LUNDS UNIVERSITET

LTH

LUNDS TEKNISKA HÖGSKOLA

Digitala System

EITA 15

Kursansvarig: Bertil Lindvall &

Lars-Göran Larsson

Projekt

Labyrintbilen

2022.03.22 – 2022.05.23

Eckhausen, Anton

Hoang, Trung

Kjellsson, Evelina

Lempinen, Jennifer

Svensson, Mikael

Sammanfattning

Här finns en kort sammanfattning av projektet, från start till resultat.
I denna rapport finns en redogörelse av projektarbetet i kursen Digitala system.
Under gruppens första möte togs det gemensamma beslutet att bygga ett självkörande fordon som skulle kunna hitta ut ur en labyrint samt bygga en hemsida för presentation och demonstration av arbetet.
Därefter gjordes en kravspecifikation för att specificera och avgränsa projektet samt för att kunna ta fram komponenter till fordonet. När kravspecifikationen var klar delades arbetet upp mellan gruppmedlemmarna.
Följande steg genomfördes: komponentlista, kretsschema i Eagle, hemsida i HTML, sekvens, tillståndsdiagram, montering av komponenter på fordon med hjälp av lödning, 3D utskrift via ritning i Solid Works, C programmering i Atmel Studio 7.
Några av de komponenter som varit en central del för projektet har varit: ATmega 1284, QTR-3A reflectance sensor array, DRV8833 dual motor driver carrier och LCD display.
Projektet resulterade i ett linjeföljande fordon som detekterar olika typer av svängar och korsningar för att sedan göra ett slumpmässigt vägval och sedan fortsätta sin rutt samt stanna vid ett stopp och känna igen en återvändsgränd. Fordonet har en latch som aktiverar den och för att börja köra trycks grön knapp ner och röd knapp för att stanna den. Fordonet har även en timer som visas på en display.

Nyckelord

ATmega1284, linjeföljande robot, digitala system, kretsschema, sekvens

Abstract

Here’s a short abstract of the project from start to result.
This report contains a declaration of the project in the course Digital systems.
During the groups first meeting, the joint decision was to build a self-driving vehicle that could find its way out of a labyrinth and to also build a website for presentation and demonstration of the vehicle.
Then a requirements specification was made to specify and limit the project to be able to find out what components were needed for this project. When the specification was completed, the work was divided between the group members.
The following steps were performed: list of components, circuit diagram in Eagle, website in HTML, sequence, condition diagram, assembling of components on vehicle by soldering, 3D printing via drawing in Solid Works and C coding in Atmel Studio 7.

Some of the components that have been a central part of the project have been: ATmega 1284, QTR-3A reflectance sensor array, DRV8833 dual motor driver carrier and a LCD display.

The project resulted in a line-following vehicle that detects different types of turns and intersections and then makes a random choice of route and then continues its route. It also stops at a stop-marking and recognizes a dead end. The vehicle has a latch that activates it, and to start driving the green button has to be pressed, and to stop it the red button has to be pressed. It also has a timer that shows how long it takes to go from start to finish, the timer is shown on a display.

Keywords

ATmega 1284, line-following robot, digital systems, circuit diagram, sequence

Innehållsförteckning

1. Inledning

I kursen Digitala system fick klassen under vårterminen 2022 i uppgift att bilda en grupp inför ett projektarbete som skulle resultera i en prototyp baserad på mikroprocessorn ATmega 1284. Gruppen skulle med handledning från kursansvarig och med gemensam kunskap samt gedigen research kunna skapa och presentera en prototyp med tillhörande dokumentation i form av rapport, kretsschema och hemsida. Tillsammans tog gruppen ett gemensamt beslut om att skapa ett självkörande fordon som skulle kunna hitta ut ur en labyrint. Tiden var begränsad till 8 veckor och gruppen råddes av handledare till att sikta högt i kravspecifikationen men eventuellt skala ner projektet om tiden ej skulle räcka till.
I rapporten finns information om hur projektet fortlöpt, motgångar och framgångar under arbetets gång, tillvägagångssätt för att skapa maskinen samt ett resultat.

