]

Arduino, 2016d. *Arduino MEGA 2560.* [Online]
https://www.arduino.cc/en/Main/ArduinoBoardMega2560 [hämtat: 14.3.2016]

Arduino, 2016e. *Arduino Pro.* [Online]
http://www.arduino.cc/en/Main/ArduinoBoardPro [hämtat: 6.4.2016]

Arduino, 2016f. *Arduino Uno.* [Online]
https://www.arduino.cc/en/main/arduinoBoardUno [hämtat: 14.3.2016]

Arduino, 2016g. *Arduino Yún.* [Online]
http://www.arduino.cc/en/Main/ArduinoBoardYun [hämtat: 6.4.2016]

Arduino, 2016h. *Arduino ZERO.* [Online]
https://www.arduino.cc/en/Main/ArduinoBoardZero [hämtat: 15.3.2016]

Arduino, 2016i. *Frequently Asked Questions.* [Online]
http://www.arduino.cc/en/Main/FAQ [hämtat: 13.3.2016]

Arduino, 2016j. *LilyPad Arduino USB.* [Online]
http://www.arduino.cc/en/Main/ArduinoBoardLilyPadUSB [hämtat: 6.4.2016]

Arduino, 2016k. *What does it change for you* [Online]
https://www.arduino.cc/en/Main/GenuinoBrand [hämtat: 4.4.2016]

Arduino, 2016l. *What is Arduino?.* [Online]
https://www.arduino.cc/en/Guide/Introduction [hämtat: 22.3.2016]

Atmel, 2015a. *Atmel 8-bit microcontroller datasheet.* [Online]
http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf [hämtat: 14.3.2016]

Atmel, 2015b. *Atmel 8-bit microcontroller datasheet.* [Online]
http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf [hämtat:14.3.2016]

Atmel, 2016. *SMART ARM-based Microcontroller Datasheet.* [Online]
http://www.atmel.com/Images/Atmel-42181-SAM-D21_Summary.pdf [hämtat: 15.3.2016]

Choudhary, H., 2012. *Difference between Microprocessor and Microcontroller.* [Online]
http://www.engineersgarage.com/tutorials/difference-between-microprocessor-and-microcontroller [hämtat: 14.4.2016]

Eglowstein, H., 2012. *Introduction to Servo Motors.* [Online]
http://www.sciencebuddies.org/science-fair-projects/project_ideas/Robotics_ServoMotors.shtml [hämtat: 2.2.2016]

Furqan, M., 2016. *Arduino: Getting Started Right Out of the Box* [Online]
http://resources.intenseschool.com/arduino-getting-started/ [hämtat 19.4.2016]

Karthik, A., 2016. *Difference between USART, UART, RS232, USB, SPI, I2C, TTL....* [Online]
http://onebyzeroelectronics.blogspot.fi/2016/03/difference-between-usart-uart-rs232-usb.html [hämtat:19.4.2016]

Nelli, F., 2014. *Arduino ZERO – when ZERO is greater than ONE.* [Online]
http://www.meccanismocomplesso.org/en/arduino-zero-quando-zero-e-maggiore-di-uno/ [hämtat: 15.3.2016]

O'Sullivan, D. (u.å). *All About Microcontrollers.* [Online]
http://www.tigoe.com/pcomp/code/controllers/all-about-microcontrollers/ [hämtat: 14.4.2016].

RC Helicopter Fun (u.å.). *Understanding RC Servos* [Online]
http://www.rchelicopterfun.com/rc-servos.html [hämtat: 2.2.2016]

Shield list, 2013. *Arduino Shield List.* [Online]
http://www.shieldlist.org/ [hämtat: 7.4.2016]

Society of Robots, 2014. *Actuators - Servos.* [Online]
http://www.societyofrobots.com/actuators_servos.shtml [hämtat: 2.2.2016]

Sparkfun, 2012. *Serial Communication.* [Online]
https://learn.sparkfun.com/tutorials/serial-communication [hämtat: 30.3.2016]

Sparkfun, 2013a. *Analog vs. Digital.* [Online]
https://learn.sparkfun.com/tutorials/analog-vs-digital [hämtat: 23.3.2016]

Sparkfun, 2013b. *Arduino Shields.* [Online]
https://learn.sparkfun.com/tutorials/arduino-shields [hämtat: 7.4.2016]

Sparkfun, 2013c. *Installing an Arduino Bootloader.* [Online]
https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader [hämtat: 6.4.2016]

