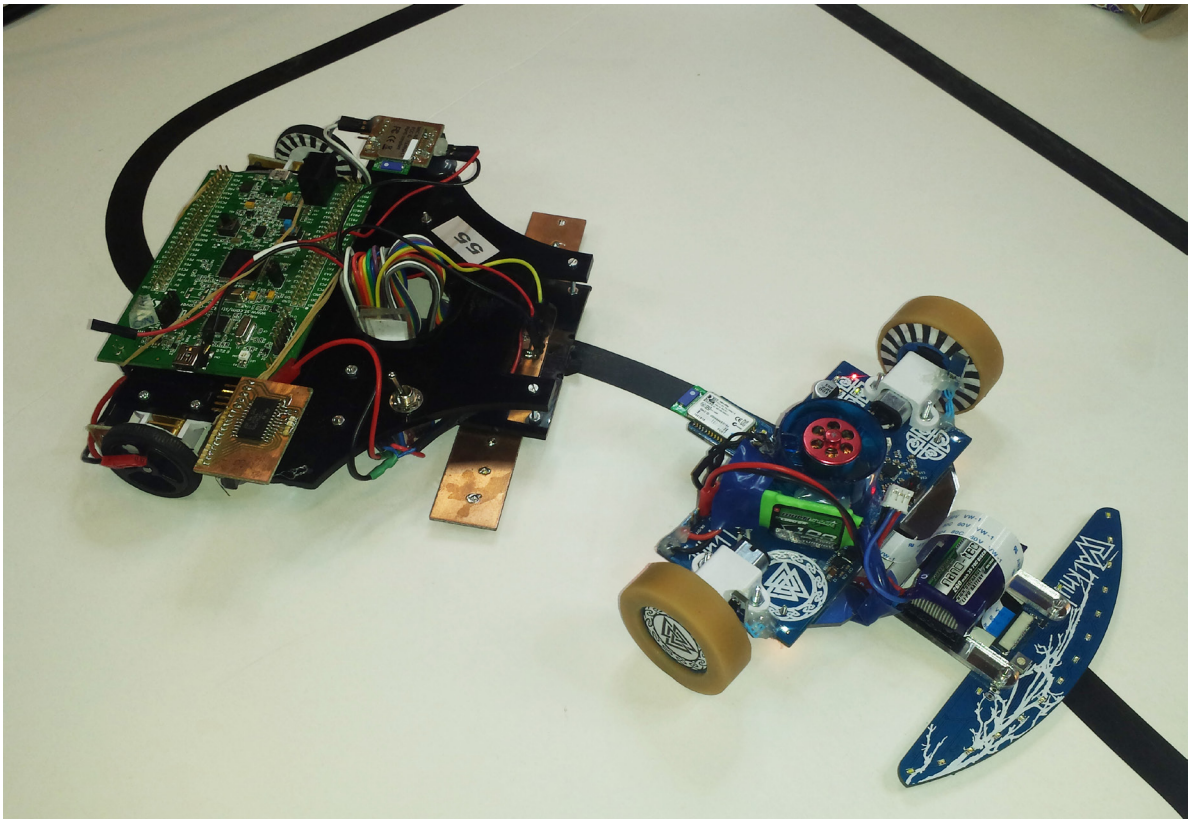
CHALMERS



Linjeföljare med STM32

Med fokus på konstruktion och implementation

Kandidatarbete inom data- och informationsteknik

BÄCKSTRÖM, LARS
PETTERSSON, TOMAS
TILLBERG, HANNA
ÅKERLUND, FREDRIK

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2015-06-01

Abstract

This bachelor thesis aims to design and construct a line follower. A line follower is a robot which follows some kind of path, in this case a black line on a white surface. A prototype was constructed early in the project and was used as a test platform for the final line follower. The robot's microcontroller is a STM32 from STMicroelectronics and the prototype was based on a STM32 discovery board. The robot is designed from scratch which means that the project include: choice of components, choosing between sensors or camera to identify the line, constructing the hardware, designing a regulator and programming the microcontroller unit. The robot constructed is intended to agree with the rules of line following in the race Robot-SM 2015. The prototype "Rulle" participated in the event who also completed the track in less than three minutes, which was required to qualify in the race. The final line follower "Walknut" was completed after Robot-SM and hence did not participate in the event.

Sammandrag

Detta kandidatarbetes syfte var att designa och bygga en linjeföljare. En linjeföljare är en robot som följer någon slags bana, i detta fall en svart linje på en vit yta. En prototyp byggdes i början av projektet och användes som testplattform för den slutgiltiga linjeföljaren. Robotens mikrokontroller är en STM32 från STMicroelectronics och prototypen var baserad på ett STM32 discoverykort. Roboten är konstruerad från grunden vilket gör att kandidatarbetet inkluderar: grundläggande designval, val av komponenter, att välja mellan givare eller kamera för att identifiera linjen, konstruera hårdvaran, designa en regulator och programmering av mikrokontrollern. Robotens konstruktion är avsedd att överensstämma med reglerna för grenen linjeföljning i Robot-SM 2015 och prototypen ”Rulle” deltog i evenemanget. Den slutförde ett varv på banan på under tre minuter, vilket behövdes för att kvalificera sig i loppet. Den slutgiltiga linjeföljaren, ”Walknut”, sammanställdes efter Robot SM och tävlade således aldrig.

Förkortningslista och nyckelord

ABS	Akrylnitril-Butadien-Styren. En typ av amorf plast
Accelerometer	Givare som mäter acceleration
ADC	Analog Digital Converter. Analog till Digital Omvandlare
AGV	Automated Guided Vehicle. En typ av autonom industrirobot
Anslutningsben	Fysisk kontakt på mikrokontroller som fungerar som utgång eller ingång
Autonom	Maskin som agerar utan interaktion av människa
Avbrott	Svenskt uttryck av det engelska "interrupt". Instruktion i en mikrokontroller som direkt får den att byta syssla
Baud	Enhet som motsvarar 1 symbol/s
Ben	Synonymt med "Anslutningsben"
Benkonfiguration	Beteckningslista om vad som fysiskt är kopplat på respektive ben
Buckomvandlare	Elektrisk krets. En likspänningsomriktare för nedreglering av spänning
Byte	Enhet som motsvarar 8 bitar
Buffert	En del av mikrokontrollerns minnen
Börvärde	Reglerteknisk term. Värde som system önskas anta
C	Programmeringsspråk
CAD	Computer Aided Design. Programvara för ritningar
CTS	Clear To Send. Kort meddelande som används i kommunikationssystem för att reglera flödet av information
Discoverykort	Utvecklingskort för STM32
DC	Direct Current. Likström
DMA	Direct Memory Access. Funktionalitet i mikroprocessorn som tillåter hårdvaran att kommunicera direkt med minnet
Duty cycle	Procentsats som en PWM-signal är hög under en period
EDF	Electrical Ducted Fan. En typ av fläkt som är monterat i ett rör vilket ökar dess sugkraft
Exekvera	När en processor utför sina instruktioner
Givarrstrip	Kretskort med givare monterade
GPIO	General Purpose Input Output. Anslutningsben som i mjukvara kan definieras till att antingen ta en insignal eller ge en utsignal
Hjälpande hand	Stativ med klämmor på arm
IMU	Inertial Measurement unit. Krets som innehåller accelerometrar och gyroskop för att skatta rörelse

IR	Infraröd. Elektromagnetisk strålning med våglängder mellan 700 nm och 1 mm
IR-lysdiod	Lysdiod som sänder ut IR-strålning
Kapacitans	Elektrisk storhet. Förmåga att hålla elektrisk laddning
Kondensator	Elektrisk komponent som huvudsakligen har en kapacitans
Kretskort	Mönsterkort med komponenter monterade
LED	Light Emitting Diode. Lysdiod
Lipo	Uppladdningsbart batteri bestående av av litium och polymer
MCU	Microcontroller Unit. Synonymt med mikrokontroller
Mikrokontroller	Liten dator. Utgörs ofta av endast ett chip utan kringutrustning
Motordrivare	Kraftelektronik och logik för styrning av motor
Mönsterkort	Kort av isolerande material med elektriska ledare på ytan. Används som bas för att montera elektriska kretsar
Optokopplare	Komponent med en IR-lysdiod och en fototransistor
Oscilloskop	Mätinstrument med bildskärm. Används för att se signalers nivå och form
PCB	Printed Circuit Board. Kretskort
Periferienhet	Enheter i mikroprocessorns som hanter skrivning och läsning av minne
PID-regulator	En komponent inom reglertekniken som minskar reglerfelet
Polla	Datateknisk term. När processorn kollar status på en resurs med ett bestämt tidsintervall
Python	Programmeringsspråk
PWM	Pulse Width Modulation. Metod som skickar information genom att variera bredden hos en puls
Reglerfel	En signal som motsvarar skillnaden mellan är- och börvärde
Resistans	Elektrisk Storhet. Elektriskt motstånd
Resistor	Elektrisk komponent som i huvudsak utgörs av elektriskt motstånd
Rippel	Återkommande störning på en elektrisk signal. Synonymt med brumspänning
RTS	Request To Send. Kort meddelande som används i kommunikationssystem för att kontrollera flödet av information
Rx	Reciever. Allmänt uttryck för mottagare
Simulink	Programvara från Mathworks som bland annat används för simulering
SPI	Serial Peripheral interface. Buss för seriell kommunikation
STM32F405RG	32-bitars mikrokontroller från STMicoelectronics med 64 st anslutningsben
STM32F407VG	32-bitars mikrokontroller från STMicrocoelectronics med 100 st anslutningsben
Styrsignal	Utsignalen från en regulator

Takometer	Givare som mäter varvtal
Timer	Interndel av mikroprocessor som hanterar tidsintervall
Transistor	En typ av halvledarkomponenter
Tx	Transmitter. Allmänt uttryck för sändare
UART	Universal Asynchronous Reciever/Transmitter
Waijung	Simulinkbibliotek med STM32 block. Från www.waijung.aimagin.com
Ärvärde	Reglerteknisk term. Värde som system håller för tillfället

Förord

Denna rapport avhandlar kandidatarbetet *Linjeföljare* vid *Chalmers Tekniska Högskola* vid institutionen för *Data- och informationsteknik* under vårterminen 2015. Projektet utfördes av teknologer som går grundutbildning inom *Automation och mekatronik, Datateknik,* samt *Elektroteknik*.

Kandidatgruppen vill tacka handledaren Sven Knutsson och examinatorn Arne Linde, för det stöd de har utgjort under projektets fortlöpande, samt Chalmers Robotförening för stöd och för tillgång till deras eminenta lokaler med tillhörande utrustning och komponentlager.

Innehåll

1 Inledning

I den här rapporten redovisas kandidatarbetet *Linjeföljare*. En linjeföljare är en robot som är självgående och följer en linje. En linje kan till exempel utgöras av en elektrisk ledare, ett magnetiskt band eller en färgad linje på ett ljust underlag. I det här projektet definieras linjeföljaren som en autonom robot som följer en svart linje på ett vitt underlag.

Uppgiften var att ta fram en linjeföljare enligt reglementet för grenen linjeföljning under Robot-SM (se Appendix I). Roboten ska i en sådan tävling slutföra ett varv på banan på så kort tid som möjligt. I kraven ingår bland annat att roboten måste vara helt autonom och således inte får kommunicera med någon extern enhet. Allt som roboten använder sig av måste finnas installerat ombord.

1.1 Bakgrund

Linjeföljning är ett intressant område då det är ett enkelt sätt att bekanta sig med robotteknik på hobbynivå. Reglertekniska problem inom både hård- och mjukvara måste lösas och ett linjeföljarprojekt utgör därigenom en god möjlighet för den som vill utveckla sina färdigheter inom ämnet. Arbetet som helhet kan ses som ett instegsprojekt som motiverar till vidare studier i robotik, artificiell intelligens och autonoma system.

Robottävlingar medverkar till att höja intresset för robotar i allmänhet, men för att ställa upp i dessa krävs viss kunskap. Detta projekt ämnar bidra med information som kan minska kunskapsglappet för den oinvigde robotbyggaren.

Linjeföljning kan användas inom industrin för att enkelt styra autonoma robotar. En typ av robotar som används inom industrin är AGV, Automated Guided Vehicle, som är en autonom truck. AGV:er kan användas för materialhantering som ett autonomt transportsystem för interna transporter, vilket är kostnadseffektivt och ger mindre antal transportskador än vad som uppstår i traditionella transportsystem [1].

Enligt [2] installerades 1993 en AGV i Thamesport-hamnen för att hantera last från fraktfartyg. Denna AGV var en 17.5 ton tung robot som kunde frakta en sjöcontainer åt gången och kunde interagera med de bemannade truckarna. Navigering skedde med millimeterradar och fyrar med kända positioner utplacerade i omgivningen.

I större och mer avancerade system med många AGV:er kan modern teknik, där man frångår den traditionella tekniken med en fix bana, användas. Det finns moderna AGV:er som är frigående enligt en mjukvarudefinierad föredragen bana, är självlärande och som på egenhand kan ta beslut [3].

1.2 Problembeskrivning och avgränsningar

Detta kandidatarbete ämnar att färdigställa en robot som uppfyller reglerna för Robot-SM och presterar tillräckligt bra för att placera sig på tävlingen, det vill säga klarar av ett varv på banan under 3 minuter.

Styrsystemet ska programmeras på en STM32 mikrokontroller [4]. Enligt planeringsrapporten ska en eller flera prototyper först utvecklas på ett STM32 discoverykort som är en utvecklingsplattform för STM32. Den slutgiltiga linjeföljaren ska baseras på ett specialbeställt kort vilket endast innehåller de komponenter som används av linjeföljaren.

En blåtandsmodul är en utmärkande komponent för den linjeföljare som projektetgruppen ska konstruera. Denna planeras att används för trådlös överföring av små mängder data, som datainhämtning till en PC samt till att ge regulator nya parametervärden, vilket underlättar trimning av PID-regulatorn och kan användas för felsökning.

Mer avancerad funktionalitet såsom att undvika och navigera runt hinder, algoritmer för att hantera avbrott i linjen, beslut för om linjen tappas samt navigera i rutnät kommer ej att implementeras.

Framdrivning avgränsas till att bestå av ett hjulpar samt stödkula och således utforskas inte möjlighet till flertalet hjulpar, bandhjul eller ben. Om optokopplare visar tillräcklig prestanda kommer kamera ej testas för linjeavkänning.

1.3 Vetenskaplig frågeställning

Hur kan en enkel implementation av ett autonomt, linjeföljande system se ut med avseende på läsning av linjen och reglering av systemet?

Att ta fram designen för en linjeföljare som helst ska kunna prestera bra och sedan realisera vår design genom att tillverka och programmera den. För att få en mer komplett bild av programmeringsproblematiken så skrevs två separata styrsystem, ett skrivet i C och ett i MATLAB/Simulink. Ett beslut togs om att tillverka linjeföljaren från grunden vilket innebär att ett flertal problem, som komponentval, fysiska konstruktionsparametrar och mjukvaruimplementation, kräver lösningar. Att bygga linjeföljaren från grunden kan potentiellt leda till både bättre prestanda och ett mer lärorikt projekt än om arbetet baseras på en köpt robot.

1.4 Syfte

Kandidataretets syfte är enligt riktlinjerna för kandidatarbet [5]:

"I kandidatarbetet skall studenten integrera, fördjupa och utveckla sina kunskaper och färdigheter inom ett begränsat område av det som behandlats inom tidigare genomförda kurser inom programmet. Kandidatarbetet syftar också till att ge kunskaper och färdigheter i ingenjörsmässigt och vetenskapligt arbetssätt."

Detta projekt har som syfte att teknologerna ska applicera sina kunskaper i praktiken samt utveckla sina kunskaper under projektets gång. Dokumentation och tidsloggning är en del av många ingenjörers vardag vilket projektet även ämnar ge erfarenhet av.

1.5 Mål

Kandidatarbetets mål formuleras enligt riktlinjerna för kandidatarbet [5]:

- ”Formulera och avgränsa en problemställning inom det valda ämnet”
- ”Planera arbetet för att lösa och avrapportera problemet med givna resurser”
- ”Söka, inhämta och värdera tillgänglig litteratur och annan bakgrundsinformation”
- ”Integrera och utveckla kunskap inom den valda problemställningen”
- ”Dokumentera projektets genomförande genom journalföring”
- ”Avrapportera kandidatarbetets resultat och hur dessa överensstämmer med arbetets projekt- och effektmål i skriftlig och muntlig form”
- ”Kritiskt granska, värdera och konstruktivt ifrågasätta ett annat kandidatarbete avseende frågeställning, genomförande och resultat”

Projektets mål är att designa och tillverka en linjeföljare som förväntas prestera i någon mån tillfredsställande under Robot-SM. Genomförandet av projektet skall utföras på ett sådant sätt att målen enligt anvisningarna tillgodoses.

1.6 Resultat

I början av projektet planerades att flera prototyper skulle tillverkas och ligga till grund för den slutgiltiga designen. Planen reviderades och istället antogs den första prototypen som testplattform för den slutgiltiga linjeföljaren. Prototypen förbättrades iterativt och komponenter testades och ersattes kontinuerligt under projektets gång. Prototypen som användes som testplattform fick namnet Rulle, medan den slutgiltiga linjeföljaren gick under namnet Walknut.

Målet att konstruera en robot som autonomt följer en linje och överensstämmer med reglerna för linjeföljning i Robot-SM nåddes. Prototypen Rulle tog sig runt banan på 30.49 sekunder och placerade sig därmed på sjätte (6:e) plats av sju (7) tävlande. Walknut hann inte färdigställas innan Robot-SM.

2 Material och Metod

Här redogörs för hur projektarbetet praktiskt gått till väga för att uppfylla syfte och mål samt uppnå förväntat resultat. För att se vilken utrustning som användes, se Appendix II.

2.1 Tillvägagångssätt och komponenter

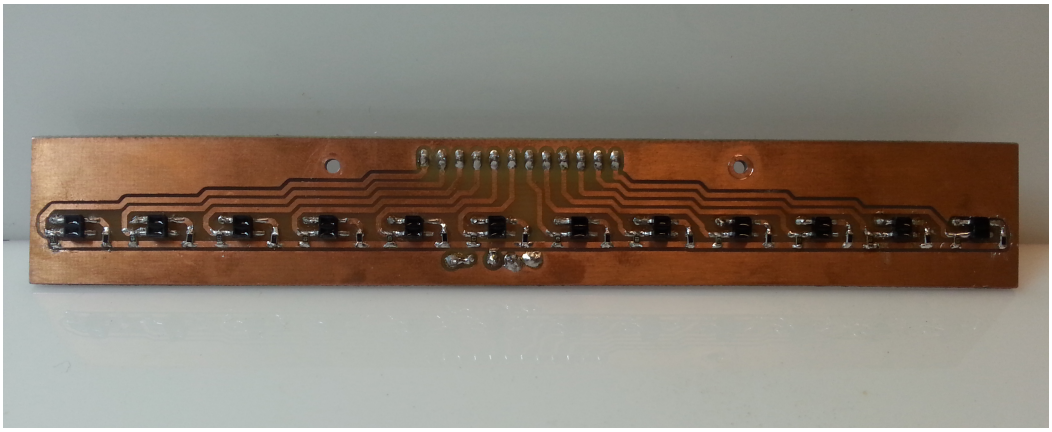
För att färdigställa projektet så krävdes en kravspecifikation. När denna fastställts så kunde arbetet med komponentval, konstruktion och implementation av styrsystemet påbörjas. En beskrivning av detaljerna gällande dessa arbetsmoment och de problem som löstes ges i kommande stycken.