1.1 Syfte

Avsikten med projektet var att tillämpa den kunskap som byggts upp under kursens gång på en prototyp i ett gemensamt grupparbete och därmed fördjupa förståelsen för det som är grundläggande i kursen digitala system. Arbetet har handlat mycket om planering och problemlösning. Det har varit viktigt med tydliga mål och deadlines för att hinna med alla steg och dessutom hålla god kommunikation inom gruppen speciellt vid de tillfällen arbetet varit uppdelat då gruppmedlemmar arbetat på distans.
Att bygga en självkörande robot som skulle kunna hitta ut ur en labyrint var något alla i gruppen upplevde skulle kunna bli ett intressant och lärorikt projekt med tydliga resultat och med god utvecklingspotential.

1.2 Målformulering

Målet med projektet var att förankra och utöka kunskapen i digitala system och i en grupp kunna bygga och visa upp en prototyp av i detta fall ett självkörande fordon.

1.3 Problemformulering

Gruppens första problem var att hitta ett sätt att kommunicera digitalt för att kunna dela med sig av information och arbetsmaterial. Därefter kom följande problemformuleringar: Hur görs ett kretsschema och hur kopplas komponenterna ihop? Hur ska uppbyggnaden av fordonet se ut? Hur implementeras sensorerna och hur används PWM för att reglera hastighet på motorer?

1.4 Avgränsningar

Tre punkter i kravspecifikation som fick bli avgränsningar på grund av tidsbrist var att fordonet skulle kunna stanna vid ett hinder, kartlägga vägen den kört och följa instruktioner för en förbestämd rutt.

2. Teknisk bakgrund

I detta avsnitt kommer de tekniska egenskaper som projektet bygger på presenteras. Mestadels av de beståndsdelar som har använts har funnits tillgängliga i Lunds Tekniska Högskolas laborationslager. Det som har saknats har med hjälp av kursansvarig gjorts tillgängligt genom beställningar. Några specifika tekniker kommer också på ett tekniskt plan presenteras lite djupare då de har en central plats i funktionaliteten för projektet.

2.1 Hårdvara

I tabell 1 presenteras de komponenter som använts för att konstruera fordonet samt hur de är väsentliga för dess funktion. I tabell 2 visas de redskap som varit mest nödvändiga för felsökning samt konstruktion under projektets gång.

Tabell 1

Komponent	Antal	Information
ATMEGA1284	1	Mikrokontroller från Microchip som styr alla komponenter.
QTR-3A Reflectance Sensor Array	3	Sensorer som används för att bilen ska följa en linje samt för att känna igen korsningar i labyrinten. Är placerade framtill, på höger och vänster sida av fordonet.
DRV8833 Dual H-Bridge Drive	1	Komponenten som styr de två DC-motorerna som används för att driva hjulen.
Bolymin 1602 LCD Display	1	Används för att visa tiden det tar för bilen att hitta ut ur labyrinten.
Potentiometer	1	Används för att justera displayens kontrast. Motstånd 10kΩ
Romi Chassis Kit (Polulu)	1	För uppbyggnad av bilen. Kittet innehåller: Rosa chassibasplatta. x1 Mini dc motorer i plast med 120:1 utväxling. x2 70x8mm hjul. x2 Stabilisatorkula. x1
3D-Printad Brygga	1	Monterades på främre delen av fordonet för att stötta kretskortet.
Resistor	2	10kΩ. För att dämpa ingång till knapparna på mikrokontrollern.
Batterichassi	1	För att placera strömkällan och med två ledare strömföra projektet.
AA-Batterier	4	Strömkälla. Laddningsbara batterier. Cirkavärde 1,3V per batteri vid full laddning.