Sparkfun, 2013d *What is an Arduino.* [Online]
https://learn.sparkfun.com/tutorials/what-is-an-arduino [hämtat: 23.3.2016]

Sparkfun, 2016. *Sensors.* [Online]
https://www.sparkfun.com/categories/23?page=all [hämtat: 6.4.2016]

Wikipedia, 2009. *Microcontroller.* [Online]
https://en.wikipedia.org/wiki/Microcontroller [hämtat: 14.4.2016]

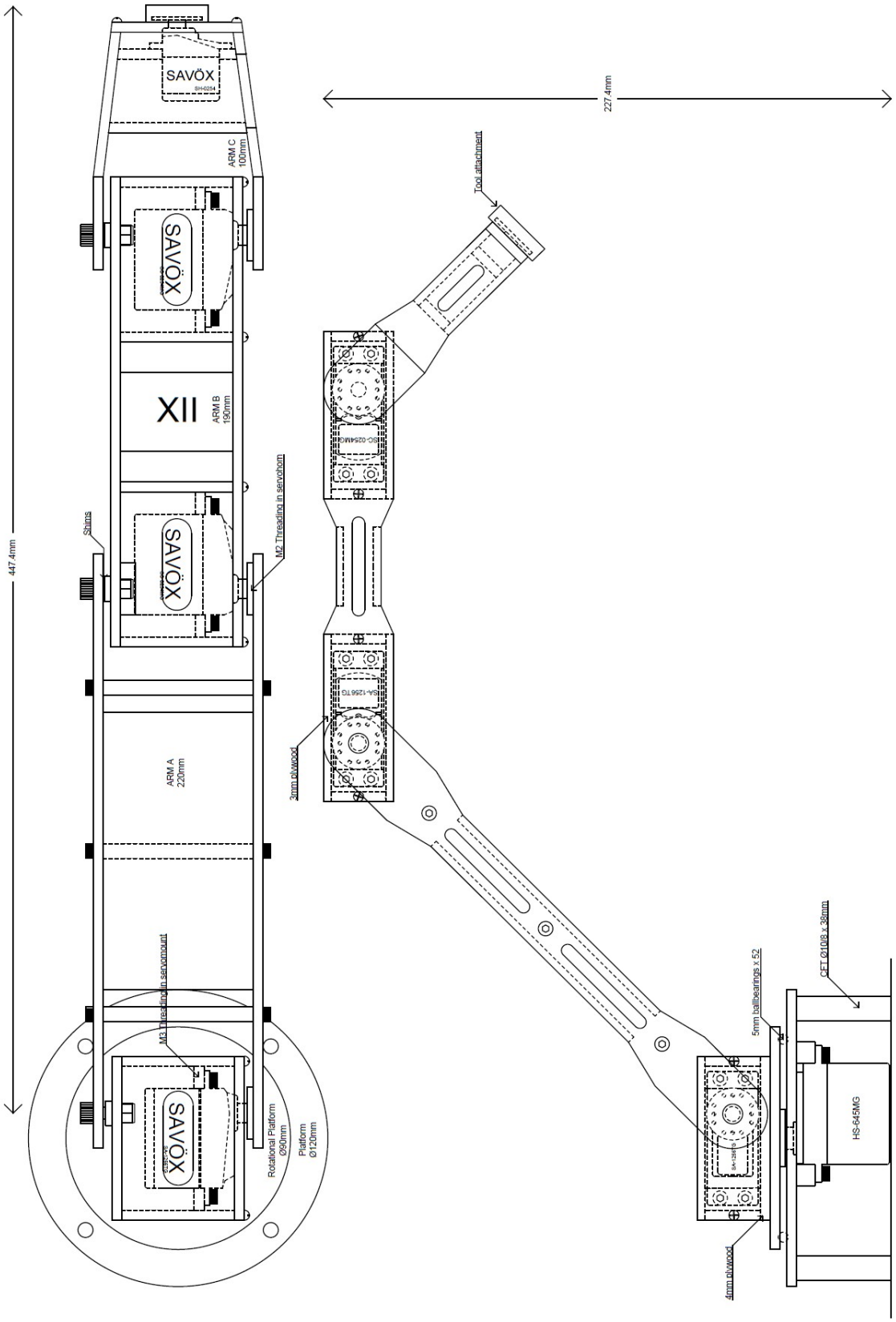
Bilagor