2.1.1 Kravspecifikation

Roboten skall vara konstruerad så att den uppfyller reglerna för att deltaga i Robot-SM 2015. Detta för att kunna ställa upp i grenen linjeföljning under Robot-SM 2015 och även placera sig. För att placera sig krävs att roboten klarar av ett varv på banan under 3 minuter.

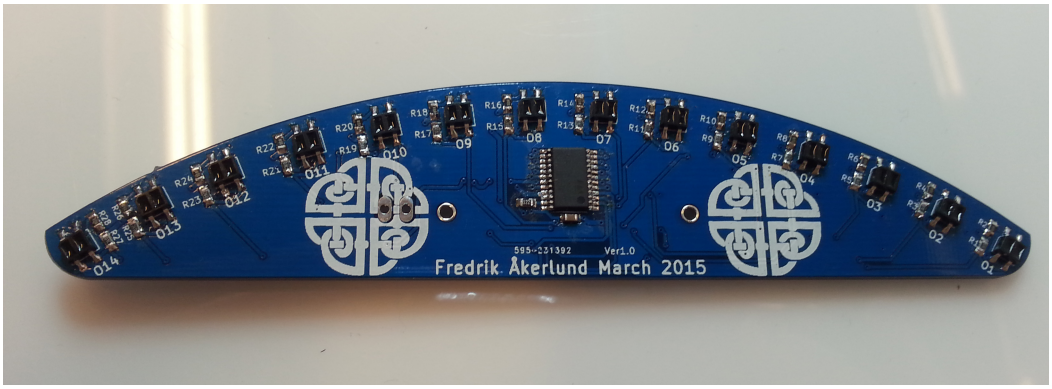
I enlighet med reglerna för Robot-SM fås ett övre tak för dimensioner och vikt på roboten samt att roboten utvecklas för att vara helt autonom. Den ska klara av att följa linjen utan yttre påverkan det vill säga externa kommandon får ej skickas under körning. Se Appendix I för fullständiga regler för Robot-SM 2015.

2.1.2 Grundläggande komponentval



Figur 1: Prototypens egentillverkade givarstrip underifrån. Optokopplarna är de svarta komponenterna

Optokopplare valdes som givare för linjeavkänning. Prototypens givarstipp utgörs av ett kretskort med tolv (12) stycken optokoppare och ett antal resistorer (figur 1). Givarna är monterade längs en rak linje.



Figur 2: Walknuts specialbeställda givarstrip

Den slutgiltiga linjeföljarens givarstrip utgörs av ett specialbetällt mönsterkort på vilket det monterades 14 stycken givare samt ett antal resistorer, jämnt placerade i en cirkelbåge (figur 2).

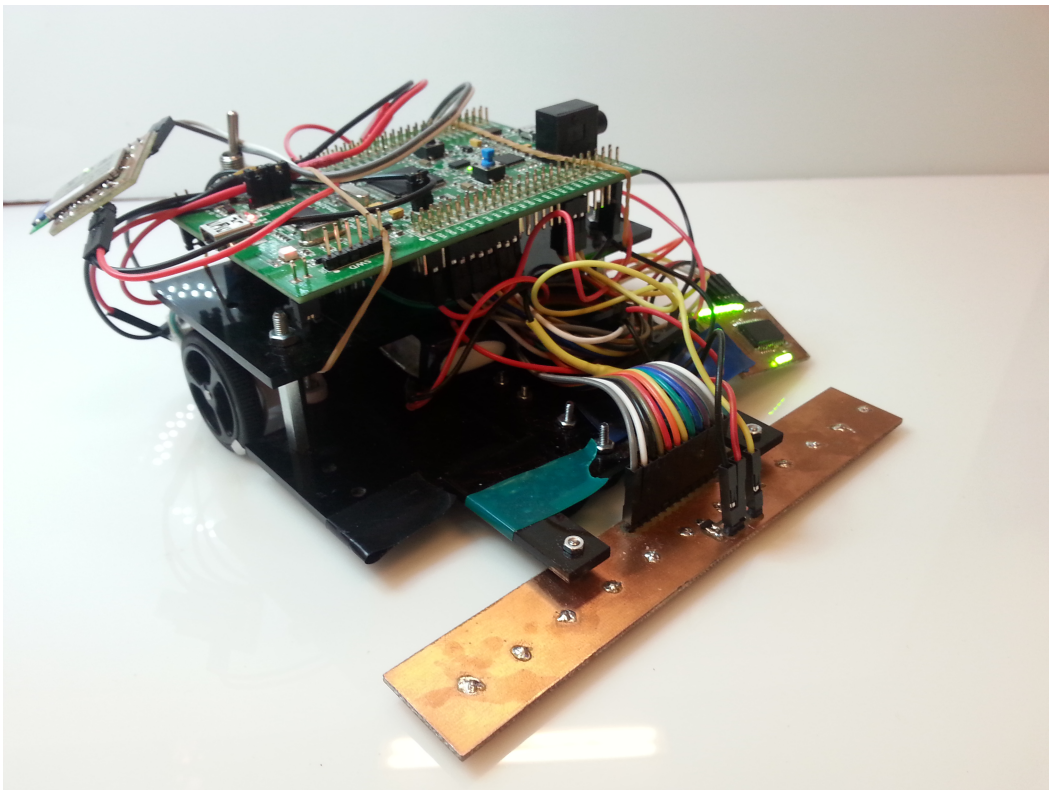
Som mikrokontrollerenhet (MCU) valdes en STM32F407VG [6] för styrning av Rulle. Monterad på ett discoverykort användes denna utvecklingsplattform som centralenhet för prototypen. Denna mikrokontroller är en 32-bitars mikrokontroller som även används inom industrin. Den valda varianten förväntas tillhandahålla den beräkningskraft, det antal ben och den funktionalitet som en linjeföljare kräver. Genom att använda tilläggskort kan denna processor få ett kamerainterface. Det öppnar upp för linjeföljare med kamera istället för optokopplare om dessa visar sig prestera undermåligt. ADC, Analog Digital Omvandlare, används för att läsa de analoga signaler som fås från givarna.

Till den slutgiltiga linjeföljaren valdes istället STM32F405RG [6] som MCU. Denna valdes, trots att den har mindre antal anslutningsben, för att den har lika stor beräkningskraft och samma funktionalitet som utnyttjas av mjukvaran.

Figur 3: Batterier. Det större batteriet användes till Rulle och det mindre batteriet användes till Walknut. Enkrona för skala

Robotarnas energikälla utgörs av ett varsitt batteri (figur 3). Vald batterityp till både prototyp och slutgiltig följare är ett två (2) cells LiPo(Litium-Polymer)-batteri. Denna batterityp anses ha tillräcklig kapacitet för att effektivt kunna testa regulatorparametrar utan att behöva byta batteri för ofta och kan leverera de höga strömmar som kan behövas vid hög acceleration. Prototypen utrustades med ett LiPo-batteri med kapaciteten 950 mAh och Walknut med ett mindre batteri som har kapaciteten 260 mAh. Prototypens batteri har mer kapacitet för att kunna testköra längre tidsperioder, medans Walknut utrustas med ett mindre batteri för att reducera robotens vikt. Det

större batteriet väger 46 g och det mindre 16 g.

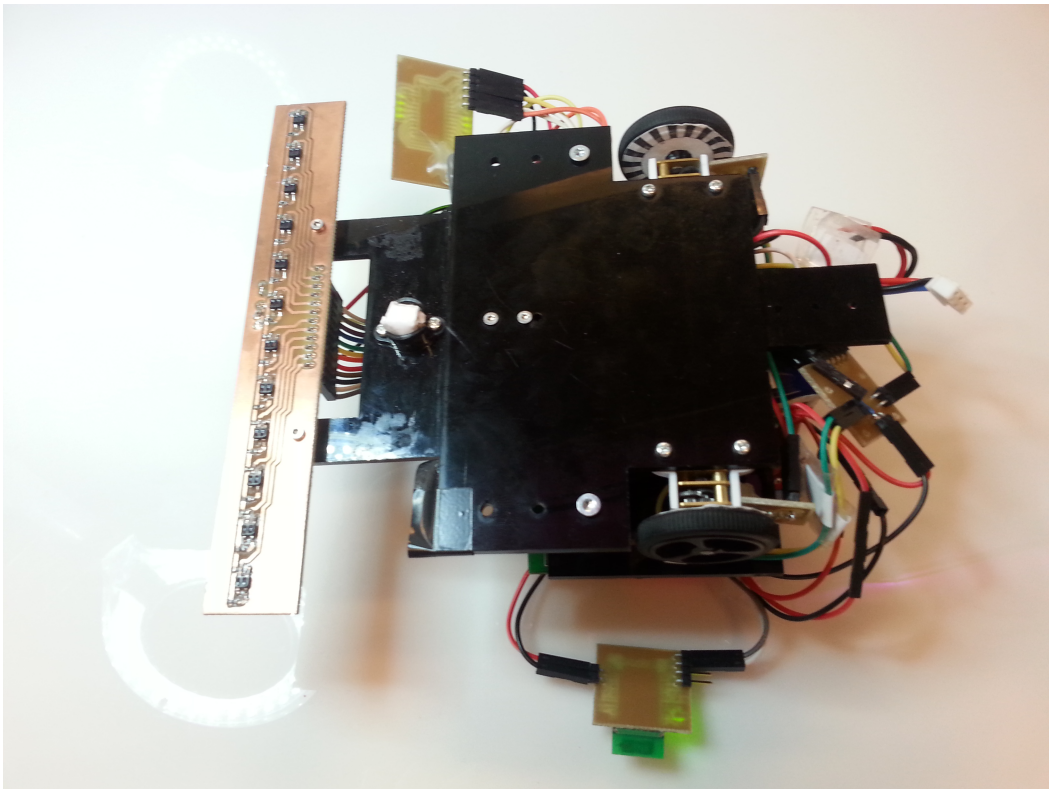


Figur 4: Prototypen Rulle som användes i ett tidigt stadie

Trots att endast en prototyp (figur 4) framställdes övergavs inte konceptet med iterativ prototypning. Den byggdes modulärt, exempelvis genom kontaktering, vilket öppnade upp för stegvis förbättringar. Testplattformens komponenter testades och byttes ut kontinuerligt under projektets gång.

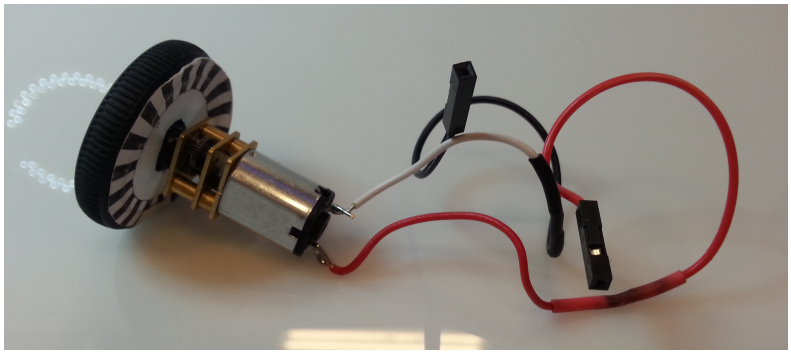
Prototypens chassi bestod av akrylplast och är designades i SolidWorks. Chassit skars ut med en laserskärare och utgjordes av en basplatta med en mindre andra nivå monterad på distanser. Chassit agerade ram på vilken de övriga komponenterna monterades. Discoverykortet hade inte skruvhål så chassit användes för att undvika onödig åverkan på kortet.

Prototypens chassi hade en arm som givarstrippen monterades på. Den utvecklades för att möjliggöra provkörning av roboten med givarstrippen på varierande avstånd från robotens kropp. Givarstrippens avstånd från roboten kunde ha inverkan på robotens prestanda.



Figur 5: Undersidan av den tidiga versionen av Rulle

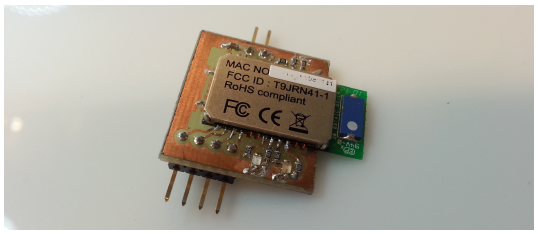
Flera val för framdrivning av robotar är möjliga. Ett hjulpar, flertalet hjulpar, bandhjul eller olika typer av ben kan användas. Vald konstruktion hade ett hjul på höger- respektive vänstersidan i robotens färdriktning och ett stöd i form av en kula bakom givarstrippen (Figur 5).



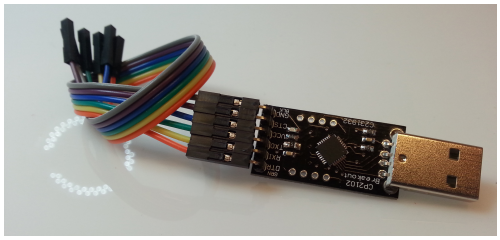
Figur 6: Motor där växellåda samt hjul är synliga

För att minimera den mekaniska konstruktionen valdes att montera hjulen på elmotorerna (figur 6). De valda motorerna [7] levererades med växellådor som hade konstant utväxling. Hjul monterades på växellådans axel, växellådan i sin tur satt direkt på elmotorns axel. Elmotorerna var av typen borstade likströmsmotorer. Både prototypen och den slutgiltiga roboten använde motorer inklusive växellåda med ungefärlig utväxling 30:1.

Programmering utfördes av två olika grupper som framställde kod separat. Den ena gruppen utvecklade C-kod med hjälp av texteditor, och den andra gruppen använde en Simulinkmodell för att autogenerera C-kod.



Figur 7: Blåtandsmodul som kopplades till MCU:ns UART



Figur 8: Serielladapter, användes för att kommunicera via UART mellan robot och PC

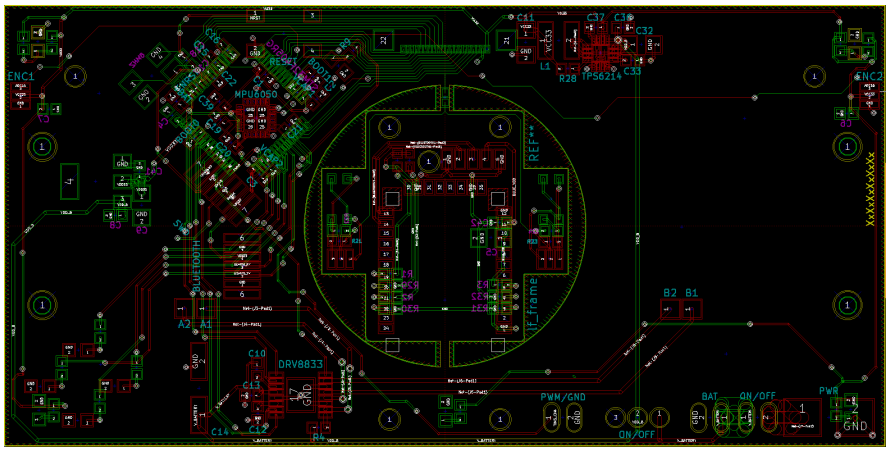
Trådlös kommunikation med linjeföljaren under körning skedde med blåtandsmodulen (figur 7) som kopplades till UART, Universal Ashynchronous Reciever/Transmitter, på mikrokontrollerns ben [8]. Roboten kunde då kommunicera med en laptop som körde ett Python-script. Under utvecklingen användes en serielladapter (figur 8) som fysiskt kopplade ihop roboten med laptopen via kopplingstrådar [9].

2.2 Konstruktion

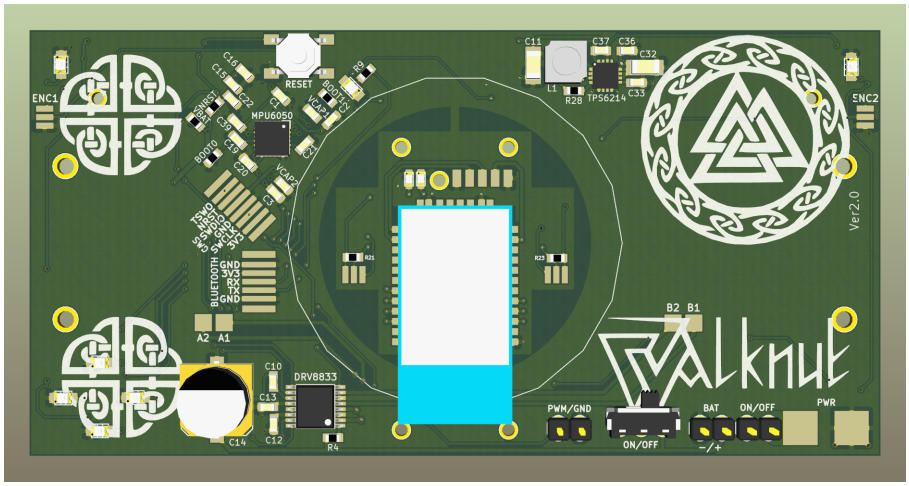
Prototypens design reviderades ett antal gånger och omkonstruerades iterativt. Den slutgiltiga linjeföljaren baserades på specialbeställda mönsterkort, ett för givare samt ett för resten av komponenterna, och sedan monterades alla komponenter för hand på dessa. Det krävdes alltså konstruktion även för Walknut som ej levererades färdigbyggd.