Tabell 2

Redskap	Antal	Information
JTAG, Atmel ICE USB	1	Debug-kontroller som används för programmering och debug av mikrokontrollern.
Oscillator	1	Användes för att mäta signaler från ATMEGA1284:an till DRV8833 Dual H-Bridge Drive.
Eltejp 19 mm	2	Användes för att konstruera labyrinten.
Lödstation	1	Vid lödning inkl. all tillhörande utrustning och förbrukningsvaror.
Kopplingsplatta	1	Kopplingsplatta inklusive kablar av olika diameter användes för realisering av kretsschemat på bilens bas.

2.2 Mjukvara

För att tillverka logiska maskiner behöver dem en kod som säger till dem hur dem ska fungera. I detta projekt skulle inte bara en maskin tillverkas, utan även en hemsida som visar de viktigaste aspekterna från projektet. För samtliga koder som har konstruerats i projektet, har webbsidan www.replit.com använts för att möjliggöra samarbete mellan projektmedlemmar över kodningen.

Kod för bilen

För att programmera mikrokontrollern för fordonet har programvaran Atmel Studio 7 använts. Språket som används är C. För att se koden som konstruerats för fordonet se appendix 7.2.

Hemsidan

För att presentera projektet skulle en hemsida skapas som visar en film på hur maskinen fungerar samt innehålla sidor med kod, rapport och kretsschema. För att konstruera hemsidan har en kod i HTML skapats. Mallar togs fram för att bestämma vilket utseende hemsidan, se appendix 7.4.

2.3 Pulsbreddsmodulering

Pulsbreddsmodulering heter på engelska Pulse Width Modulation (PWM) och är en teknik för att modulera effekten man ger till en komponent. Istället för att skicka en kontinuerlig ström kan man slå på och av strömmen med väldigt korta intervaller. Detta sker så snabbt att en komponent, vilket är en elektrisk motor i detta fallet, inte kan urskilja på de olika lägena för på respektive av. Detta gör att den upplevda effekten för komponenten blir lägre än vid kontinuerlig matning av ström. Både elektriska motorer och LED-ljuskällor är bra exempel på hur PWM kan användas, i fallet med motorn, kan hastigheten ställas och i fallet med ljuskällan, kan ljusstyrkan från denna ställas.

Det finns några begrepp som är viktiga att förstå för hur PWM fungerar och hur den kan sättas upp med hjälp av en mikrokontroller. Titta på figur 1 och läs sedan följande beskrivningar och begrepp.