Bilaga 1: CAD ritning: Robotarm

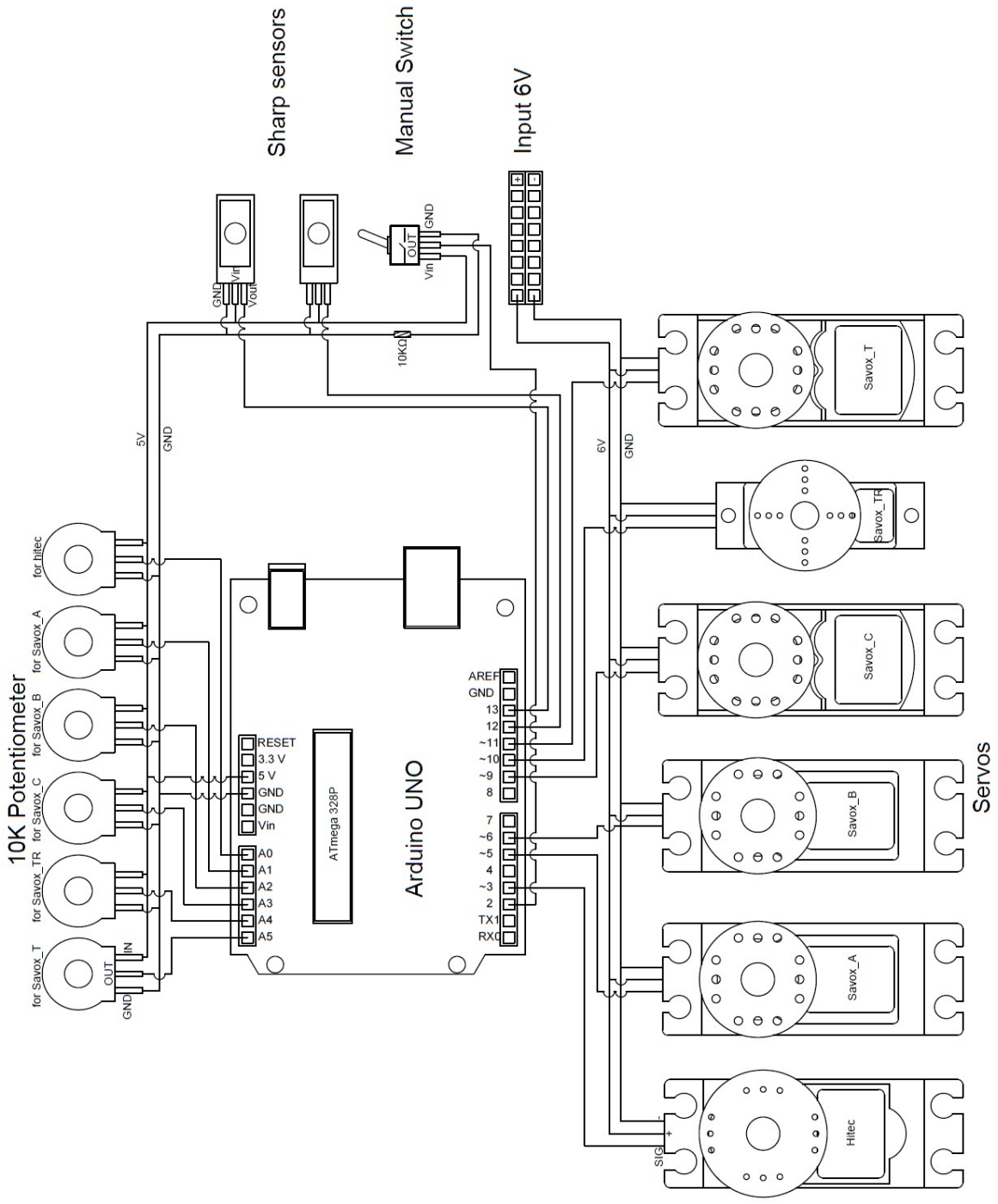
Bilaga 2: Kopplingsschema

Bilaga 3: Programkod

Bilaga 1


447.4mm
227.4mm
SAVÖX
ARM C
100mm
XII
ARM B
150mm
ARM A
220mm
Shims
M2 Threading in servohorn
M3 Threading in servomount
Rotational Platform
Ø90mm
Platform
Ø120mm
Tool attachment
3mm plywood
4mm plywood
5mm ballbearings x 52
CFT Ø1008 x 38mm
HS-645MG