2.2.1 Mikrokontrollerkort

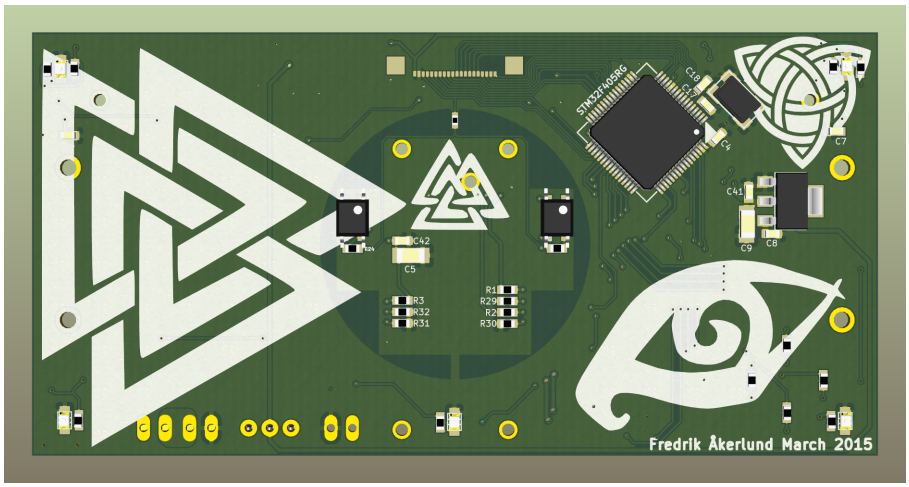
Prototypen hade en stor nackdel i form av ett överflöd av kablar som lätt lossnade. Speciellt utsatt var blåtandsmodulen, som till följd av det resulterade glappet ofta startade om under körning. Discoverykortet bidrog också med extra vikt genom en mängd integrerade komponenter som ej nyttjades. För att lösa dessa tillkortakommanden designades ett uppgiftsspecifikt mikrokontrollerkort. Detta utrustades även med ett monteringshål för en electrical ducted fan (EDF) för att möjliggöra ökad kontroll av marktrycket utan att behöva justera vikten. Även fästen för motorerna integrerades för att förenkla och minimera mängden nödvändig kabeldragning. Även motordrivare avsedda för bruk tillsammans med tre (3) cells LiPo-batteri testades, men integrerades ej på kretskortet efter test med icke önskvärt utfall. Därför behölls den redan fungerade motordrivaren. Resultatet av kretskortsdesignen samt det färdiga mönsterkortet kan skådas i figur 9, figur 10, figur 11 och figur 12.



Figur 9: Kretskortsdesign för mikrokontrollerkortet från KiCad



Figur 10: 3D-modell från KiCad av kretskortets framsida



Figur 11: 3D-modell från KiCad av kretskortets baksida

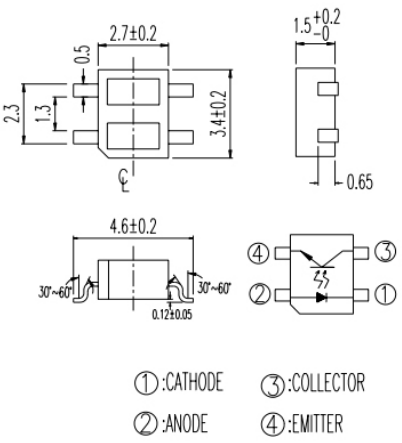
Figur 12: Mönsterkortet i fysisk form med det cirkelformade hålet utskuret, dock utan några pålödda komponenter

Kretskortet ritades i KiCAD och levererades av Mitch Davis (`http://www.hackvana.com/`), en australiensare som via sina kontakter med en fabrik i Kina erbjöd kvalitativ tillverkning till förmånligt pris. Kortet hade dimensionerna 100 × 50 mm och i mitten fanns ett cirkelformat hål med en diameter på 32 mm för montering av en EDF (se figur 12). Ytan som skars bort togs tillvara genom att där förlägga kretskort till nya takometrar samt ett kort för blåtandmodul.

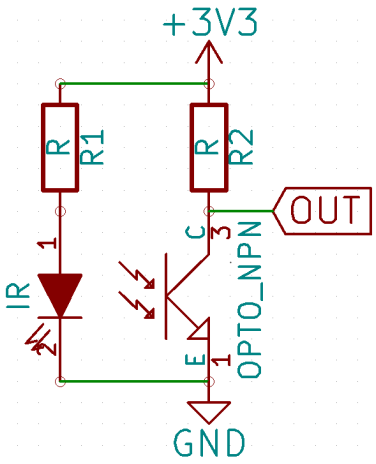
Som processor valdes STM32F405 som är nästintill identisk med STM32F407 [6], men med endast 64 stycken ben istället för 100 stycken. Ovanför processorn sattes en MPU6050-krets [10] innehållande accelerometer och gyroskop (vilket kan implementeras i framtiden). Kopplingen till givarstrippen utgjordes av en tunn kabel endast 10 mm:s bredd. Kortet hade även en stiftlist avsedd för batterianslutning, en motordrivare som i sin tur var avsedd att styra en EDF samt en bygel för att möjliggöra avstängning av motordrivaren. Mellan dessa anslutningar sitter strömbrytare för att slå av eller på MCU:n och den switchade spänningsregulatorn. Spänningsregulatorn gav inte bara ström till givarstrippen utan även till takometrarna, som användes för hastighetsmätning, och sju (7) lysdioder, som agerar utsmyckning. Anslutningarna till motorerna och motordrivaren till EDF löddes fast på exponerade kontaktytor. En bana drogs mellan SPI-klockan och pluspolen på takometrarna. Denna fyllde dock ingen elektrisk funktion utan var enbart ett sätt att komma runt tillverkarens restriktioner mot flera kretskort på samma platta. Kopplingen bröts när komponenterna separerades. Motorerna fästes i hålen på varje sida och gav en total bredd, mätt från motoraxlarnas ändar, om 138 mm.

2.2.2 Optokopplare och ADC

Givarstrippen bestod av ett antal optokopplare som riktades mot underlaget. Varje optokopplare bestod av en kapsel som innehöll en IR-lysdiod och en fototransistor (figur 13). IR-lysdioden sände ut ljus mot underlaget som reflekterades mot detta och träffade fototransistorn. Då underlaget under optokopplaren utgjordes av svart tejp absorberades en stor del av strålningen och lite ljus nådde fotoresistorn. Transistorn hade då hög resistans. Då underlaget istället utgjordes av den vita bakgrunden reflekterades mycket av strålningen som då nådde fototransistorn, vilken då fick låg resistans. Fototransistorns resistans avgjorde hur mycket spänning som låg över transistorn, spänningen kunde mätas av mikrokontrollerenheten vilket gjorde det möjligt att avgöra vilken givare som linjen för närvarande låg under. Kretsen utökades med resistorer (figur 14) för att begränsa strömmen som varje optokopplare krävde av batteriet. De givare som använts för att känna av en linje och även för mätning av hastighet var optokopplare av modell ITR-8307 från Everlight [11].



Figur 13: Visar vilka ben på kapseln som kopplades till fototransistor respektive IR-lysdiod



Figur 14: Visar hur resistanser kopplades till optokopplarna för att strömbegränsa

Svarta ytor reflekterar generellt sett mindre ljus än ljusa ytor. Ytans textur påverkar också hur ljus reflekteras då ojämna ytor sprider ljuset mer än plana ytor. Två ytor som absorberar mycket av ljuset är svart PVC-tejp, vilket är det material som använts som linjematerial på de banor som linjeföljaren körts på, och svart lasertoner som använts till de mönstrade skivor som sattes på insidan av hjulen och användes för hastighetsmätning.

Lysdioderna hde ett framspänningsfall på 1.2 V och klarade en maxström om 50 mA [11], vilket var avsevärt mer än vad som behövs för det korta avstånd som givarna befann sig från underlaget. De strömbegränsades därför med ett motstånd R1 (figur 14) till cirka 20 mA. En linjär spänningsregulator som omvandlar den oönskade effekten till värme skulle vid dessa effekter snabbt överhettas och en switchad dito (en buckomvandlare) användes därför för att reglera ner batterispänningen till 3.3 V.

Dimensionering av motståndet till IR-lysdioden gjordes med Ohms lag enligt ekvation (1):

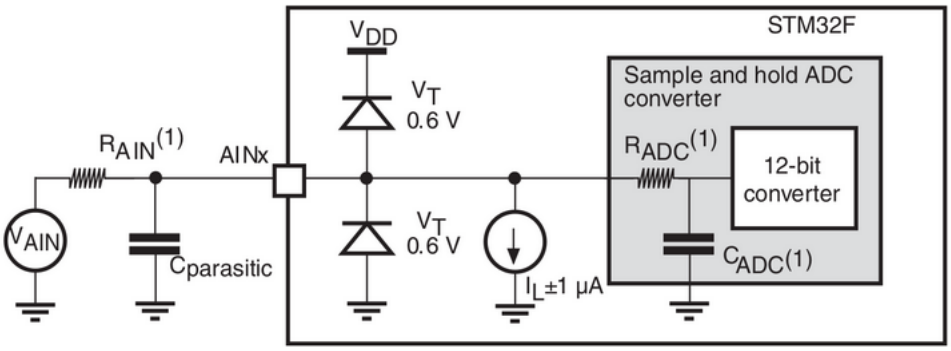
$$R1 = \frac{U}{I} = \frac{3.3 - 1.2}{0.02} = \frac{2.1}{0.02} = 105\ \Omega \tag{1}$$

Då den närmsta tillgängliga storleken var 100 Ω valdes detta med följande strömstryka som resultat, se ekvation (2):

$$I = \frac{U}{R} = \frac{2.1}{100} = 21\ mA \tag{2}$$

Till givarstrippen på Rulle användes tolv (12) stycken optokopplare för att känna av linjen och då uppstod en total strömförbrukning för hela uppsättningen om 252 mA. Till givarstrippen på Walknut användes istället 14 stycken optokopplare och då blev den totala strömförbrukningen 294 mA.

Det behövdes också ett motstånd R2 (figur 14) till fototransistorn för att strömbegränsa denna. Den styrande faktorn för bestämmandet av denna storhet var ADC:n.



Figur 15: Visar hur ingången på en ADC på mikrokontrollern såg ut

Analog till digital omvandlaren, ADC:n (figur 15), var en del av mikrokontrollern. Den tolkade analoga signaler och översätter dessa till digitala värden. En STM32F40x hade sina ADC:er kopplade till en MUX med flera anslutningsben. ADC1 som var den som användes hade en (1) ingång till en MUX med 16 ingångar.

Mellan MUX:en och själva ADC:n satt en samplingskondensator C_{ADC} som får olika spänningar varje gång man byter ingång på MUX:en och under samplingstiden måste den ladda ur eller upp från sitt föregående tillstånd till nya värdet. Är impedansen liten spelar samplingstiden mindre roll eftersom kondensatorn då snabbt antog samma spänning, är den stor skulle kondensatorn inte ha laddats upp till det ingående värdet innan samplingen. Den uppmätta spänningen påverkades då av det föregående värdet.

I databladet [6] för STM32F40x så finner man följande formel, ekvation (3), för att skatta max ingångsresistans:

$$R_{AIN} = \frac{(k - 0.5)}{f_{ADC} \cdot C_{ADC} \cdot ln(2^{N+2})} - R_{ADC} \tag{3}$$

Där k är antalet samplingsperioder som sätts i ADC_SMPRx registret. För det sökta $R_{AIN} = 10$ kΩ så fås ur detta samband, se ekvation (4):

$$k = (R_{AIN} + R_{ADC}) \cdot f_{ADC} \cdot C_{ADC} \cdot ln(2^{N+2}) + 0.5 \tag{4}$$

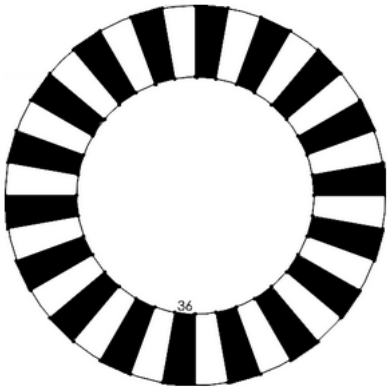
Med värdena $R_{AIN} = 10$ kΩ, $R_{ADC} = 6$ kΩ, $f_{ADC} = 30$ MHz, $C_{ADC} = 4$ pF och $N = 12$, där N är antalet samplingsbitar, erhölls sedan i ekvation (5):

$$k = (6k + 10k) \cdot 30 \cdot 10^6 \cdot 4 \cdot 10^{-12} \cdot ln(2^{14}) + 0.5 \approx 19.1 \tag{5}$$

Därför valdes det närmsta tillgängliga värde som låg över detta, det vill säga 28 samplingcykler. Antalet samplingscykler bestämdes utifrån satta värden i ST:s bibliotek för C-programmering.

2.2.3 Hastighetsmätning

För att uppnå någon form av autonom hastighetsreglering så krävde mätning av den nuvarande hastigheten så att en jämförelse mellan är- och börvärde kunde göras. Om detta gjordes genom mätning av hjulens rotationshastighet möjliggjordes även andra typer av enkla mätningar. Om till exempel antalet varv som hjulen hade snurrat sparats kunde sträckor mätas och i kombination med hastighet- och tidsinformation möjliggjordes allt mer komplexa beteenden.

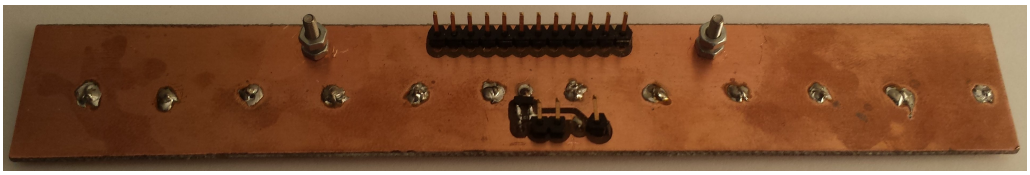


Figur 16: Schematisk bild över den skiva som monterades på insidan av vardera hjul. Användes för att mäta hjulens vinkelhastighet

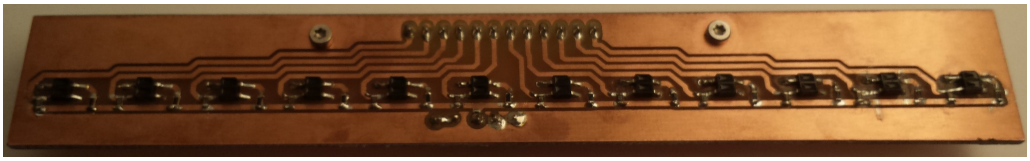
Den implementation som valdes för hastighetsmätning bestod av en svart- och vitrandig skiva som fästes på insidan av varje hjul (figur 16) och en optokopplare riktad mot skivan. Skivan designades så att varje sektion var ungefär lika bred som en givare. Detta resulterade i 36 sektioner, 18 svarta och 18 vita. Varje gång som en rand passerade optokopplaren så förändrades mätvärdet från denna. Varje förändring kunde med mjukvara sedan räknas som ett tick och systemet utgjorde en enkel takometer. För läsning av rotationsriktningen behövs två stycken fasförskjutna givare per hjul, men eftersom antalet kontaktpinnar var begränsat och linjeföljaren bara skulle åka framåt så implementerades aldrig denna funktionalitet.

2.2.4 Givarstrip ett (1)

Som första givarstrip för linjeavkänning så valdes den enklaste geometrin, en rak linje (figur 17 och figur 18). Ett annat alternativ som ansågs intressant att testa var att placera optokopplarna i en båge. Alternativen att placera givare i en matris eller ha fler rader ansågs vara en alltför tidskrävande och onödigt komplicerad lösning. Givarna sattes med ett avstånd om 12 mm från varandra, center-till-center. Detta avstånd valdes eftersom den svarta linje som skulle följas hade bredden 15 mm och det därför var önskvärt med ett avstånd som gör att någon givare alltid låg på linjen. 15 mm var även linjebredden på de testbanor som användes under projektets gång. Prototypens givarstrip egentillverkades och den hade en total längd på 144 mm.



Figur 17: Givarstrippens ovansida



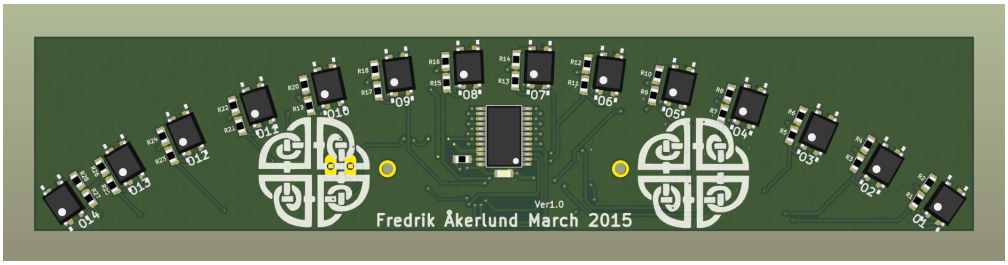
Figur 18: Givarstrippens undersida

Totalt tolv (12) stycken givare placerades längs en rak linje på den första givarstrippen, vilket i grundutförandet krävde mycket ström. Mätt med en multimeter drog en givare ca 20 mA, vilket resulterade i en total strömförbrukning om 240 mA.

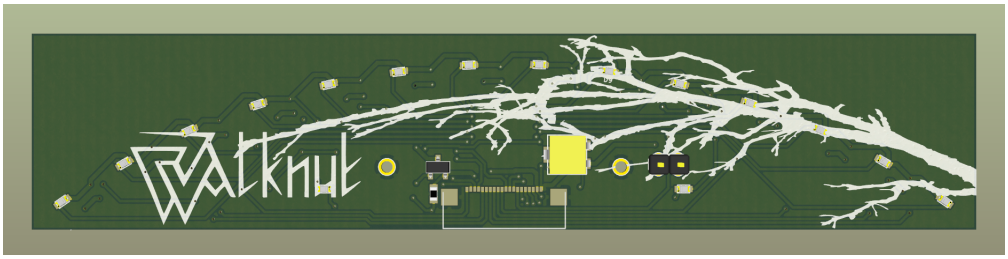
För ett batteridrivet system kan det tyckas vara en onödigt hög ström. Givarna behöver bara matas med ström då läsning av dessa sker. Därför kopplades en transistor in, som i sin tur kopplade alla lysdioders katoder till batteriets minuspol. Transistorn kunde sedan slås på och av med mikrokontrollern precis innan den skulle läsa ADC:n och då samtidigt slå på och av givarna. Denna funktion användes endast i det styrsystem som skrivits i C då Simulink ej erbjöd något enkelt sätt att implementera funktionaliteten. Mätningar med oscilloskop visade en reducering av givarnas strömförbrukning om 92 % jämfört med att ha dem igång kontinuerligt.

2.2.5 Givarstrip två (2)

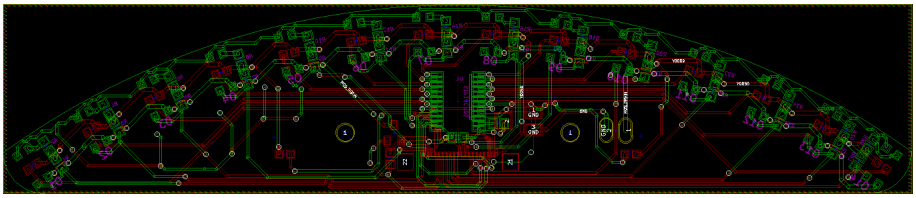
Även en ny givarstrip designades (figur 19, figur 20 och figur 21). Antalet optokopplare ökades till 14 stycken och dessa placerades med ett avstånd ca 9 mm längs en båge med radien 10.36 cm, för att tack vare detta få högre upplösning. Positionsvisande lysdioder placerades på ovansidan av givarna. Dessa kontrollerades med en STP16CP05 [12] och gjorde det möjligt att enkelt se om linjen registrerades av den underliggande optokopplaren. Även en potentiometer för reglering av lysdiodernas ljusstyrka placerades i mitten, precis bakom raden av optokopplare.