Tillslagstid

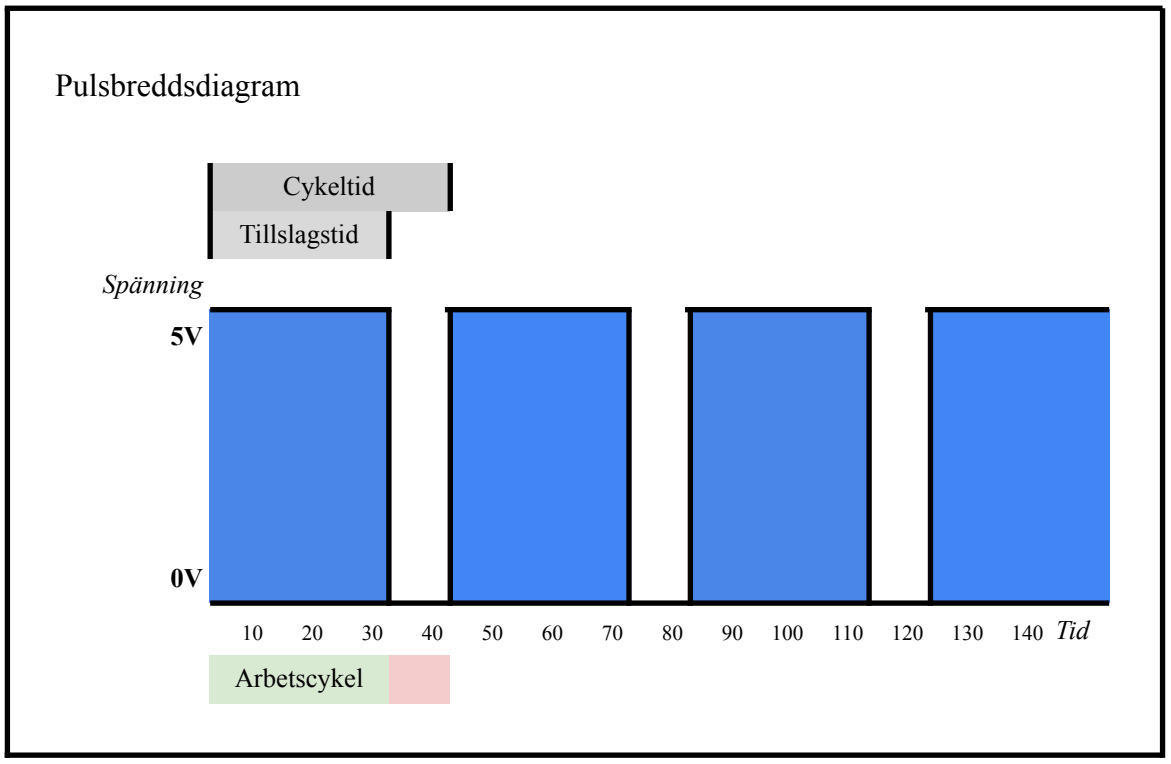
Tillslagstiden är den tid som löpt under tiden signalen eller spänningen varit aktiv hög. Man har alltså slagit till strömmen under denna tidsperiod.

Cykeltid

Cykeltiden är den tid användaren definierat som en konstant tid för en cykel. Det är över denna tidsperiod man tittar på hur stor del där spänningen är tillslagen eller av.

Arbetscykel

Arbetscykeln beskriver förhållandet mellan på och av -läge under en cykeltid. Tittar vi på figur 1 nedan så har vi en cykeltid på 40, där mellan 0-30 på tidsaxeln det anges att signalen är påslagen. Detta beskriver därför förhållande som en arbetscykel om 75%. ¾-delar av cykeltiden är aktiv hög spänning på 5V, ¼-del är spänningen 0V.



Figur 1: Pulsbreddsdiagram

2.4 Sensorer av fototransistortyp

Sensorerna som används i projektet är av fototransistortyp. Fototransistorn är i princip en kombination av två olika komponenter. Fotodioden och transistorn. Därav fototransistor.

Sensorerna har också en ljusgenererande del som genererar ljusenergi (fotoner) i frekvensområdet för infrarött ljus.

Fotodioden är en diod, men utan skyddande hölje, eller med ett hölje som endast kan släppa igenom våglängder av valt frekvensomfång. Fotodioden är omvänd, så den genererar inte ljus som en vanlig diod, utan den är känslig mot inkommande ljus istället. Den genererande enheten skickar i många fall ut infrarött ljus varvid när detta återvänder till basen på fototransistorn, genererar en ström som är proportionell mot energin, det vill säga, hur många fotoner av ljuset som träffat diodens bas. Starkare ljus mot fototransistorn ger mer ström. Efter detta förstärks (gain) energin och man får mätbar spänning från sensorn. Detta ger att man beroende på vilket håll fototransistorn är konstruerad, får en låg spänning vid mycket ljus som absorberas (stor reflektion mot objekt exempelvis) eller en hög spänning.

Notera gärna det faktum att våra lampor numera nästan alltid är konstruerade med LED-teknik (Light Emitting Diode) som alltså är ljusdioder. Dessa dioder skulle kunna kopplas omvänt och belysas med ljus för att reagera (skicka ström) beroende på hur ljusstarkt de blir belysta. Detta kan man experimentera på om man kopplar upp en lysdiod omvänt för att sedan med en voltmeter mäta hur mycket spänning som kommer ur kretsen när man belyser med olika ljusstyrka.