Bilaga 2


Sharp sensors
Manual Switch
Input 6V
10K Potentiometer
for Savox_T
for Savox_TR
for Savox_C
for Savox_B
for Savox_A
for hitec
OUT
IN
GND
5V
GND
6V
GND
Vin
Vout
OUT
Vin
GND
10KΩ
RESET
3.3 V
5 V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
ATmega 328P
Arduino UNO
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX1
RX0
Savox_T
Savox_TR
Savox_C
Savox_B
Savox_A
Hitec
Servos

Bilaga 3

```
#include <Servo.h>

Servo hitec;                                    //Rotational servo
Servo savox_A;                                  //Arm A servo
Servo savox_B;                                  //Arm B servo
Servo savox_C;                                  //Arm C servo
Servo savox_TR;                                 //Toolrotation servo
Servo savox_T;                                  //Tool servo

int potpin_H = 0;                               //Analog input: Potentiometer
int potpin_A = 1;
int potpin_B = 2;
int potpin_C = 3;
int potpin_TR = 4;
int potpin_T = 5;

int potpos_H;                                   //Used for servo position
int potpos_A;
int potpos_B;
int potpos_C;
int potpos_TR;
int potpos_T;

int potpos_write_H_R;
int potpos_write_H_L;
int potpos_write_A_R;
int potpos_write_A_L;
int potpos_write_B_R;
int potpos_write_B_L;
int potpos_write_C_R;
int potpos_write_C_L;

int sensor_left = 12;                           // Digital input: Sharpsensors
int sensor_right = 13;

int val_H;                                      //Potentiometer value
int val_A;

int val_B;
int val_C;
int val_TR;
int val_T;

int sensor_reading_left = 1;
int sensor_reading_right = 1;

int pos_A;
int pos_B;
int pos_C;
int pos_H;

int manual_switch_pin = 2;
int manual_switch = 0;


void sharpsensor(){
                    sensor_reading_left = digitalRead(sensor_left);
                    sensor_reading_right = digitalRead(sensor_right);
                    delay(5);
}
```

```
void control(){                                   //Manual control of the robot

                val_H = analogRead(potpin_H);
                val_A = analogRead(potpin_A);
                val_B = analogRead(potpin_B);
                val_C = analogRead(potpin_C);
                val_TR = analogRead(potpin_TR);
                val_T = analogRead(potpin_T);

                potpos_H = hitec.read();
                potpos_A = savox_A.read();
                potpos_B = savox_B.read();
                potpos_C = savox_C.read();

                potpos_write_H_R = (potpos_H + 1);
                potpos_write_H_L = (potpos_H - 1);
                potpos_write_A_R = (potpos_A + 1);
                potpos_write_A_L = (potpos_A - 1);
                potpos_write_B_R = (potpos_B + 1);
                potpos_write_B_L = (potpos_B - 1);
                potpos_write_C_R = (potpos_C + 1);
                potpos_write_C_L = (potpos_C - 1);

                val_TR = map(val_TR,0,1023,1000,2000);
                val_T = map(val_T,0,1023,1300,1950);

                if (val_H < 10){
                                hitec.write(potpos_write_H_R);
                }
                if (val_H > 1010){
                                hitec.write(potpos_write_H_L);
                }

                if (val_A < 10){
                                savox_A.write(potpos_write_A_R);
                }
                if (val_A > 1010){
                                savox_A.write(potpos_write_A_L);
                }

                if (val_B < 10){
                                savox_B.write(potpos_write_B_L);
                }
                if (val_B > 1010){
                                savox_B.write(potpos_write_B_R);
                }

                if (val_C < 10){
                                savox_C.write(potpos_write_C_R);
                }
                if (val_C > 1010){
                                savox_C.write(potpos_write_C_L);
                }

                savox_TR.writeMicroseconds(val_TR);
                savox_T.writeMicroseconds(val_T);
                Serial.println(val_TR);
                Serial.println(val_T);

                delay(25);
}

void center(){                                    //Moves all servos to middleposition
                hitec.writeMicroseconds(1500);
                savox_A.writeMicroseconds(1500);
                savox_B.writeMicroseconds(1500);
                savox_C.writeMicroseconds(1500);
                savox_TR.writeMicroseconds(1600);
                savox_T.writeMicroseconds(1800);
```

```
}