Figur 19: Givarstrip tvås (2) undersida



Figur 20: Givarstrip tvås (2) ovansida



Figur 21: Kretsdesign av givarstrip två (2) från KiCad

3 Programmering

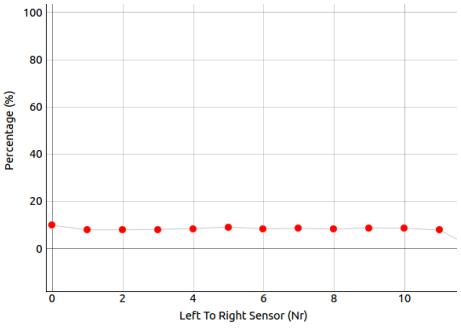
Både C och Simulink är språk som är flitigt använda inom industrin och eftersom de skiljer sig väldigt mycket åt så skapades två (2) styrsystem, ett (1) programmerat i C och ett (1) i Simulink. C är mycket maskinnära och ger precis kontroll över programflöden och beteenden, men kräver också mycket av användaren. Simulink har en hög abstraktionsnivå och erbjuder användaren en mycket begränsad nivå av kontroll på de lägsta nivåerna men ger genom sitt grafiska gränssnitt istället en tydlig överblick över hur information flödar i systemet. Programmeringen i C gjordes med hjälp av ST:s egna bibliotek, medan Simulink-programmeringen istället nyttjade *Waijung blockset* [13] eftersom detta var mycket enklare att komma igång med än ST:s bibliotek. Underrubriken 3.5 avhandlar den kod som implementerats på PC medans resten av kapitlet avhandlar kod på mikrokontrollern.

Simulink är ett tilläggspaket till MATLAB och erbjuder avancerade verktyg för simulering och modellering av system. En modell skapad med Simulink kan sedan användas för att generera kod som kan kompileras och skrivas till en mikrokontroller. Till exempel Airbus har använt Simulink för att autogenerera kod till system ombord på flygplan, bland annat till system för hantering av passagerardörrar[14].

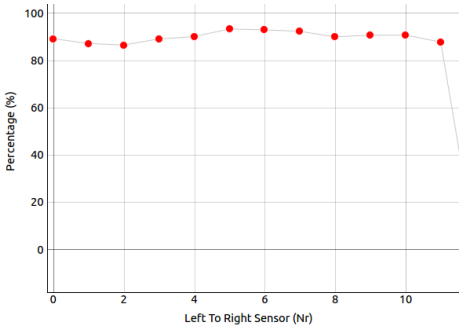
I projektgruppen så hade vissa personer mer erfarenhet av C, andra mer av MATLAB/Simulink. Arbetsgrupperna tog form utefter dessa erfarenheter och resultatet kan därför ses som indikativt för språkens lämplighet vid småskaliga projekt med en STM32-baserad mikrokontroller.

3.1 Värden från optokopplarna för linjeavkänning

I figur 22 redovisas de tolv (12) linjegivarnas värden för ett vitt underlag. Ett optimalt utslag skulle vara 0% men de uppnådda mätvärdena kring 10% var tillräckligt låga för att anses fullt brukbara i detta sammanhang. Figur 23 redovisar motsvarande värden för ett svart underlag. Värdena låg runt 90%, vilket också det anses vara funktionsdugligt, nära det optimala 100%.



Figur 22: Mätvärden för givare mot vitt underlag

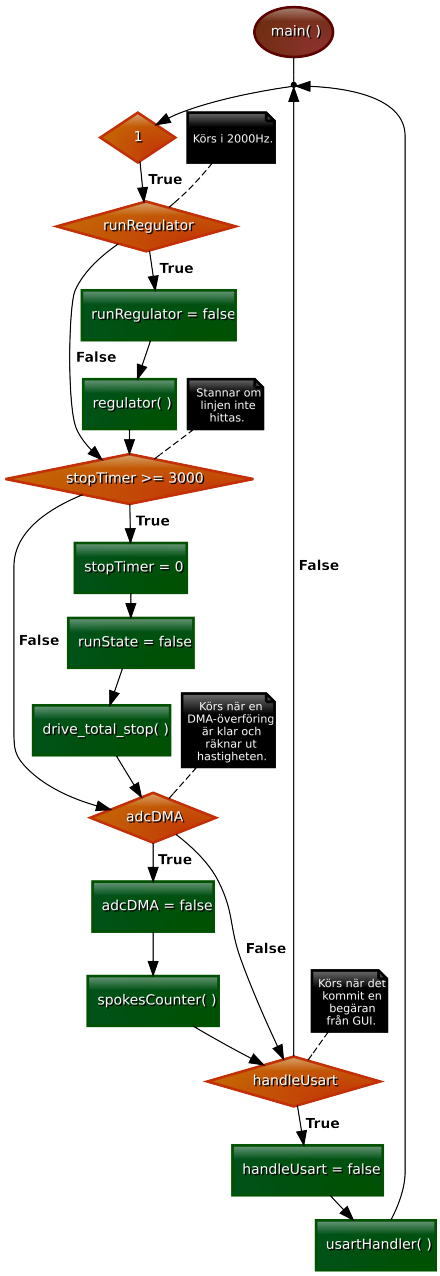


Figur 23: Mätvärden för givare mot svart underlag

När avståndet till ytan ökade så blev värdena allt sämre för att till slut helt tappa sin funktionsduglighet. Anledningen till detta är att kretskortet skuggar underlaget och hindrar mycket av ljuset från omgivningen från att träffa optokopplarna. När avståndet ökar läcker mer ljus utifrån in, samtidigt som ljuset från IR-lysdioderna sprids mer och därför inte reflekteras tillbaka lika starkt.

3.2 Översikt för autonom styrning implementerad i C

Den totala mängden C-kod i projektet består av cirka 1300 rader. Här läggs fokus på de delar som styr roboten. Innan main-funktionen körs så initieras enheter såsom GPIO, ADC, UART, SPI, TIMER samt alla variabler och pekare. Det är detta som gör så att motorer, givare och kommunikation fungerar.



Figur 24: Flödesschema över traditionellt kodade C-programmets main

Main-funktionen bestod av en oändlig loop som kontinuerligt undersökte om olika flaggor blivit satta till 'sant'. De som blivit det ändrades till 'falskt' följt av en exekvering av till flaggan tillhörande kod (figur 24).

Regulatorns flagga sattes i ett timer-avbrott som inträffade 2000 gånger per sekund. I regulatorfunktionen räknades också en variabel 'stopTimer' upp ifall linjen var tappad. Denna variabel kontrollerades sedan av main-funktionen och om räknaren då nått högre än den satta gränsen så tolkades detta som att roboten åkt av banan, med följd att referenshastigheten sattes till noll (0).

En optokopplare läste kontinuerligt av den randiga skivan på hjulet och vid varje avbrott som genererades av DMA (för mer information gällande DMA, se stycke 3.3) lästes värdet av. Hade värdet skiftat från en färg till en annan så räknades en variabel upp och när den nådde 36 noll-

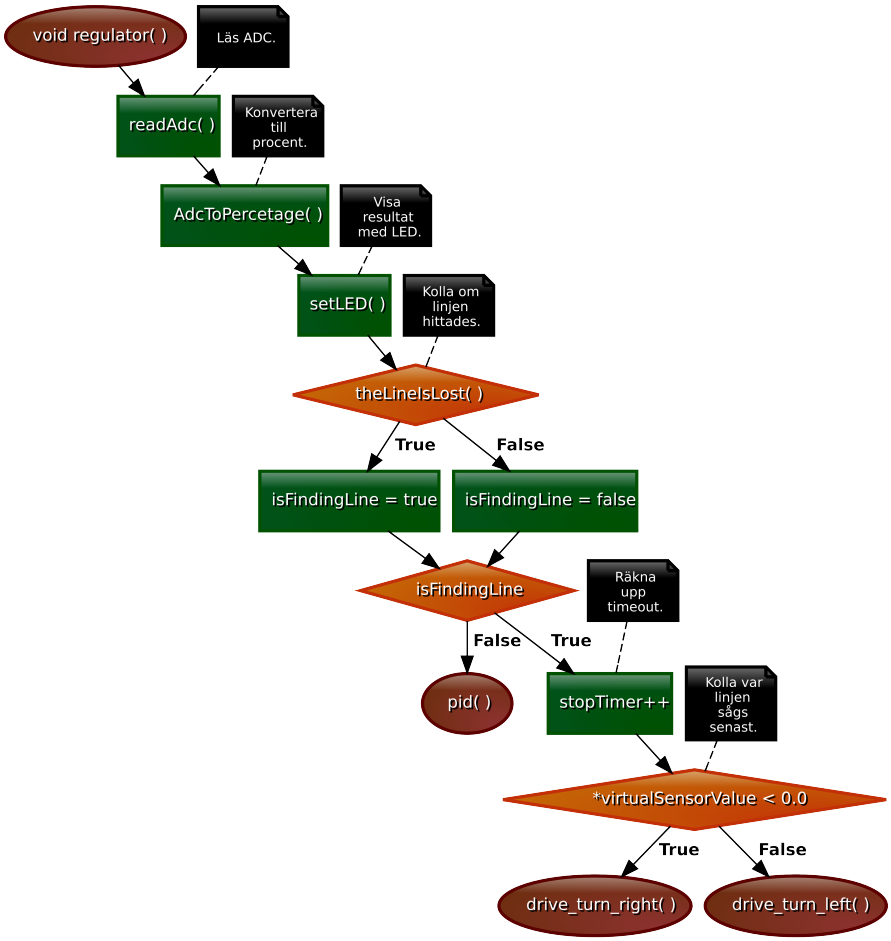
ställdes den samtidigt som en varvräknande variabel räknades upp ett steg. Ett annat avbrott som genererades av en timer användes sedan för att räkna ut vinkelhastigheten genom att ta antalet varv delat med tiden som passerat mellan avbrotten. Hjulets fysiska dimensioner användes sedan för att räkna ut den faktiska hastigheten.

Givarna som läste av hjulet översamplades väldigt mycket. För att visa hur mycket undersöks först hjulets omkrets genom ekvation (6):

$$O = 2 \cdot \pi \cdot (\frac{0.032}{2}) \approx 0.1\ m = 10\ cm \tag{6}$$

Avrundat till 10 cm så ger detta 10 varv/m och om roboten klarade av att köra i 1 m/s gav detta alltså 10 varv/s, vilket innebär att 360 sektioner passerat givaren under en sekund. ADC:n lästes av med DMA 65720 gånger per sekund, vilket ger 182 samplingar per sektion som passerar optokopplaren. Även om roboten skulle klara av en tävlingshastighet likt de snabbaste om cirka 3 m/s så skulle detta fortfarande ge 60 samplingar per sektion.

När användaren ville läsa data från roboten skickades en begäran om detta, via blåtand, som ett paket innehållande en specifik flagga. Om skrivning önskades skickades en annan flagga följt av datan. Båda flaggorna togs emot i ett avbrott som i sin tur sedan satte en annan flagga som signalerade att data inkommit och fanns redo för behandling.



Figur 25: Flödesschema över regulatorn implementerad i C

När regulatorn kördes (figur 25) läste den först av givarna och konverterade sedan dessa värden

till procent av de kända maxvärdena. Därefter tändes robotens LED:s beroende på vilken eller vilka givare som läst av linjen. Om linjen fortfarande befann sig under en givare så kördes PID-funktionen för att reglera motorerna och om linjen ej befann sig under en givare, men tidsgränsen för stopp ej ännu nåtts, svängde roboten åt det håll som linjen senast setts. Om skillnaden mellan linjepositionens är- och börvärde var tillräckligt stort så svängde roboten runt sin egen axel, det vill säga att motorerna gick i motsatt riktning.

Funktionen 'pid( )' agerade PID-regulator och började med att interpolera fram var linjen befann sig (se mer i stycke 4.3) och registrerade sedan på vilken sida av mitten på givarstrippen som linjen befann sig på ifall den skulle vara borttappad vid nästa körning. Därefter räknades felet, alltså skillnaden mellan är- och börvärde ut relativt mitten. Detta värde nyttjades senare för att beräkna styrsignalen till motorerna. Tack vare seriell kommunikation och visuell presentation av datan tydliggjordes snabbt att integraldelen kunde växa sig väldigt stor. Fenomenet var ej användbart och därför implementerades ett minimum och ett maximum för denna storhet.

Både integraldelen och derivatadelen använde ett simpelt lågpassfilter som tog en viss procent 'p' av det nuvarande värdet och (1-p) procent av det nya felet, för att sedan spara summan av de två som de nya värdena för respektive del. Styrvärdet räknades sedan ut med konstanterna Kp, Ki och Kd (se stycke 4.1) som faktorer framför sina respektive variabler. Styrsignalen lades sedan till motorernas bashastighet, fast med olika tecken. Det vill säga att det på den ena motorns styrsignal adderades ett värde medan det på den andra motorn subtraherades. Styrsignalerna skickades sedan som ett argument till den funktion som ställde in motsvarande PWM till motordrivaren.

För att ytterligare förbättra precisionen för de mätvärden som beskrevs i stycke 3.1 togs en funktion fram för att kalibrera givarna. Kalibreringen bestod av att låta systemet först mäta upp ett max- och ett minvärde mot ett mörkt och ett ljust underlag. Skulle till exempel en givare ha haft ett minimum på 400 och ett maximum på 3800 så skulle den alltså ha ett spann på 3400. När sedan ett värde lästes, exempelvis 2100, så subtraherades 400 från detta. Kvar blev 1700 som sedan användes för att beräkna hur många procent av maxvärdet som givarsignalen representerade, i detta exempel 100*1700/3400 = 50 %.

Först var funktionen programmerad för att hitta max- och minvärdena under en viss tid då användaren samtidigt svepte givarna över en linje. Detta gjorde dock att vissa givare fick felaktiga spann, troligtvis för att givarna inte legat helt parallellt mot underlaget under hela manövern, eller för att smuts stört kalibreringen. Funktionen skrevs därför om för att kalibrera mörkt och ljust underlag var för sig.

Styrsystemsimplementationen skriven i C använde DMA för ADC:erna och UART. Eftersom den alltid var aktiverad fanns de senaste värdena alltid tillgängliga i bufferten. Genom att slå av och på ett anslutningsben under DMA-avbrottet kunde frekvensen för dataöverföringar mätas med ett oscilloskop (Rigol DS2072A). Vid en sådan mätning registrerades en frekvens om 32,86 kHz, vilket tolkas som att ADC-data fördes över 65720 gånger per sekund. Datan som förts över innefattade alla 16 ADC-mätningar. Från detta beräknades antalet samplingar per körd meter vid en given hastighet. Till exempel gav en hastighet om 1 m/s cirka 657 stycken samplingar per körd centimeter. Om regulatorn i koden då kördes i 2 kHz så skulle justeringen från PID-kontrollern ske 20 gånger per centimeter.

Datan som skickades seriellt via UART och en blåtandsmodul (RN41) hade en överföringshastighet om 115200 baud, vilket var modulens maximala kapacitet. Genom användandet av DMA, istället för att skicka datan sekvensiellt, sparades mycket exekveringstid. En skillnad mot tidigare nämnd DMA-användning var att den ej var aktiverad kontinuerligt utan endast startades när ett paket som begärde att datan skulle skickas kommit in.

3.3 DMA

Direct Memory Access (DMA) är en funktion i hårdvaran, hos till exempel en STM32 mikroprocessor, som låter andra delar av processorn få direkt åtkomst till minnet oberoende av CPU:n.

Mer korrekt är det en AHB [15], en Advanced High-performance Bus, som för över data mellan olika moduler, som till exempel minnet och en ADC eller UART. Med andra ord kan en perifierienhet kopplas till att automatiskt få sin data överförd till vald minnesplats utan inblandning av processorn (CPU). Medan detta händer är alltså CPU:n fri att ta hand om andra saker och när dataöverföringen är klar finns möjligheten att sätta en flagga så att ett avbrott startas. Med avbrott menas att processorn hoppar till en vald adress och kör den koden som finns där. Alltså kan ADC:n kopplas till en buffert och först när alla ADC:er har blivit avlästa körs vald kod eller ingen kod alls och processorn slipper polla och vänta på begärd data utan kan istället under tiden sköta andra uppgifter.

3.4 Översikt för Simulinkbaserade styrsystemsversioner

Det styrsystem som tagits fram i Simulink har genomgått ett flertal revisioner där lösningar av problem och förbättringar kontinuerligt implementerats. Den första versionen var mycket modulär, vilket gjorde det enkelt att byta ut delar av programmet mot nya när detta önskades. Nackdelen visade sig under utvecklingens gång vara att när systemet blev komplext, med många olika funktionsblock, så blev exekveringen långsam. Totalt gjordes tre omskrivningar som skiljde sig markant från tidigare versioner och den slutgiltiga versionen var den minst modulära av dessa. Antalet funktionsblock var som mest 24 stycken och i den slutgiltiga versionen endast fyra (4).

Det första som togs fram var en metod för att avgöra om linjeföljaren låg på linjen och om så var fallet kontrollera om några kursförändringar behövdes. Optokopplarna gavs koordinater i förhållande till mitten av hjulaxeln sedd uppifrån. Sedan testades vilka kurvradier med utgång från origo som skar dessa koordinater. Dessa kurvradier sparades sedan i mjukvaran som en lista där varje kurvradies position i listan motsvarade en optokopplares position räknat från mitten. När en hög signal till exempel registrerades från optokopplare nummer tre (3) på vänster sida rapporterade systemet att kurvradie nummer tre (3) skulle gälla och att kurvan gick åt vänster. Bestämningen av referenshastighet skedde därefter per hjul enligt ekvation (7) och ekvation (8). En nackdel med denna regleringsmodell var att om en kurva registrerades på båda sidorna samtidigt så sänktes hastigheten på båda hjulen och en T-korsning orsakade ett totalstopp. Detta system användes genom alla versioner av mjukvaran om än något modifierat.

$$maxhastighet_{kurva} = \sqrt{r\mu g} \tag{7}$$

Där r är kurvradien, μ är den statiska friktionskoefficienten mot underlaget och g är gravitationskonstanten.

$$hastighet_{hjul} = v_{kurva}\frac{r-w}{r+w} \tag{8}$$

Där v_{kurva} är den önskade hastigheten genom kurvan, r är kurvradien och w är avståndet från linjeföljarens mitt till hjulet.