Det finns sensorer som har en konvertering till digital utgång, som alltså ger ström eller inte ström på en utgång (1 eller 0). För detta projektet användes endast sensorer med analoga utgångar på sensorerna.

2.5 Analog till digital omvandling

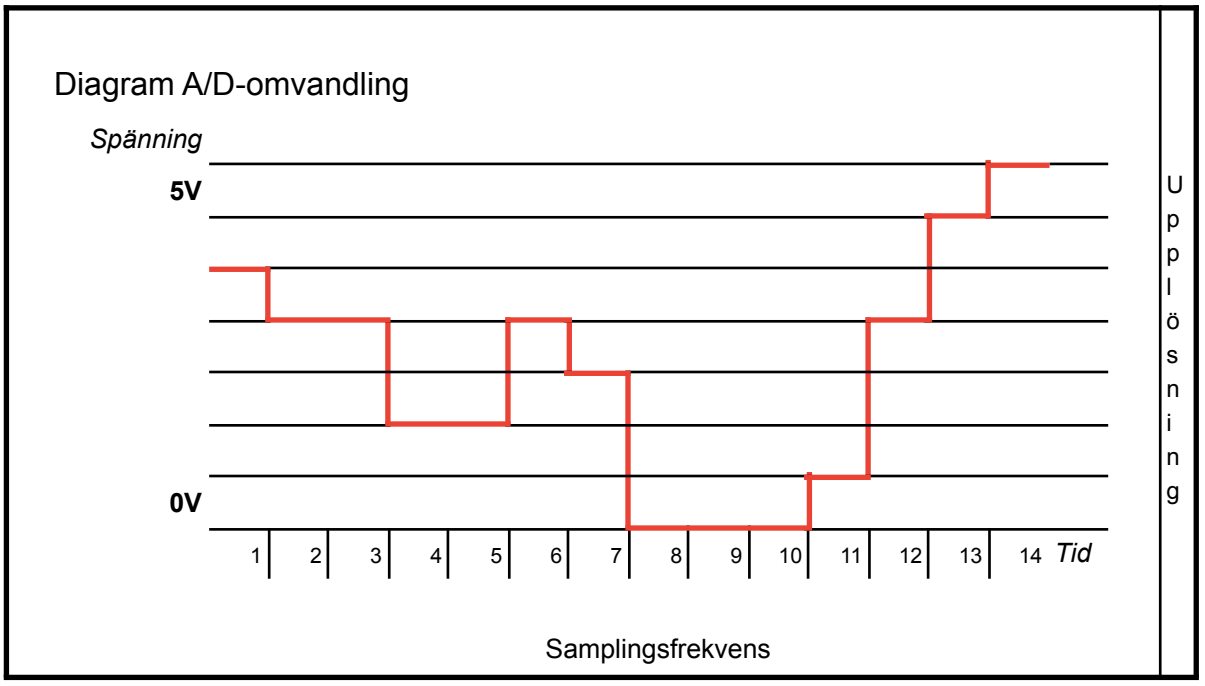
För att den analoga signalen från sensorerna ska kunna hanteras av mikrokontrollern krävs en analog till digital omvandling, detta beskrivs i följande text.

Konvertering av analog insignal sker genom den ADC (Analog to digital converter) enhet som finns på mikrokontrollern. För att förstå konvertering från analog till digital signal behöver man förstå begreppen upplösning och samplingsfrekvens. I figur 2 visas en röd signal som beskriver vilken spänning som just vid samplingstillfället finns på ingången. Samplingsfrekvens beskriver hur ofta sampling sker, dvs hur långt är det mellan samplingarna. Detta beskrivs oftast med enheten Hertz, Hz, vilket är antal per sekund.

Upplösningen är den vertikala uppdelningen som kan ses som svarta streck över diagrammet. Detta avgör hur många steg man kan beskriva för mikrokontrollern. Mikrokontrollern som bara förstår digitalt språk, kräver därför att de olika stegen beskrivs i bitar (binär form).