void rest(){// Hitec = 1600, savox_A = 1750 savox_B = 2250, savox_C = 700, savox_TR = 1750, savox_T = 1950

                    for (pos_A = 1500; pos_A <= 1750; pos_A += 5){
                                savox_A.writeMicroseconds(pos_A);
                                delay(20);
                    }

                    savox_T.writeMicroseconds(1950);
                    savox_TR.writeMicroseconds(1750);
                    hitec.writeMicroseconds(1600);

                    for (pos_C = 900; pos_C >= 700; pos_C -= 5){
                                savox_C.writeMicroseconds(pos_C);
                                delay(20);
                    }

                    for (pos_B = 1800; pos_B <= 2250; pos_B += 5){
                                savox_B.writeMicroseconds(pos_B);
                                delay(20);
                    }

}

// Main program for XII
void move_objectOne(){


                    savox_A.writeMicroseconds(1750);
                    savox_B.writeMicroseconds(2250);
                    savox_C.writeMicroseconds(700);
                    savox_T.writeMicroseconds(1950);
                    savox_TR.writeMicroseconds(1750);
                    hitec.writeMicroseconds(1600);

                    for (pos_B = 2250; pos_B >= 1750; pos_B -= 5){
                                savox_B.writeMicroseconds(pos_B);
                                delay(20);
                    }

                    for (pos_A = 1750; pos_A >= 1500; pos_A -= 5){
                                savox_A.writeMicroseconds(pos_A);
                                delay(20);
                    }

                    for (pos_H = 1600; pos_H >= 700; pos_H -= 5){
                                hitec.writeMicroseconds(pos_H);
                                delay(5);
                    }

                    delay(15);
                    savox_T.writeMicroseconds(1500);
                    delay(15);

                    for (pos_A = 1500; pos_A >= 1220; pos_A -= 5){
                                savox_A.writeMicroseconds(pos_A);
                                delay(20);
                    }

                    for (pos_C = 700; pos_C <= 850; pos_C += 5){
                                savox_C.writeMicroseconds(pos_C);
                                delay(20);
                    }

                    for (pos_B = 1750; pos_B <= 1830; pos_B += 5){
                                savox_B.writeMicroseconds(pos_B);
                                delay(20);
```

```
		}

		for (pos_C = 850; pos_C <= 950; pos_C += 5){
				savox_C.writeMicroseconds(pos_C);
				delay(20);
		}

		delay(15);
		savox_T.writeMicroseconds(1880);
		delay(15);

		for (pos_A = 1220; pos_A <= 1500; pos_A += 5){
				savox_A.writeMicroseconds(pos_A);
				delay(20);
		}

		for (pos_H = 700; pos_H <= 1650; pos_H += 5){
				hitec.writeMicroseconds(pos_H);
				delay(10);
		}

		savox_TR.writeMicroseconds(1680);

		for (pos_A = 1500; pos_A >= 1255; pos_A -= 5){
				savox_A.writeMicroseconds(pos_A);
				delay(20);
		}

		for (pos_C = 950; pos_C >= 900; pos_C -= 5){
				savox_C.writeMicroseconds(pos_C);
				delay(20);
		}

		delay(15);
		savox_T.writeMicroseconds(1500);
		delay(15);

		for (pos_A = 1252; pos_A <= 1500; pos_A += 5){
				savox_A.writeMicroseconds(pos_A);
				delay(20);
		}


}

void move_objectTwo(){

		for (pos_H = 1600; pos_H >= 880; pos_H -= 5){
				hitec.writeMicroseconds(pos_H);
				delay(5);
		}

		for (pos_A = 1500; pos_A >= 1350; pos_A -= 5){
				savox_A.writeMicroseconds(pos_A);
				delay(20);
		}

		delay(15);
		savox_TR.writeMicroseconds(2095);
		savox_T.writeMicroseconds(1500);
		delay(15);

		for (pos_B = 1830; pos_B >= 1800; pos_B -= 5){
				savox_B.writeMicroseconds(pos_B);
				delay(20);
		}

		for (pos_A = 1350; pos_A >= 1235; pos_A -= 5){
				savox_A.writeMicroseconds(pos_A);
```

```
                        delay(20);
                }

                delay(15);
                savox_T.writeMicroseconds(1920);
                delay(15);

                for (pos_A = 1235; pos_A <= 1500; pos_A += 5){
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }

                for (pos_B = 1800; pos_B <= 1830; pos_B += 5){
                        savox_B.writeMicroseconds(pos_B);
                        delay(20);
                }

                for (pos_H = 880; pos_H <= 1650; pos_H += 5){
                        hitec.writeMicroseconds(pos_H);
                        delay(10);
                }