Den andra större omskrivningen utökade systemet med seriell kommunikation via UART. Detta möjliggjorde läsning av data från enheten och även inmatning av data under tiden som systemet kördes. Det blev därmed möjligt att felsöka mycket snabbare än tidigare. Data skickades mellan en PC och linjeföljaren med hjälp av en seriell USB-adapter och ett Python-script. Funktionaliteten avvecklades i en senare version då all nödvändig datainsamling, som linjeposition och hastighet, fungerade som tänkt. En stor faktor till beslutet var att Simulink skickade och tog emot data varje exekveringsloop och detta i sin tur ledde till att en ansenlig del av exekveringstiden bestod av att skicka eller ta emot långsam, seriell data. Eftersom funktionen ej längre ansågs nödvändig, om än praktiskt, föreföll det lämpligare att ta bort den helt istället för att utveckla en effektivare implementation.

En annan förbättring var kraftigt utökad lagring av information via minnesblock. Detta gjorde det enkelt att spara data mellan exekveringslooparna och förenklade delning av information mellan flera

olika Simulink-block. Den stora mängden datapunkter från programflödet tillsammans med den seriella kommunikationen underlättade felsökning genom att åskådliggöra var i programflödet som fel uppstått. Den slutgiltiga programvaran hade i och med avvecklandet av seriell kommunikation inte längre någon nytta av denna stora mängd datapunkter och den fjärde versionen hade cirka en tredjedel så många minnesblock som den andra (se Tabell 1).

De block för PID-kontroll som följde med Simulink hade stora begränsningar gällande användarvänlighet då alla värden var statiska och endast kunde ändras manuellt mellan kompileringarna genom att öppna blocket och fylla i nya värden. Implementationen av en egendesignad PID-kontroller med externa insignaler möjliggjorde inte bara enkla förändringar genom konstantblock som sedan kopplades till båda PID-kontrollerna samtidigt, utan även mjukvarustyrda kontrollerkonstanter som kunde bestämmas genom såväl funktionsblock som läsning av värden från inkommande seriell data. Den nya implementationen utnyttjade de överföringsfunktioner för z-transformen som Simulink tillhandahåller. I version fyra (4) byttes denna ut mot en tvåkanalsversion, där ett funktionsblock hanterade regleringen av båda hjulen. Denna slutgiltiga implementering gick ifrån z-transformen och nyttjade istället enkla summor och skillnader mellan datapunkter i en samling av tidigare uppmätta fel. Integralen beräknades som summan av de tidigare uppmätta felen, se ekvation (9), och derivatan som skillnaden mellan det senast uppmätta och det föregående felet, se ekvation (10).

$$\int e(t)dt \cong \sum E_s T_s \tag{9}$$

Där $e(t)$ är skillnaden mellan systemets är- och börvärde vid en viss tidpunkt, E_s är en samling med den senaste såväl som tidigare uppmätta skillnader mellan systemets är- och börvärde och T_s är tiden mellan att två samplingar påbörjas.

$$e'(t) = E_n - E_{n-1} \tag{10}$$

Där $e'(t)$ är derivatan av skillnaden mellan systemets är- och börvärde vid en viss tidpunkt, E är en samling med den senaste såväl som tidigare uppmätta skillnader mellan systemets är- och börvärde och n är index för det senaste mätvärdet.

Version tre (3) av mjukvaran utökade funktionaliteten med fungerande hastighetsmätning. Optokopplare riktade mot hjulens insidor reagerade på deras svart- och vitrandiga mönster. En ADC vidarebefordrade mätvärdena till mjukvaran där dessa samplingar sparades. Sparade mätvärden kasserades allteftersom och erbjöd alltså endast en tidsbegränsad historik bestämd av antalet sparade samplingar. Antalet förändringar mellan hög och låg signal och totaltiden för alla samplingar användes sedan för att beräkna tickfrekvensen och följaktligen hastigheten enligt ekvation (11) och ekvation (12). En stor nackdel med denna mätmetod var att mätvärdet alltid utgjordes av ett medelvärde för den gångna tidsperioden, vilket inte nödvändigtvis korrekt reflekterade den hastighet som faktiskt hölls då förändringarna var plötsliga. Reaktionstiden för reglersystemet blev alltså lidande.

$$frekvens = \frac{c}{2nt_s} \tag{11}$$

Där c är antalet nivåförändringar bland de sparade värdena, n är antalet sparade värden och t_s är tidsperioden som ett sparat värde upptar.

$$hastighet = \frac{f\pi d}{p} \tag{12}$$

Där f är frekvensen, d är hjuldiametern och p är antalet pulser per varv.

Version fyra (4) av mjukvaran frångick medelvärdesmetoden. Den nya implementationen mätte istället tiden mellan två signalförändringar från takometrarna, antingen hög-låg-hög eller låg-hög-låg, och beräknade sedan hastigheten enligt ekvation (13) och ekvation (12). Den uppmätta hastigheten blev således mer representativ för det faktiskt rådande tillståndet då endast de senaste mätvärdena togs i beaktning vid beräkningen.

$$frekvens = \frac{1}{2nt_s} \tag{13}$$

Där n är antalet sparade värden mellan två nivåförändringar och t_s är tidsperioden som ett sparat värde upptar.

Fjärde (4:e) omskrivningen medförde en kraftigt förenklad programarkitektur. Många funktioner, som till exempel automatisk kalibrering av givare, togs bort eftersom dessa visats sig ej vara nödvändiga och endast tog upp extra exekveringstid. Många funktionsblock slogs samman och koden blev samlad i större enheter med tydligare exekveringsföljd. Modulariteten blev lidande och ytterligare förändringar blev alltså svårare att implementera utan att först ha en ordentlig överblick över blocket som helhet. För att ytterligare förenkla och tydliggöra exekveringsföljden samlades all inhämtning av givarvärden på en ADC, istället för att som tidigare placera linjegivarna och takometrarna på olika ADC:er.

Version	**Totalt antal block**	**Antal funktions-block**	**Minnesblock Total** Total: S/R/W*	**Portar** in : ut
1	128	10	6: 2/2/2	35 : 20
2	455	24	102: 34/34/34	135 : 86
3	314	24	32: 9/11/12	111 : 50
4	176	4	34: 11/12/11	74 : 35

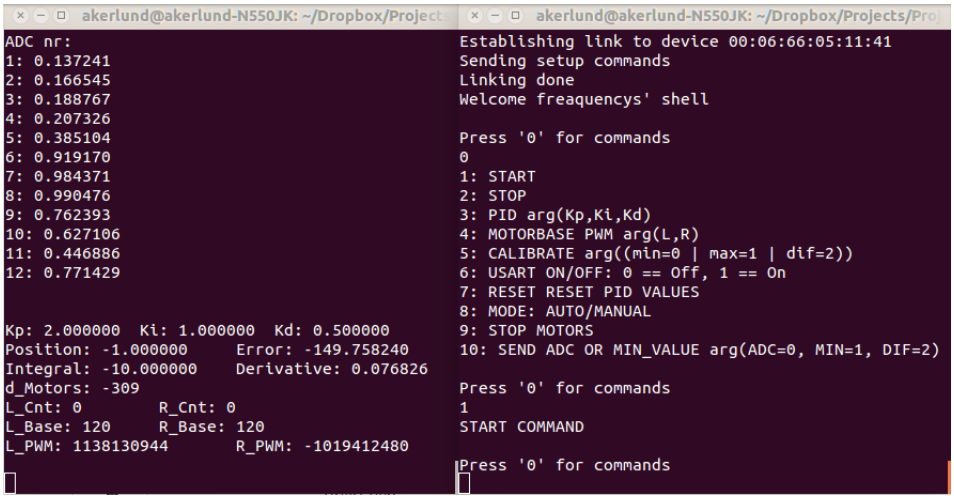
Tabell 1: Versionsstatistik för styrsystem skrivet i Simulink

* S – antal allokerade minnesblock, R – antal minnesläsande block. W – antal minnesskrivande block

3.5 Grafiskt användargränssnitt

För att underlätta kommunikationen med roboten programmerades ett grafiskt användargränssnitt för användning tillsammans med styrsystemet som skrivits i C. Tack vare detta så kunde nya värden för till exempel PID-regulatorn sättas under körning. Läsning av data från roboten gjorde också felsökning lättare då mätvärden presenterades grafiskt. Tillsammans med blåtandsmodulen kunde detta ske utan kabelanslutning och mycket tid sparades in.

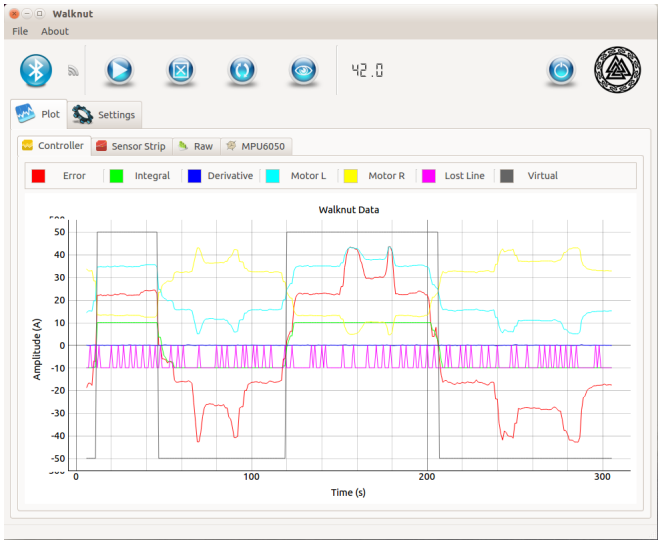
Ett användargränssnitt implementerades först som ett relativt simpelt script i Python med två (2) terminalfönster, vilket kan ses i figur 26. I det ena fönstret presenterades datan och i det andra kunde användaren välja mellan olika kommandon som skickade med inskrivna argument till roboten. De två terminalerna kommunicerade via en Python event-server som skickade data mellan klienterna.



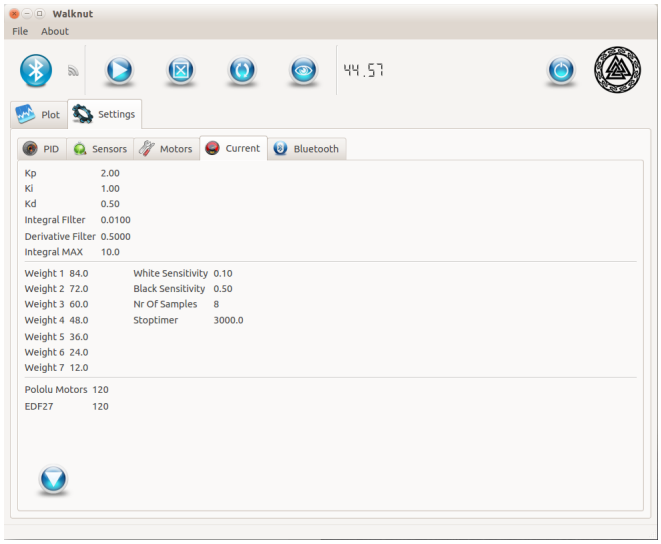
Figur 26: Python event-server, terminalfönster

Efter en del tester med den första linjeföljaren framgick det att det skulle vara av intresse att få datan plottad i diagram för att därefter kunna bättre avgöra varför roboten ibland betedde på ett visst sätt, till exempel varför den tog vissa svängar på ett specifikt sätt eller hur datan såg ut precis innan en avkörning. I följande version implementerades därför en graf för utsignalen från regulatorn (figur 27) samt inställningarna för regulatorparametrarna (figur 28) och information om linjens förmodade position ("error"i figur 26). Även värdena för ADC:n kunde plottas i en graf och de obehandlade värdena som kom in kunde presenteras som vanlig text för att möjliggöra verifiering av att Pythonmodulen pyqtgraph använde datan rätt i de plottade graferna. För extra tydlighet så gavs användaren också möjligheten att välja färg på respektive graf.

Knappar för start och stopp samt knappar för nollställning av PID:en och kalibrering av svart och vit yta implementerades, precis som möjligheten att ändra parametrar som PID-konstanterna (K_p, K_i och K_d), motorernas bashastighet, givarnas känslighet, hur många samplingar som går på en avläsning, hur lång tid som ska gå innan roboten stannar efter att den tappat linjen och sökning efter blåtandsmoduler. Sist lades det också till en graf för värdena från accelerometer och gyro (MPU6050) samt en timer för att kunna mäta tiden ett varv runt banan tar.



Figur 27: Visar data som roboten skickar till PC via blåtand



Figur 28: Visar hur roboten är konfigurerad

4 Teori

Nedan följer den teori som används som grund i projektet. PID-regulator för reglering av linjeföljaren, resonemang kring friktionsekvationer och hjulens påverkan vid linjeföljning samt interpolering för positionering av linjen.

4.1 Allmänt om regulatorer

P-reglering:

$$u(t)_p = K_p e(t) + u_0 \tag{14}$$

$$e(t) = r(t) - y(t) \tag{15}$$

Den mest simpla regulatorn av de alla är p-regulatorn, se ekvation (14), vilken verkar genom att en skalning av reglerfelet, se ekvation (15), adderas med u_0 vilka tillsammans bildar styrsignalen. Konstanten u_0 är den utsignal man får i systemet vid $e(t) = 0$, dvs inget reglerfel [16]. Börvärdet är det värde som systemet önskas anta. Ärvärdet är det värde som systemet för närvarande antar. $e(t)$ betecknar reglerfelet, $r(t)$ referensen och $y(t)$ regulatorns styrsignal.

Utsignalen närmar sig börvärdet eftersom insignalen är en negativ återkoppling av utsignalen [17]. P-regleringen motverkar störningar och tar utsignalen nära börvärdet, men kan i allmänhet ej helt motverka reglerfel varaktigt.

PI-reglering:

$$u(t)_{PI} = K_p e(t) + u_0 + K_I \int_0^t e(\tau) d\tau \tag{16}$$

En integral över reglerfelet läggs till styrsignalen så att medans reglerfel existerar ökas styrsignalen proportionellt mot integralen av reglerfelet, se ekvation (16). Med integrerande verkan kommer felet gå mot noll om systemet är stabilt [16].

En nackdel med integrerande reglering är att när reglerfelet blir noll kommer integraldelen av regulatorn fortfarande ge ett bidrag till styrsignalen och därmed kommer utsignalen öka ytterligare. Med PI-reglering nås börvärdet i regel snabbare än vid bara P-reglering men kan ge upphov till översvängning och ett insvängningsförlopp mot börvärdet. Man kan med för stora värden på K_I och/eller K_P göra systemet instabilt och då kommer reglersystemet i självsvängning.

PD-reglering:

$$u(t)_{PD} = K_p e(t) + u_0 + K_D \frac{d}{dt}(e(t)) \tag{17}$$

Man kan även lägga till en skalning av reglerfelets derivata till styrsignalen för att få en deriverande verkan på utignalen, se ekvation (17). D-reglering kan användas för att minska instabiliteten i ett reglersystem. Om mätdatan är brusig blir det stora problem med införandet av en derivata då denna blir stor vid snabba förändringar [16].

PID-reglering:

$$u(t)_{PID} = K_p e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt}(e(t)) \tag{18}$$

PID-reglering är en kombination av proportionell, integrerande och deriverande verkan, se ekvation (18), med vilken man hoppas kunna styra utsignalen tillräckligt snabbt, med tillräckligt liten översvängning och reglerfel och samtidigt hålla systemet stabilt.

4.2 Mätmetod friktionskoefficient

För hårda och torra ytor gäller enligt Amontons lag, se ekvation (19), att friktionskraften är direkt proportionell mot normalkraften. Leonardo da Vinci upptäckte detta samband under 1400-talet men resultaten föll i glömska och återupptäcktes ca 200 år senare av Guillaume Amonton [18]. Statiska friktionskoefficienten, ofta betecknad med den grekiska bokstaven μ_s, är den koefficient som normalkraften skalas med för att få friktionskraften. Detta är en approximation som fungerar i det flesta system [19].

$$F_f = \mu_s F_N \tag{19}$$

Enligt Da Vinci, givet två fixa hårda ytor, är normalkraften det enda som påverkar friktionskoefficienten, se ekvation (20) [18].

$$\mu_s = \mu_s(N) \tag{20}$$

Friktionskoefficienten kan skrivas som kvoten mellan friktionskraften dividerat med normalkraften som verkar på kroppen, se ekvation (21). När man applicerar en kraft med den amplitud att kroppen precis börjar glida kan man approximera denna kraft med friktionskraften. Om man mäter den kraften med dynamometer och massan hos kroppen är känd kan man räkna fram friktionskoefficienten.

$$\mu_s = \frac{F_f}{F_N} \tag{21}$$

Ett annat sätt att empiriskt få fram friktionskoefficienten är att placera kroppen på ett slutande plan vilket man kan ändra lutningen på. Öka vinkeln till den där kroppen precis börjar glida. Friktionskoefficienten fås nu direkt ur tangens för denna vinkel, se ekvation (22) [19].

$$\mu_s = \frac{F_f}{F_N} = \tan(\varphi) \tag{22}$$

Då hjul av gummi eller polyuretan anses som mjuka material kan inte Amontons lagar appliceras. Statiska friktionskoefficienten kan inte antas vara oberoende av kontaktytan med underlaget och kan inte antas vara linjärt beroende av normalkraften. Om robotens vikt inte förändras, och underlaget som mätningen utförs på är samma underlag som under tävlingen, bör resultatet av mätningen ge en god skattning av statiska friktionskoefficienten hos systemet på tävlingsdagen.

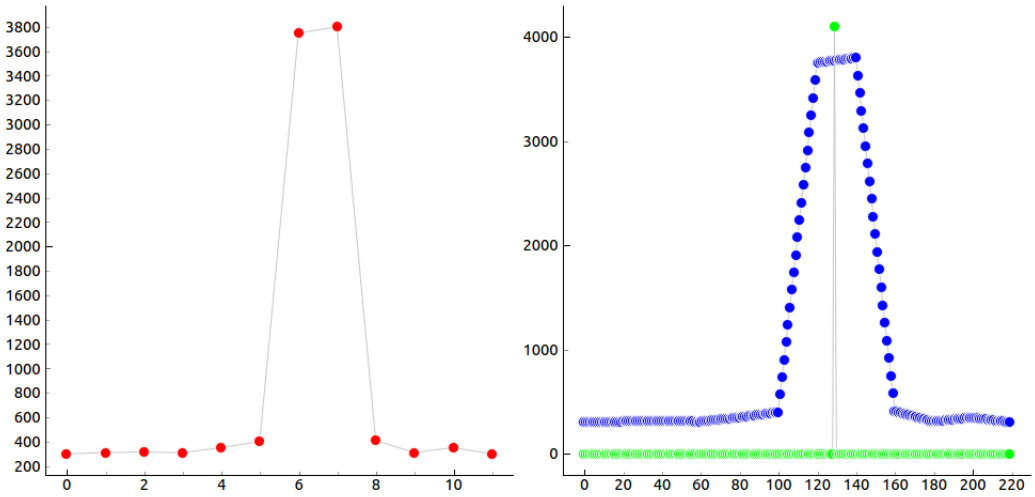
Mätning bör utföras på samma underlag som Robot-SM planen är tillverkad av då friktionskoefficienten är beroende av både hjulen yta och underlagets yta. Vikten på roboten påverkar normalkraften, så vikten bör vara samma som vid tävling. Mätning avser att jämföra de köpta hjulen med de egentillverkade för att undersöka om de egentillverkade hjul ökar robotens prestanda eller ej. Simulinkprogrammet använder sig av friktionskoefficienten för att beräkna den maximala fart som kan hållas i kurvorna utan att slira. De egentillverkade hjulen tillverkades av polyuretan som för de flesta material har högre friktionskoefficient än gummi. De egentillverkade hjulen var även bredare och hade en större diameter än de köpta hjulen.

4.3 Interpolering för positionering av linjen

Genom interpolering av givarvärdena kan linjens position bestämmas med högre upplösning än enbart diskret avläsning av värdena. Virtuella givare skapas mellan de riktiga, desto fler virtuella givare, desto noggrannare positionsvärde möjliggörs. Varje virtuell givare tilldelas ett värde beroende på vilken position den har mellan de riktiga givarna och vilket värde de har uppmätt enligt formeln nedan, se ekvation (23).

$$V_i = \frac{(s_v \cdot (n-j) + s_{v+1} \cdot j)}{n} \tag{23}$$

Där n kännetecknar antalet virtuella givare mellan två riktiga givare, s_v och s_{v+1}, och j den virtuella givarens position mellan de två riktiga givarna.



Figur 29: Exempel på interpolering

Alla givarvärden adderas till en totalsumma. Därefter adderas givarvärdena stegvis till en partiell summa och när denna når halva totalsumman har linjens position skattats (figur 29). Detta gör det möjligt att finna linjens position oavsett hur många givare som signalerar dess närvaro, till exempel då linjens vinkel mot givarstrippen ligger när 0 eller 180 grader och den därför täcker fler än två givare. En matematisk formulering för summan av de virtuella givarna kan ses i ekvation (24).

$$S = \sum_{v=0}^{\#givare-1} \sum_{j=0}^{n-1} \frac{(s_v \cdot (n-j) + s_{v+1} \cdot j)}{n} \tag{24}$$

Om 20 stycken virtuella givare används mellan tolv (12) stycken verkliga fås ett spann om 20·11 = 220 virtuella positioner där mitten blir 110,5. För varje givarvärde som adderas till den partiella summan så räknas ett index upp. När halva totalsumman uppnåtts motsvarar det senast adderade indexet positionen. Skulle det sista indexet bli ett (1) vet man att linjen befinner sig längst till vänster och om det blir 220 befinner sig linjen längst till höger. Ett problem med att ha ett jämt antal givare är att den exakta mitten aldrig går att hitta eftersom den, precis som i exemplet, blir ett flyttal och index är heltal. Om linjen befann sig mellan position 110 och position 111 så fås alltid 111 som resultat. För att lösa detta så kan man använda ett ojämnt antal givare eller öka antalet virtuella givare för att minska felet.

5 Resultat

Målet att konstruera en robot som autonomt följde en linje och överensstämde med reglerna för linjeföljning i Robot-SM nåddes. Eftersom roboten Walknuts konstruktion inte slutfördes i tid fick prototypen Rulle istället delta i tävlingen. Rulles regulatorparametrar justerade på plats och tid för finjusteringar fanns alltså inte. Rulle tävlade under pseudonymen "Böpotrex2000 v9.4" och tog sig runt banan på 30.49 sekunder. Detta resulterade i en sjätte (6:e) placering bland de sju (7) tävlande. Den vinnande roboten, "Lilla", bröt den populära trenden med givarstrip och hade istället enbart två (2) IR-givare av större modell. Det i sammanhanget lilla formatet möjliggjorde således skarpa svängar i hög fart. På andra plats hamnade en robot med EDF av samma modell som monterats på Walknut.

Styrsystemet som skrivits i C presterade i grunden bra. DMA utnyttjades effektivt för att frigöra beräkningskraft från processorn. Denna kunde då ägna mer resurser åt regleringen av roboten. Mikrokontrollern hade troligtvis tillräckligt med beräkningskraft för att klara av uppgiften även utan DMA och utrymme fanns antagligen för mer avancerade funktioner.

Prototypen var byggd utan större optimeringar gällande layout och kabeldragning och problem med kabeltrassel minskade överskådligheten och sänkte pålitligheten. Det var svårt att se hur kablar var kopplade och ibland även svårt att upptäcka när kablar lossnat. Den var både otymplig att hantera och ömtålig. Walknut lyckades lösa denna problematik genom smartare placering av komponenter och anslutningar, samtidigt som vikten reducerades.

För att ytterligare förbättra kurvtagningsförmågan utrustades Walknut med ett hål för att möjliggöra installation av en EDF. Denna fläkt installerades, men hann dock aldrig testas och därför inte heller utvärderas.

Hjul tillverkades i slutet av projektet som en uppgradering till Walknut. Fälgar ritades upp i CATIA och designades med 25 mm:s innerdiameter. Fälgarna belades med polyuretan vilket gav en total hjuldiameter på 40 mm och en bredd på 13 mm. Dessa var betydligt bredare än de köpta hjulen som hade bredden 6 mm och diametern 32 mm.

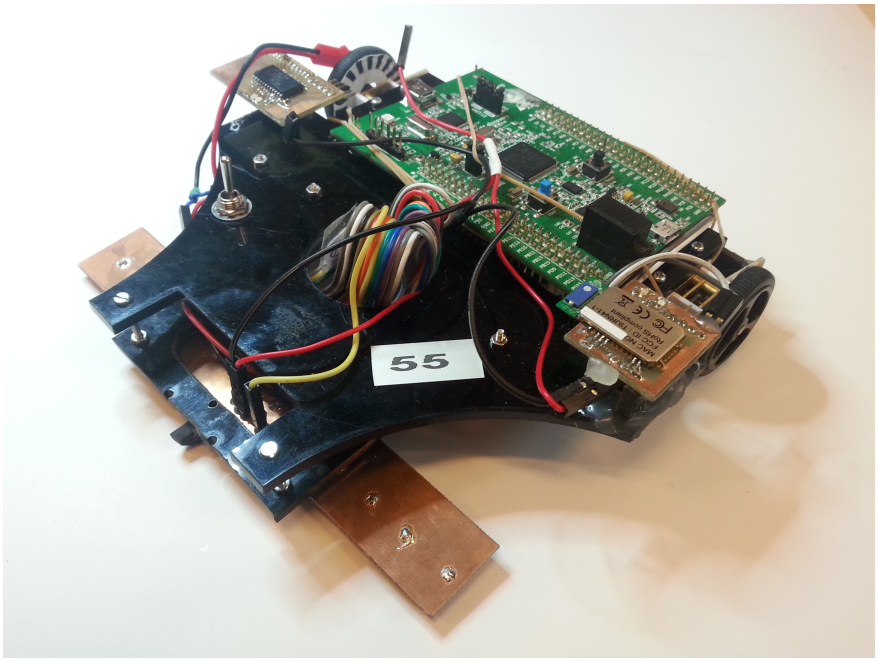
Mätning av den statiska friktionskoefficienten genomfördes på de två olika hjulparen, båda monterade på Walknut, med hjälp av ett sluttande plan. Som underlag användes samma bana som linjeföljarna testkördes på. Det sluttande planet fungerde som hypotenusa i en rätvinklig triangel vars kateter mättes. Ena kateten beteckandes "Höjd" och den andra "Längd". Walknut placerades med färdriktining vinkelrätt mot planets lutning. Planets vinkel ökades tills Walknut precis började glida. Kateterna mättes med en 60 centimeters stållinjal. Sedan beräknades $\tan(\varphi)$ samt φ med hjälp av mätvärdena. Resultatet kan ses i Tabell 2. Slutsatsen som dras är att de egentillverkade hjulen har betydligt bättre grepp än de köpta och därmed ökar prestandan. Dock bör det noteras att de tillverkade hjulen var nytillverkade och hade en aningen klibbig yta. Denna yta kommer möjligen ansamla smuts från banan vilket då försämrar greppet. Båda hjulparen torkades av innan, men sköljdes inte av och var alltså inte helt fria från damm. Medelvärdet av de tre mätningarna på respektive hjul ger statiska friktionskoefficienten för de köpa hjulen $\mu_s = 0.95$ och för de egentillverkade $\mu_s = 1.7$, vilket innebär att de egentillverkade har 79% högre friktionskoefficient.

Hjultyp	**Höjd [mm]**	**Längd [mm]**	$\mu_s = \tan(\varphi)$	φ [°]
Köpta	36	35	1.0	46
Köpta	33	35	0.94	43
Köpta	31	35	0.90	42
Tillverkade	60	35	1.7	60
Tillverkade	55	27	2.0	64
Tillverkade	53	35	1.5	58

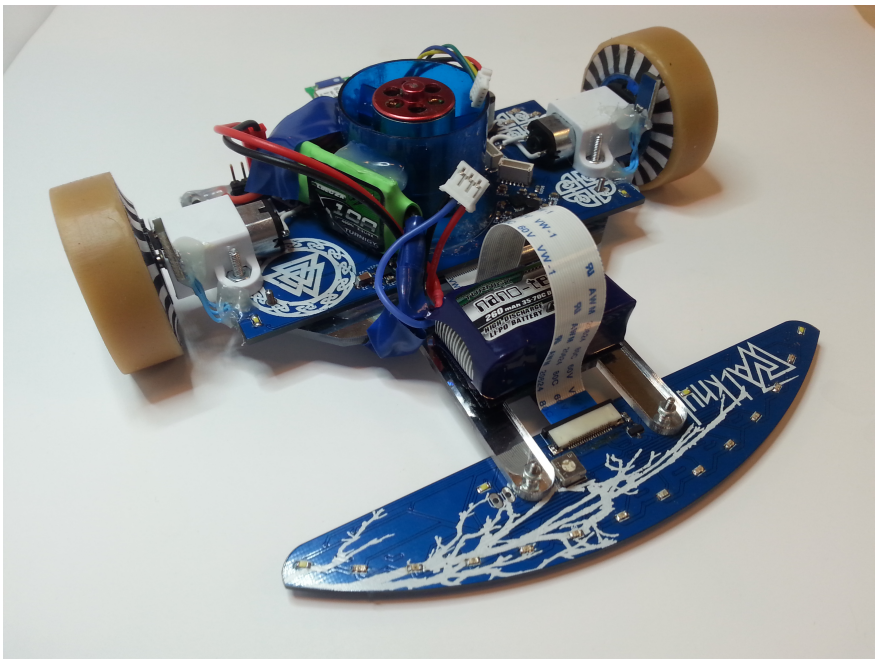
Tabell 2: Mätvärden från friktionsmätning med hjälp av ett sluttande plan

Den första prototypen Rulle (figur 30) användes sedan som testplattform och förbättrades iterativt och efter den skapades den slutgiltiga linjeföljaren Walknut (figur 31). Walknut baserades på ett

specialbeställt mönsterkort på vilket alla komponenter monterades fast på. Det färdiga kretskortet agerade alltså som en del av chassit. En gaffel av akrylplast förbinder givarstrippen med resten av robotkroppen.



Figur 30: Testplatformen Rulle i sin slutgiltiga form



Figur 31: Den slutgiltiga linjeföljaren Walknut

6 Diskussion

Under projektet togs många beslut innan den slutgiltiga linjeföljaren blev färdigställd. Här följer några av de frågeställningar som genererats samt diskussion kring resultatet.

6.1 Frågeställningar som togs upp

Några av de aspekter som resonerades kring under utvecklingen av robotarna.

6.1.1 Vilka fördelar finns med att tillverka en egen linjeföljare mot att köpa en färdigbyggd?

Att konstruera en robot mer eller mindre från grunden ger möjlighet till förståelse för både hårdvara och mjukvara, förståelsen blir inte lika omfattande om man utgår från en färdig robot. En av de större fördelarna med att konstruera från grunden är att designen kan optimeras enligt egna preferenser samt funktionalitet som kommersiella robotar inte har kan implementeras.

6.1.2 Hur påverkar givarstrippens avstånd från robotkroppen prestandan?

Då givarstrippen sitter långt framför kroppen på roboten upptäcks kurvor långt innan roboten anländer till kurvan vilket medför att styrsignalerna kan bestämmas långt i förväg. En nackdel med att ha givarstrippen långt framför roboten är att skarpare kurvor är svårare att följa eftersom roboten kommer skära i kurvorna. Om hastigheten är för hög kan givarna helt passera linjen vilket gör att linjen tappas.

6.1.3 Varför valdes optokopplare istället för kamera?

Optokopplare ansågs vara en enklare lösning vilket passar till en första prototyp. Prototypens mikroprocessor har möjlighet att ansluta tilläggskort som öppnar upp för kamera. Att implementera bildbehandlingsalgoritmer till kamera ansågs för tidskrävande. Om optokopplare hade visat dålig prestanda hade möjligheten till kamera eller andra typer av givare till den slutgiltiga linjeföljaren undersökts grundligare.

6.1.4 Varför tillverka egna hjul?

Friktionen mot underlaget är en betydande begränsande faktor för att klara av skarpa svängar i hög hastighet utan att tappa greppet. Att tillverka hjul i polyuretan ansågs vara en relativt liten uppgift som bör förbättra linjeföljarens prestanda. Vid egentillverkning kan hjulens bredd och diameter anpassas efter designen hos övriga delar av roboten. Att tillverka hjul med större diameter än de köpa hjulen ökar linjeföljarens höjd vilket öppnar upp för att montera komponenter på robotens undersida. Dock höjs tyngpunkten om hjuldiametern ökar och komponenter ej flyttas. Efter mätningar kan det konstateras att egentillverkade hjul förbättrar prestandan.

6.1.5 Hur fungerade det med att iterativt skapa prototyper?

Enligt planeringsrapporten skulle tre stycken diskreta prototyper av linjeföljaren tas fram. Endast en prototyp blev färdigställd helt från grunden och under projektets gång användes denna som testplattform för att sedan iterativt testa, förbättra och byta ut enskilda komponenter. Detta ledde till att arbetet kunde fortsätta på ett effektivt sätt och roboten förbättras kontinuerligt vilket resulterade i en slutgiltig linjeföljare med bättre prestanda än prototypen.

Strukturen på arbetet i projektet blev därför först inte som planerats. Istället för att bygga en hel linjeföljare från grunden varje gång något behövdes förbättras, kunde istället valda delar bytas ut eller förbättras. På samma sätt kunde delar som fungerade bra bevaras och inte bytas ut i onödan. Ur en ekonomisk synvinkel är det även fördelaktigt då inte flera uppsättningar av varje komponent behövde införskaffas.

6.1.6 Funkade det bra med testplattform för slutgiltiga följaren?

En fördel som gick förlorad var att med flera robotar hade båda programmeringsgrupperna kunna testa sin kod på hårdvaran simultant. Korta testkörningar gjorde att inga krockar förekom. Det mest tidskrävande testet på hårdvara utgörs av intrimning av regulatorparametrar vilket endast C-kod gruppen utförde då det var det program som skulle tävla på Robot-SM.

6.1.7 Hur fungerade linjeavläsningen?

Avläsningen av linjen med interpolering är en intressant metod för att öka upplösningen på linjeavläsningen. Användbarheten av interpolering för en enkel linjeföljare är dock diskutabel. Skillnaden mellan de skattade värdena och den mycket enklare implementeringen då endast de riktiga givarna och enkla gränsvärden användes för att identifiera linjens position. Den höga upplösningen behövs troligen inte för denna typ av uppgift, men kan säkerligen vara mycket användbar för andra typer av mätvärden. Att använda tolv (12) stycken optokopplare med 12 mm avstånd mellan dessa fungerade tillfredsställande. Rulle tappade linjen flera gånger, men endast i skarpa kurvor. Detta kan bero på att den givarstrip som användes var helt rak och därför kan hela givarstrippen snabbt passerade linjen då hastigheten var hög. I mjukvaran som skrivits i C så användes en funktion som noterade vilken sida av givarstrippen som linjen senast befunnit sig. Tack vare denna funktionalitet så kunde roboten ofta hitta tillbaka till linjen vid dessa avåkningar och funktionen ger alltså ett mycket gott resultat trots sin enkelhet.

6.1.8 Vad hade kunnat gjorts om mer tid givits?

På Robot-SM är det tillåtet att köra banan flertalet gånger. Om man har en IMU, Inertial Measurment Unit, ombord på roboten kan banans form skattas nästa gång roboten kör ett varv på banan. Då roboten har en uppfattning om hur banan ser ut kan den anpassa farten till att vara högre på raksträckor och minska när den närmar sig kurvor. Att implementera denna funktionalitet utelämnades på grund av tidsbrist.

Ultraljudsgivare eller annan avståndsmätare hade behövts för att ställa upp i avancerad linjeföljning på Robot-SM. Även fler beslutsalgoritmer hade behövts som, på grund av tidsbrist, inte var realistiskt att implementera.

I väntan på en körduglig Walknut så avbröts planerna på fler regulatorer än den redan implementerade för vinkel. En PID-kontroller för acceleration hade kunnat implementerats i mån av tid. Denna skulle ha hjälpt till att stabilisera roboten när den återfunnit linjen efter en hastig sväng. Utan denna extra regulator så överreagerar roboten med resultatet att mitten av givarstrippen passerar över linjen och orsakar ytterligare en positionskorrigering, fast i motsatt riktning. En extra regulator för acceleration skulle alltså möjliggöra en aggressivare körstil utan negativ inverkan på precisionen som systemet erbjuder. Ytterligare en regulator, då för fart, skulle kunna ändra maxhastigheten beroende på felets storlek och på så sätt tillåta högre hastigheter på raksträckor för att sedan sänka denna i kurvor. Denna typ av funktionalitet implementerades i det Simulinkbaserade styrsystemet men i brist på testdata kan funktionsdugligheten av denna implementation ej utvärderas. Ytterligare en möjlig regulatorimplementation skulle kunna kontrollera en EDF så att den inte roterar konstant, utan varvar upp när felet blir stort för att på så sätt erbjuda extra grepp i kurvorna. Hurvida det är möjligt eller ej beror på hur lång tid det tar innan EDF:n varvat upp.

6.1.9 C och Simulink, hur blev resultatet av C-implementeringen jämfört med Simulinks autogenererade kod?

I Robot-SM användes det styrsystem som programmerats i C. Programmet hade högre funktionsduglighet och mer avancerade funktioner än sin Simulink-motsvarighet. Blåtandskommunikationen med det grafiska användargränssnittet för C-programmet möjliggjorde enkel konfiguration av parametervärden. När något värde betedde sig på ett orimligt sätt tydliggjordes detta i en graf och lämpliga åtgärder kunde vidtas. Resultatet blev ett väl fungerande reglersystem som kunde utföra

den uppgift den konstruerats för. Simulink-systemet lyckades aldrig uppnå samma prestanda. Men varför blev det så?