                savox_TR.writeMicroseconds(1680);

                for (pos_A = 1500; pos_A >= 1300; pos_A -= 5){
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }

                delay(15);
                savox_T.writeMicroseconds(1500);
                delay(15);

                for (pos_A = 1300; pos_A <= 1500; pos_A += 5){
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }
}

void move_objectsBack(){

                delay(15);
                hitec.writeMicroseconds(1650);
                savox_T.writeMicroseconds(1500);
                savox_TR.writeMicroseconds(1680);
                delay(15);

                for (pos_A = 1500; pos_A >= 1300; pos_A -= 5){                  //Arm at grip one position
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }

                delay(15);
                savox_T.writeMicroseconds(1920);
                delay(15);

                for (pos_A = 1300; pos_A <= 1500; pos_A += 5){
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }

                for (pos_H = 1680; pos_H >= 700; pos_H -= 5){
                        hitec.writeMicroseconds(pos_H);
                        delay(10);
                }

                for (pos_C = 700; pos_C <= 950; pos_C += 5){
                        savox_C.writeMicroseconds(pos_C);
                        delay(20);
```

```
}

for (pos_B = 1750; pos_B <= 1830; pos_B += 5){
                    savox_B.writeMicroseconds(pos_B);
                    delay(20);
}

for (pos_A = 1500; pos_A >= 1220; pos_A -= 5){
                    savox_A.writeMicroseconds(pos_A);
                    delay(20);
}


delay(15);
savox_T.writeMicroseconds(1500);
//Release object
delay(15);

for (pos_A = 1220; pos_A <= 1500; pos_A += 5){
                    savox_A.writeMicroseconds(pos_A);
                    delay(20);
}

for (pos_H = 700; pos_H <= 1650; pos_H += 5){
                    hitec.writeMicroseconds(pos_H);
                    delay(10);
}

savox_TR.writeMicroseconds(1680);

for (pos_A = 1500; pos_A >= 1255; pos_A -= 5){
                    savox_A.writeMicroseconds(pos_A);
                    delay(20);
}

for (pos_C = 950; pos_C >= 900; pos_C -= 5){                    //Arm at grip two position
                    savox_C.writeMicroseconds(pos_C);
                    delay(20);
}

delay(15);
savox_T.writeMicroseconds(1900);
delay(15);

for (pos_A = 1252; pos_A <= 1500; pos_A += 5){
                    savox_A.writeMicroseconds(pos_A);
                    delay(20);
}

for (pos_H = 1600; pos_H >= 880; pos_H -= 5){
                    hitec.writeMicroseconds(pos_H);
                    delay(5);
}

for (pos_A = 1500; pos_A >= 1350; pos_A -= 5){
                    savox_A.writeMicroseconds(pos_A);
                    delay(20);
}

delay(15);
savox_TR.writeMicroseconds(2095);
delay(15);

for (pos_B = 1830; pos_B >= 1800; pos_B -= 5){
                    savox_B.writeMicroseconds(pos_B);
                    delay(20);
}

for (pos_A = 1350; pos_A >= 1235; pos_A -= 5){
```

```
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }

                delay(15);
                savox_T.writeMicroseconds(1500);
                //Release object
                delay(15);

                for (pos_A = 1235; pos_A <= 1500; pos_A += 5){
                        savox_A.writeMicroseconds(pos_A);
                        delay(20);
                }
}

//Main program end

void setup(){
                hitec.attach(3,600,2400);               //Standard is 1000,2000µs and max is 600,2400µs
                savox_A.attach(5,600,2400);
                savox_B.attach(6,600,2400);
                savox_C.attach(9,600,2400);
                savox_TR.attach(10,600,2400);
                savox_T.attach(11,1300,1950);

                pinMode(sensor_left,INPUT);
                pinMode(sensor_right,INPUT);
                pinMode(manual_switch_pin,INPUT);
}

void loop(){
                sharpsensor();
                manual_switch = digitalRead(manual_switch_pin);

                if ((sensor_reading_left == LOW) || (sensor_reading_right == LOW)){
                }

                if (manual_switch == HIGH){
                        move_objectOne();
                        move_objectTwo();
                        delay(5000);
                        move_objectsBack();
                        rest();
                        delay(5000);
                }

                if (manual_switch == LOW){
                        control();
                }
}
```