Sammanfattat så kan man säga att reglersystemsimplemtationen i C helt enkelt blev bättre, men det intressanta är varför. Vid en första anblick så verkar Simulink vara ett bra val för den som snabbt vill komma igång med ett projekt utan att behöva djupdyka i hårdvarudokumentationen.

Simulinkmodeller är relativt enkelt att skapa då kodning utförs genom att dra block från ett blockbibliotek, koppla ihop blockens in- och utgångar genom att klicka och dra, samt ställa in blockens parametrar genom att klicka och fylla i värdena. I implementationen ingick flertalet MATLAB-block där blockets funktion definieras med hjälp av MATLAB-kod, vilket gruppen har god kännedom om från tidigare kurser. Felsökning av MATLAB-blocken är därmed enkel att utföra. Felsökning av programmet som helhet är dock svårare att utföra.

En fördel med Simulink är att användaren en god överblick över systemet genom grafiska representationer av informationsflödet mellan funktionsblocken. Minimalt med arbete behöver läggas för att få tillgång till grundläggande hårdvarufunktioner. Nackdelen är att koden som dessa block representerar är mer eller mindre dolt för användaren och inte tillräckligt dokumenterad. Det som visat sig svårare är att felsöka programmet som helhet då den simpla kodningen ger lite insyn i hur den genererade koden faktiskt ser ut. Vad som pågår i de block som ej definieras av användaren är svårare att kontrollera då den enda interaktionen som går att utföra är att definiera parametrar och undersöka utsignalen. UART lyckades implementeras och användes till att avläsa enstaka signaler i systemet för att kontrollera om de antar rimliga värden eller ej. Då UART kopplades till en seriell adapter till PC kunde data plockas ut under körning om roboten hålls över en vit testplatta med svart linje på. Dock lyckades inte blåtandsmodulen implementeras så körning på en faktisk bana samtidigt som data tas ut lyckades aldrig.

C är ett språk som ligger väldigt nära hårdvaran och därför krävs en god förståelse för hårdvaran som används samt programmeraren måste vara väl införstådd med vad som krävs för att uppnå önskat resultat. Bland annat måste man i C initiera all hårvara. När koden färdigställts framgår det tydligt vad programmet gör. I Simulink kan ett system se fungerande ut, enligt den information som finns tillgänglig, men på grund av de implementeringar som ligger utom synhåll för användaren kan systemet bete sig annorlunda än vad som var tänkt.

Resultatet blir att något som borde vara lättare att uppnå i Simulink än i C, istället blir svårare när den som skriver programmet tvingas försöka gissa sig till vad som händer bakom kulisserna. När det fungerar som tänkt sparas dock mycket arbetstid, tyvärr var inte så fallet i detta projekt där mycket av arbetstiden gick till att förstå hur Simulinkprogrammet faktiskt agerade.

En möjlig förklaring till den otydliga och ibland frånvarande dokumentationen kan vara att det bibliotek som användes inte var hårdvarutillverkarens eget, utan ett tredjepartsbibliotek. Utan att testa båda biblioteken förblir detta dock vara ren spekulation. Gruppen som arbetade med C kunde, istället för att lägga tid på felsökning utan dokumentation, fokusera på att implementera mer avancerade funktioner. Bland dessa fanns den trådlösa kommunikationen och det grafiska användargränssnittet som ytterligare snabbade upp utvecklingstakten.

6.2 Utvärdering av projektet

Tidsplanen frångicks i slutet av projektet på grund av problem med testning av nya motordrivare till vilka PCB skulle beställas. Därför deltog inte Walknut i Robot-SM. Ej heller testkörning med EDF förverkligades. Men prototypen Rulle placerade sig på plats sex (6) i tävlingen.

Den simpla lösningen för hastighetsmätning genom att använda optokopplare mot hjulen visade sig svår att mjukvarumässigt implementera i Simulink. Även problem med de beställda mönsterkorten för Walknut som tog längre tid att felsöka och lösa problemen än förväntat.

Det finns accelerometrar som hade kunnat implementeras för att skatta robotens hastighet, istället

för att mäta hjulens hastighet. Det angreppssättet hade möjligtvis gett ett bättre resultat, men undersöktes inte tillräckligt för att kandidera som alternativ till optokopplarlösningen.

Det finns andra motordrivare som även kan mäta strömmen som motorerna kräver. Med denna information kan mikrokontrollern sedan räkna ut robotens hastighet, givet att en modell finns som omvandlar uppmätt ström till vridmoment. Systemanalys hade i större mån kunnat utföras och därmed hade en bas för regulatorparametrar kunnat etableras vilket hade förenklat intrimning av regulatorn.

Att få roboten att varaktigt följa linjen var ett tidskrävande moment då parametrarna till regulatorn krävde ett flertal justeringar med testkörningar emellan. Den seriella kommunikationen underlättade förloppet och visade sig vara väl värt tiden den tog att implementera.

Att lågpassfiltrera integraldelen och derivatadelen i regulatorkoden verkar, utifrån den data som plottats i det grafiska användargränssnittet, vara ett bra designval. Integralverkan kunde tydligt ses växa likt typiska integralkurvor och derivatadelen steg inte lika okontrollerat vid korta störningar, störningar som tidigare har orsakat relativt stora avvikelser i styrsignalen.

Att inte kunna handla mot kvitto visade sig vara en stor nackdel då många av de komponenter som behövdes ej fanns hos Farnell. Små borstade likströmsmotorer och Lipo-batterier är exempel på komponenter som Farnell ej tillhandahöll i tillfredsställande grad.

6.3 Angående hållbar utveckling

En version av linjeföljning förekommer hos självkörande truckar inom industrin som benämns AGV, Automated Guided Vehicle. Självkörande truckar kan ta över ensidigt arbete samt tunga lyft från personalen och därigenom kan förbättra arbetsmiljön.

I små linjeföljare tar datorsystemet en stor del av den totala energiförbrukningen. Energin som förbrukas av en mikrokontroller blir i allt väsentligt värme som avges till omgivningen. I stora linjeföljare som körs med stora elmotorer, vilka ofta har ganska hög verkningsgrad, kan man tänka sig att mikrokontrollens energiförbrukning är mycket liten i jämförelse med elmotorernas. Då dessa har relativt hög verkningsgrad fås en stor andel nettoenergi, exempelvis för förflyttning av roboten eller lyft av gods. Därmed kommer en stor linjeföljare totalt sett ha ganska bra energieffektivitet, i alla fall om man jämför med en liten. Algoritmer för energieffektivitet kan implementeras genom att till exempel undvika onödiga accelerationer och minimera körsträckor. Detta bidrar till en maskin som potentiellt kräver mindre resurser en vad en el- eller diseltruck med mänsklig förare kan åstadkomma.

Lastning kan ske i källarlokaler och liknande, vilket inte är en optimal arbetsplats för en människa på grund av avsaknaden av dagsljus. Alltså kan det vara lämpligt att installera linjeföljande truckar på sådan platser. Ljudnivån kan vara hög i sådana lokaler, speciellt om man använder sig av dieseldrivna truckar, vilket kan förbättras genom eldrivna linjeföljare.

Akrylplasten som användes till chassidelar kan återvinnas och bli nya plastprodukter [20]. Skruvförband, distanser och dylikt läggs tillbaka där de togs ifrån. Komponenterna, som buckomvandlare och de laddningsbara LiPo-batterierna, kan sparas och användas till andra projekt i framtiden. De PCB:n som tillverkats är specialdesignade och svåra att utnyttja till andra projekt. Dessa kan tas om hand via elektronikskrotinsamingen. De små ytmonterade komponenter som använts är svåra att återanvända så dessa följer med kretskorten i elektronikinsamlingen.

Källförtäckning

Texter

[1] N. E. Bouguechal, D. A. Bradley, and R. V. Chaplin, ”Operation of AGVs using information derived from encoded tiles,” Computer Integrated Manufacturing Systems, vol. 6, s. 167-175, 8// 1993.

[2] H. F. DurrantWhyte, ”An autonomous guided vehicle for cargo handling applications,” International Journal of Robotics Research, vol. 15, s. 407-440, Oct 1996.

[3] T. Le-Anh and M. B. M. De Koster, ”A review of design and control of automated guided vehicle systems,” European Journal of Operational Research, vol. 171, s. 1-23, May 2006.

[4] F.Åkerlund. (2014, Oktober 15). Linjeföljare [Online]. Tillgänglig från: `http://www.chalmers.se/sv/institutioner/cse/utbildning/Grundutbildning/kandidatprojekt/Sidor/linjef%C3%B6ljare.aspx`

[5] (2012.09.18), ”Riktlinjer för genomförande och examination av kandidatarbete på Chalmers fr o m VT 2007”,[Online], Tillgänglig från: `https://student.portal.chalmers.se/sv/chalmersstudier/kandidat-och-examensarbete/kandidatarbete/Documents/Riktlinjer%20f%C3%B6r%20genomf%C3%B6rande%20och%20examination%20av%20kandidatarbete%20p%C3%A5%20Chalmers.pdf`

[6] STMicroelectronics, ”STM32F405xx STM32F407xx”, datasheet - production data, Rev 5, Mars 2015

[7] Shenzhen Kinmore Motor Co.,Ltd, ”KM20100122/KM-12FN20-39-06430”, datasheet

[8] Roving Networks, ”RN-41/RN-41-N Class 1 Bluetooth Module”, datasheet, Rev. 3.4, 15 oktober 2012

[9] Silicon Labs, ”CP2102/9 - SINGLE-CHIP USB TO UART BRIDGE”, datasheet, Rev. 11, juni 2014

[10] InvenSense, ”MPU-6000 and MPU6050 Product Specification”, datasheet, Rev. 3.4, 19 augusti 2013

[11] Everlight, ”Technical Data Sheet Optical Interruptor”, ITR8307/TR8 datasheet, Rev. 1.1

[12] STMicroelectronics, ”STP16CP05 - Low voltage 16-bit constant current LED sink driver”, datasheet - production data, Rev. 11, juni 2014

[13] 2015.01.29 [Online], Tillgänglig från: waijung.aimagin.com

[14] H. F. Yu, Y. Ma, T. Gautier, L. Besnard, P. Le Guernic, and J. P. Talpin, ”Polychronous modeling, analysis, verification and simulation for timed software architectures,” Journal of Systems Architecture, vol. 59, s. 1157-1170, Nov 2013.

[15] STMicroelectronics, ”AN4031 Application note Using the STM32F2 and STM32F4 DMA controller”, AN4031 application note, Februari 2014

[16] T. Glad & L. Ljung. ”Principer för återkoppling”, Reglerteknik Grundläggande teori, 4e upplagan, Lund , Sverige, Studentlitteratur, 2006, kap. 1.5, s. 17-21.

[17] B. Lennartsson, ”Enkla regulatorer för störkompensering och referenssignalföljning” , Reglerteknikens grunder, 4e upplagan, Malmö , Sverige, Studentlitteratur, 2008, kap 1.6 , s. 17.

[18] V. L. Popov, ”Coulomb’s Law of Friction”, Contact Mechanics and Friction, New York, Springer-Verlag Berlin Heidelberg, 2010, kapitel 1, s. 1-7

[19] V. L. Popov, ”Coulomb’s Law of Friction”, Contact Mechanics and Friction, New York, Springer-Verlag Berlin Heidelberg, 2010, kapitel 10, s. 133-154

[20] V. Popescu, C. Vasile, M. Brebu, G. L. Popescu, M. Moldovan, C. Prejmerean, et al., ”The characterization of recycled PMMA,” Journal of Alloys and Compounds, vol. 483, s. 432-436, 8/26/ 2009.

Bilder

[Figur 13] Everlight, ”Technical Data Sheet Optical Interruptor”, ITR8307/TR8 datasheet, Rev. 1.6, 2013

[Figur 15] STMicroelectronics, ”STM32F405xx STMF407xx", datasheet - production data, Rev 5, Juni 2013

Appendix I

Regler för Linjeföljning under Robot-SM 2015

Regler för Linjeföljning under Robot-SM

Uppdateringar

- *2014-04-27 - Specifikation på hur långt ifrån hindret roboten får släppa samt återta linjen i avancerad linjeföljning (15 cm före och efter).*
- *2014-02-16 - Punkt 4.2.3 (90°-svängar) är borttaget, eftersom det var öppet för tolkningar. Skarpa vinklar är alltså inte tillåtet på planen. Detta kan dock vara tillåtet i Avancerad linjeföljning.*
- *2014-02-16 - Robotens begränsningar är minskade till 30cm bred och 40cm lång, samt max 3 kg massa.*
- *2014-02-16 - Definition av när en robot anses följa linjen.*

Sammanfattning av reglerna

Linjeföljning går helt enkelt ut på att följa en bana utmärkt av en linje. Reglerna för Linjeföljning bygger delvis på de regler som används i Robotchallenge, Europas största robottävling. Nedan följer reglerna i sin helhet. Använd ert sunda förnuft när ni tolkar reglerna. Om ni är tveksamma på hur ni ska tolka dem rekommenderar vi att ni hör av er snarast så att vi kan räta ut alla frågetecken i god tid. Ni kontaktar oss enklast på e-postadressen info@robotsm.se.

Regler för Linjeföljning samt Avancerad linjeföljning

1. Mål

1. Målet är att roboten ska följa en bana utmärkt av en linje. Roboten måste hela tiden följa linjen. Hela roboten måste börja bakom startmarkeringen och räknas som i mål när någon del av roboten har passerat startmarkeringen efter ett komplett varv runt hela banan.

2. Roboten

1. Roboten skall vara autonom (ingen yttre styrning är tillåten).
2. Roboten får anpassas och ändras under tävlingen så länge reglerna efterföljs. All fysisk förändring av roboten ska rapporteras till tävlingsledningen för godkännande. Tävlingsledningen behöver dock ej underrättas om det endast gäller förändring av mjukvara
3. Roboten får inte vara bredare än 30 cm eller längre än 40 cm.
4. Robotens massa får inte överstiga 3 kg.
5. Roboten måste kunna ses av ett automatiskt tidtagningssystem. Den måste därför på minst ett ställe vara mer 5 cm hög (förslagsvis genom en flagga eller liknande ifall roboten är lågt byggd).
6. Roboten får inte vara byggd för att skada robot, människa eller omgivning.
7. Roboten får inte skrapa eller på annat sätt göra åverkan på eller skada arenan (t.ex. lämna efter klistriga hinnor, färg och liknande).

3. Inspektion

1. Tävlingsledningen har rätt att när som helst kontrollera att roboten uppfyller kraven för tävlingen.

2. Tävlingsledningen har rätt att kräva att roboten modifieras för att uppfylla tävlingskraven.
3. Om problemen inte åtgärdats kan roboten diskvalificeras från tävlingen.

4. Arenan

1. Arenan består av ett vitt underlag med en svart linje ovanpå. Linjen kan antingen vara målad eller tejpad. Arenan ska i bästa fall bestå av målat trä, men kan även bestå av annat material (såsom papper eller plast).
2. Linjen uppfyller följande restriktioner:
    1. Linjen är 19±2mm bred (linjens bredd kan variera inom dessa gränser på olika delar av arenan).
    2. Inga segment av linjen är närmare varandra än 15 cm (mätt från centrum på linjerna).
    3. Arenans kant är aldrig närmare linjen än 15 cm (mätt från centrum på linjen).
    4. Kurvradien är minst 7.5 cm.
3. I början av banan finns ett s.k. startområde. Runt startområdet står portar för tidsmätning. Avstånden mellan dessa portar är minst 30 cm.
4. Det kan förekomma skarvar, ojämnheter och håligheter i arenan. Även linjen kan vara skarvad, med ojämnheter och håligheter som följd. Dessa effekter kommer att minimeras, men kan förkomma.

5. Tävlingen

1. Roboten har 3 minuter på sig att ta sig runt banan.
2. Tävlingen går till så att den tävlande placerar ut roboten bakom startlinjen. Därefter aktiverar den tävlande roboten på domarens signal.
3. Tiden startar när roboten kör förbi porten första gången och stannar när någon del av roboten går förbi porten andra gången efter att ha kört ett helt varv (varje del av linjen har varit överlappad). Tiden ska om möjligt mätas med ett automatiskt system, men kan (ifall ett sådant inte finns tillgängligt) mätas av en domare med tidtagarur.
4. Roboten måste ha markkontakt hela tiden.
5. Roboten måste följa linjen hela tiden. En robot anses följa linjen ifall någon del av roboten överlappar linjen. Om roboten lämnar linjen måste den återta linjen igen på det stället där linjen tappades eller tidigare.
6. Tävlingsformatet kommer att bero på antalet deltagare, så detta kommer att bestämmas närmare tävlingen.

6. Avancerad linjeföljning

1. Avancerad linjeföljning går till på precis samma sätt som vanlig linjeföljning förutom att det finns en del hinder på banan. Hindren är potentiella 90°-hörn, en tunnel, ett avbrott och ett objekt i form av en tegelsten eller motsvarande. Dessa beskrivs mer ingående nedan:
2. Det kan förekomma skarpa vinklar på linjen ned till 90°. Vid dessa områden kan linjen överlappa.
3. Linjen kommer att gå genom en tunnel. Tunneln är en låda som är minst 30 cm bred och 30 cm hög. Roboten måste fortfarande följa linjen trots den ändrade ljussättningen.
4. På ett ställe på linjen finns ett avbrott. Avbrottet är 10 cm långt och realiserad med vit tejp.

Avbrottet är placerat på en raksträcka, vilket innebär att det kommer att vara rak linje innan och efter avbrottet.

5. På banan finns ett objekt utplacerat. Objektet är en tegelsten (eller motsvarande) med ungefärliga dimensioner 25 cm x 12 cm x 6 cm. objektet är placerad mitt på linjen, och roboten måste vika av från linjen, åka runt objektet och hitta tillbaka till linjen igen för att fortsätta följa den. Roboten får vika av från linjen senast 15 cm innan objektet och återta linjen senast 15 cm efter objektet. Roboten får inte knuffa bort objektet.

7. Tveksamheter

1. Använd ert sunda förnuft när ni tolkar reglerna. Vid tveksamheter och tvister gällande regeltolkningar har huvuddomaren beslutsrätt att avgöra vad som är rätt och fel.

Appendix II

Lista över utrustning

Nedan redogörs all utrustning och programvara som nyttjats vid framställningen av den fysiska linjeföljaren och tillhörande styrsystem och mjukvara.

Instrument och Hårdvara

Instrument/hårdvara skrivs i fet stil, eventuell modell skrivs i kusiv stil och kortfattad förklaring skrivs till höger.

Oscilloskop, *Rigol DS2072A*	Användes för att undersöka nivåer och störningar på signalerna från optokopplarna samt rippel från spänningsomvandlare och utnivå för batteri.
Multimeter, *Fluke 79 III*	Användes för mätning av spänning, ström och resistans hos komponenter och spänningskällor. Användes till enklare mätningar där stor precision ej nödvändigt, oscilloskop användes när mer precision behövdes.
Etsnings-utrustning	Användes för att framställa egentillverkade kretskort. Innefattar skrivare, mask, UV-box, framkallare natriumhydroxid (NAOH), etstank, etsvätska natriumpersulfat (Na2S2O8) och stripper (aceton).
Laserskärare, *Jinan King Rabbit Technology Development Company Ltd. Rabbit HX40A*	Användes för att skära ut chassidelar och fästen för fälgar. Materialet som delana tillverkades i var akrylplast.
3D-skrivare, *Bukobot v2*	Användes för att skriva ut fästen och fälgar. Materialet som skrevs ut i utgörs av ABS-plast.
Lödutrustning	Användes för lödning av komponenter på kretskort, lödning av kontakter samt ledare och isolering av kontaktytor genom uppvärming av krympslang. Innefattar lödkolv, lödtenn, lödpasta, lödugn, "hjälpande hand" och krympslang.
STM32F407 *DISCOVERY* BOARD	Utvecklingskort som användes för utveckling av systemet. Mikrokontrollern utgörs av en STM32407VG. Sköter all behandling av insignaler från givare, reglering av systemet och dess utsignaler. Innehåller även kretsar för ADC, UART och DMA.
STM32F405RG	Den mikrokontroller som ersatte STM32F407VG på det specialtillverkade PCB som styr den slutgiltiga linjeföljaren.

Optokopplare De optiska givare som användes för läsning av linjen samt hjulens rotationshastighet. Dessa består av en sändardel, en IR-lysdiod, och en fototransistor som mottagare. De används för att mäta hur reflektiv en yta är. Ljuset från IR-lysdioden reflekteras mot ytan och fototransistorn resistans bestäms av den återreflekterade ljusmängden.

Batterier, *Turnigy nano-tech 950 mAh* Ett litium-polymerbatteri (LiPo) som användes som kraftkälla i systemet. LiPo-batterier är fördelaktiga då de klarar av att leverera stora strömmar och har hög energitäthet.

Spännings-omvandlare En buckomvandlare användes för att sänka spänningen från batteriet till fem (5) volt. Den lägre spänningen användes för att driva mikrokontrollern.

Motordrivare Kraftelektonik som används för att styra motorer. Styrsignalen är pulsbreddsmodulerad (PWM) och duty cycle kontrollerar nivån på utsignalen.

Likströmsmotor Motor som drivs med likspänning. Vald typ till roboten är borstade likstömsmotorer.

Blåtandsmodul *RN41* Användes för att trådlöst kommunicera med linjeföljaren.

Datorer Behövs för att programera mikrokontrollen, felsöka kod och hårdvara samt att skriva rapporten.

Mjukvara

Flera typer av mjukvara användes för att genomföra projektet då linjeföljaren både designades, konstruerades och programmerades. Program skrivs i fet stil, eventuell underrubrik skrivs i kusiv stil och kortfattad förklaring skrivs till höger.

Programmering:

C Ett lågnivåspråk som användes för att programmera ett av de program som användes för styrning av roboten.

Python Ett scriptspråk som användes för att med hjälp av en PC kommunicera med roboten via seriell kommunikation (UART). Detta gjorde det möjligt att ända parametrar och hämta data under körning istället för att programmera om systemet.

PyQt En bindning till Python som användes för att skapa ett grafiskt användargränssnitt som förbättrade användarvänligheten vid kommunikation med roboten.

MATLAB R2014b, Programvaror från Mathworks. Högnivåspråk med visuella element som möjliggör enkel sammankoppling av systemet genom användning av färdiga eller användardesignade block.

Simulink	Waijung-blockset är ett tilläggspaket med Simulink-block som är anpassade för STM32-mikrokontrollern och utvecklas av det thailändska företaget Aimagin användes också.
ST-Link	Programvara från STMicroelectronics. Används för att programmera mikrokontrollern via USB.
OpenOCD	Open source programvara som användes för att programmera mikrokontrollern hos den slutgiltiga linjeföljaren.

Kompilatorer:

GCC Arm	kompilator för Cortex-R/Cortex-M. Användes av gruppen som programmerade traditionell C-kod för att kompilera kod till mikrokontrollern.

Texteditor:

Sublime	Texteditor. Användes för att skriva C-kod till ett av styrprogrammen.

Bilder:

Indesign CS6	Programvara från Adobe för layout. Användes till att skapa posters till utställning av kandidatarbeten samt för att skapa rapportens framsida.
Photoshop CS6	Programvara från Adobe för fotoredigering. Användes till att beskära bilder.
KiCad bibliotek	KiCad bibliotek som används för att i framställa 3D-bilder av kretsar som designats i mjukvaran. Tillgänglig från: `http://smisioto.no-ip.org/elettronica/kicad/kicad-en.htm`

Mekanik:

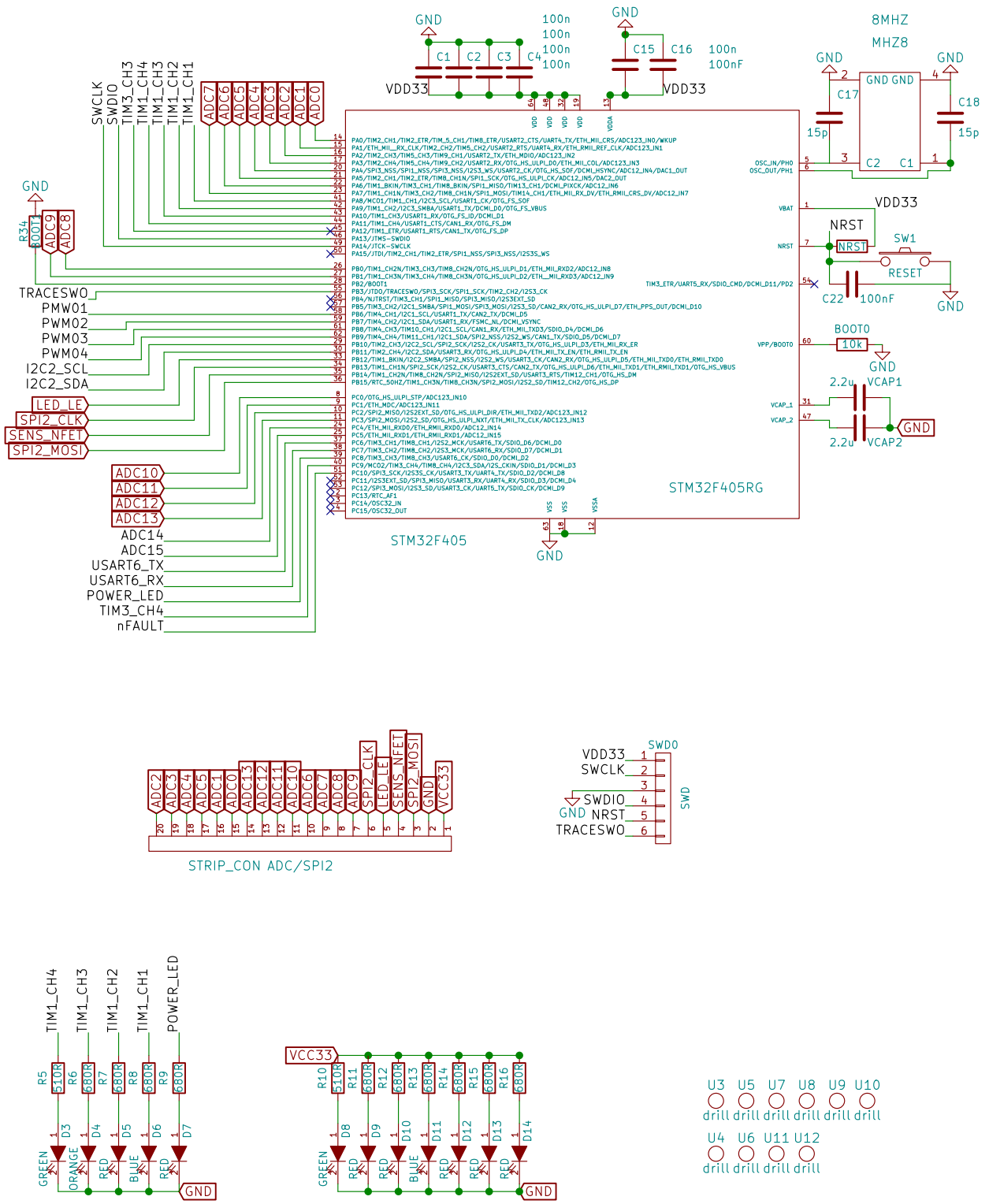
CATIA	Mjukvara för design av mekaniska komponenter, utvecklat av Dassault Systems. Användes för design av hjul/fälgar.
SolidWorks	Mjukvara för design av mekaniska komponenter, utvecklat av Dassault Systems. Användes för design av chassi.

Elektronik:

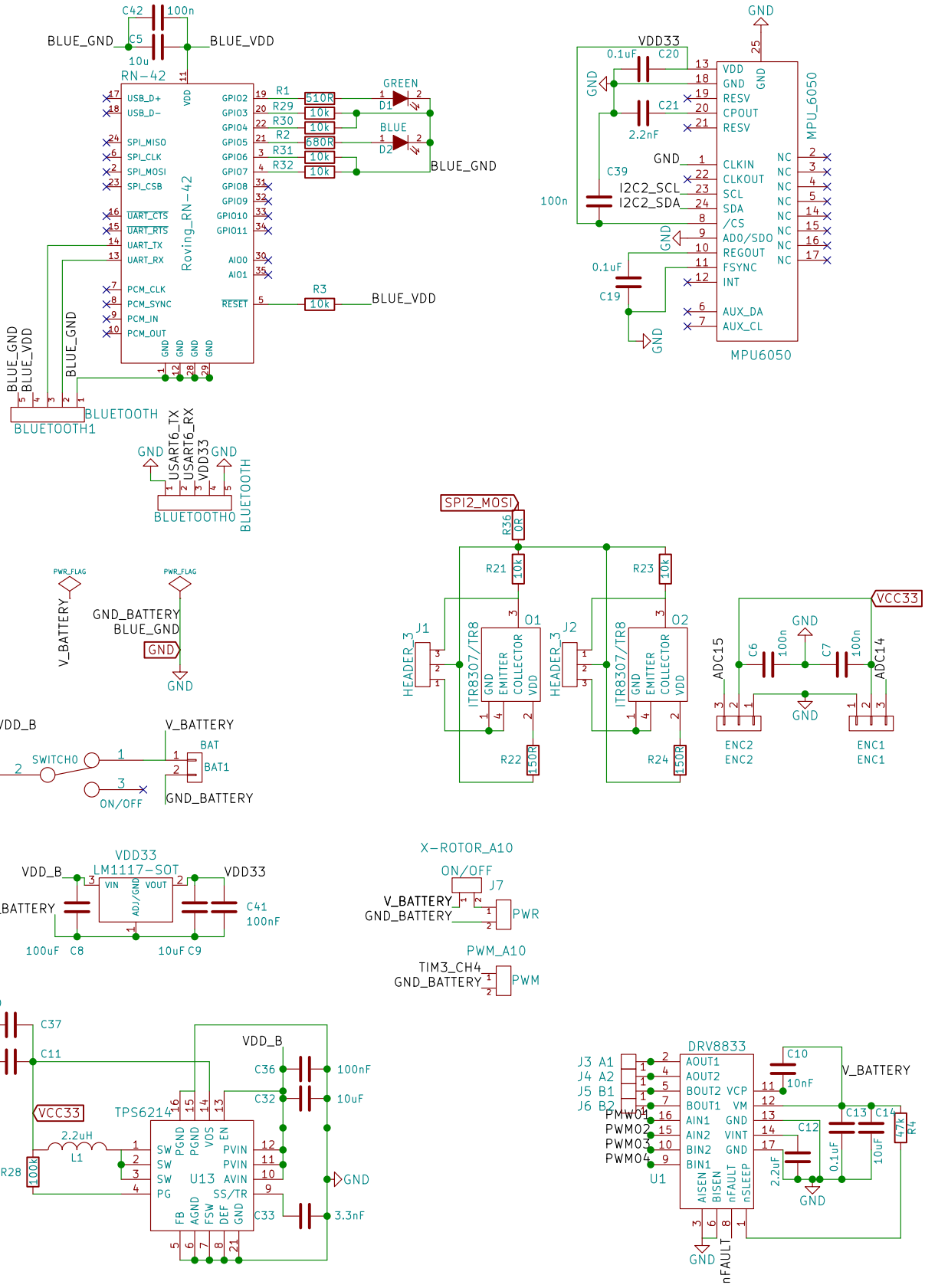
KiCad	Open Source programvara för kretsdesign. Användes för kretsdesign av givarstrip, blåtandsmodul, optokopplare till hjulen samt övriga egentillverkde kretsar.

Appendix III

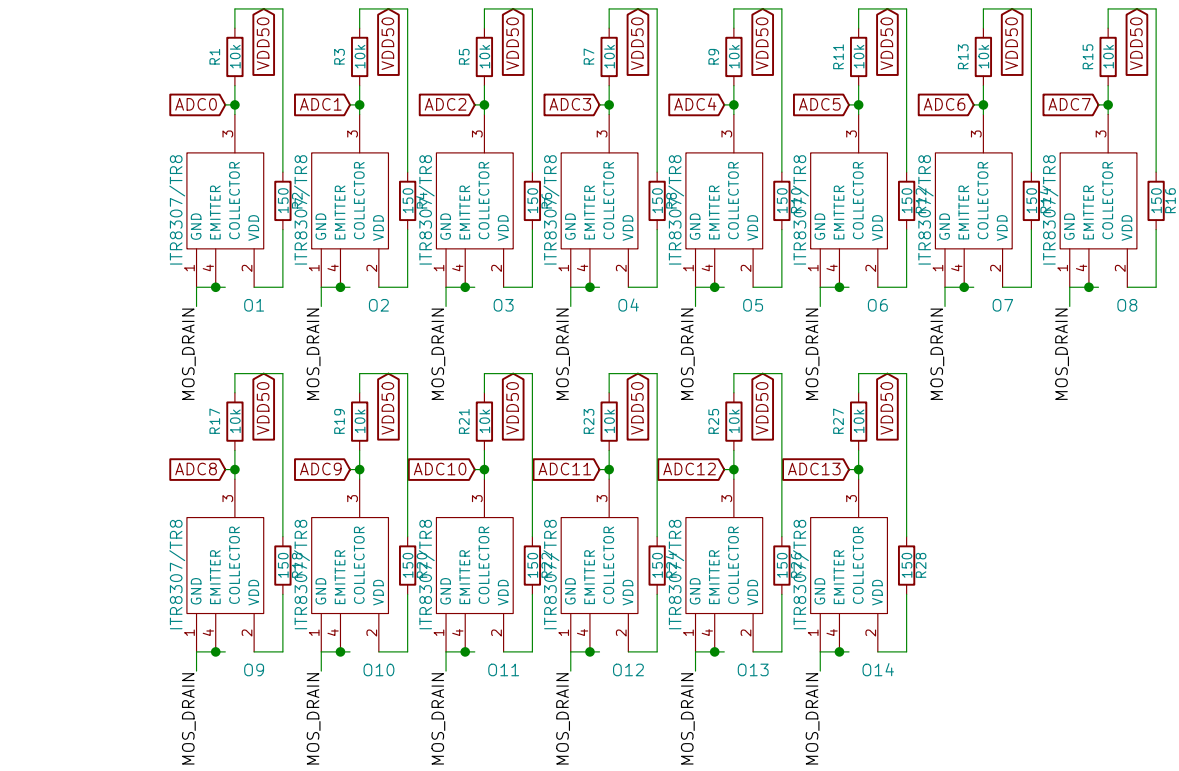
Kopplingsschema för Walknut

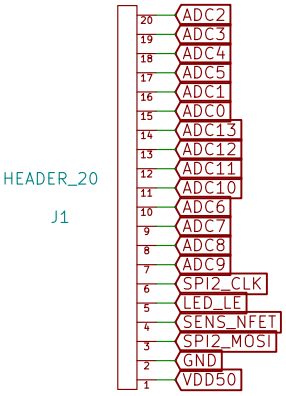


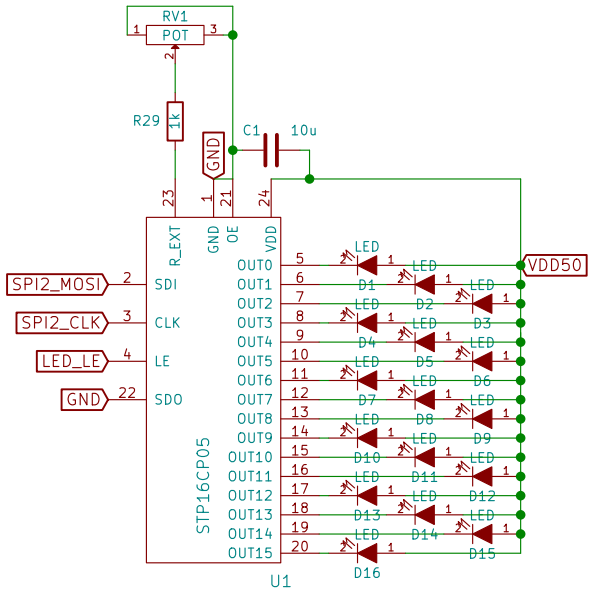
Fredrik Åkerlund		
Sheet: / File: lf_motherboard_v2.sch		
Title: Linefollower Motherboard		
Size: A4	Date: 20-4-2015	**Rev: 2.0**
KiCad E.D.A. eeschema (2015-03-13 BZR 5510)-product		Id: 1/1

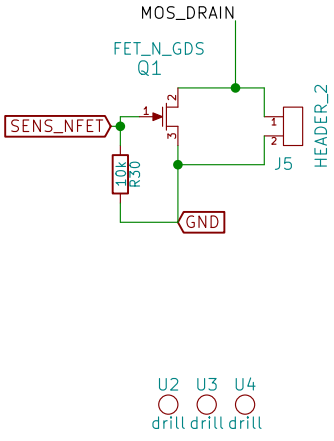


Fredrik Åkerlund		
Sheet: / File: lf_motherboard_v2.sch		
Title: Linefollower Motherboard		
Size: A4	Date: 20-4-2015	**Rev: 2.0**
KiCad E.D.A. eeschema (2015-03-13 BZR 5510)-product		Id: 1/1









Fredrik Åkerlund		
Sheet: / File: lf_strip_2.sch		
Title: Linefollower Sensorstrip		
Size: A4	Date: 2015-04-20	**Rev:**
KiCad E.D.A. eeschema (2015-03-13 BZR 5510)-product		Id: 1/1