I diagrammet i figur 2 har vi en uppdelning på 8 steg. Detta kan beskrivas med 3 bitar, från 000 till 111. Diagrammet visar alltså en A/D-omvandlare på 3-bitar. Vid fler bitar desto mer upplöst blir de binära värden som ges till mikrokontrollern och mikrokontrollern kan programmeras för att agera på olika logiska värden.

Notering: För ATMega1284 finns 8 ingångar med 10-bitars omvandling (1024 steg).



Figur 2: Diagram A/D-omvandling

3. Metod

I detta avsnitt kommer strukturen på hur projektet genomförts att presenteras. I försök om att ge en tydlig överblick på hur gruppmedlemmar samarbetat för att enas om ett tydligt mål samt vilka tillvägagångssätt som har implementerats för att lyckas få fram ett lyckat slutresultat.

3.1 Planering

Första steget i projektet var att säkerställa vilken maskin som ska tillverkas. Första mötet med samtliga projektmedlemmar var ett idémöte, för att tillsammans komma överens om ett projekt. Det tog inte lång tid innan gruppen enades om att skapa en självkörande bil. Då samtliga gruppmedlemmar ville göra projektet mer utmanande och intressant, bestämdes det även att en labyrint skulle konstrueras. Bilen skulle sedan programmeras så att den själv lyckas göra egna vägval och kunna hitta ut från labyrinten.

Efter att ha kommit överens om vad som ska tillverkas började arbetet kring att ta fram en kravspecifikation för hela projektarbetet. För att säkerställa att samtliga medlemmar hade en enhetlig bild om vad som behövde genomföras delades kravspecifikationen upp i en tydlig struktur för att enkelt kunna följas samt justeras allt eftersom projektarbetet fortlöpte. Kravspecifikationen delades upp i kategorier enligt tabell 3.
För att projektarbetets momentum inte skulle stagnera vid något tillfälle, sattes även tydliga mål inför nästa träff innan varje gruppmöte avslutades.

Tabell 3

Kategori	**Innehåll**
Fordons egenskaper	Specificerar vad fordonet ska kunna åstadkomma samt vilka begränsningar som kommer finnas då projektet avslutas.
Fordonskomponenter	Tydliggör vilka komponenter som antas behövas utifrån önskade egenskaper för fordonet.
Hemsidans innehåll	Vad ska visas på hemsidan. Rapport, kretsschema etc.
Hemsidans Utseende	Vilken utseende ska hemsidan ha. Layout, font etc.

3.2 Kravspecifikation

Kravspecifikationerna har kompletterats alltefter att projektet har fortlöpt. De krav som har blivit justerade har strukits och har eventuellt ersatts med kompletterande X.2 krav i stället. I tabell 4 presenteras vilka egenskaper som önskas åstadkommas för fordonet, samt i tabell 5 visas komponenter som planeras användas. För att presentera projektet skulle även en hemsida skapas. Två kravspecifikationer togs fram för att strukturera hur hemsidan skulle se ut, det visas i tabell 6 och tabell 7.

I tabellerna 4 - 7 har varje krav märkts upp med en statussymbol.
✓ - Indikerar att kravet har lyckats åstadkommas.
X - Kravet har behövts strykas eller kompletteras under arbetets gång.

Tabell 4

Fordonsegenskaper	**Status**
1. Läsa av linje med hjälp av X antal sensorer	✓
2. Ta sig fram med hjälp av två oberoende motorer	✓
3. Startas med ett knapptryck	X
3.2. Strömförsörjning till komponenter med en latch	✓
3.3. Starta med en knapp - GRÖN	✓
3.4. Stanna med en knapp - RÖD	✓
4. Göra vägval	✓
~~5. Stanna vid ett hinder~~	X
6. Försöka hitta ut ur en labyrint	✓
7. Ska kunna ta tiden på hur lång tid det tar att hitta ut (timer)	✓
~~8. Kartlägga vägen den kört (komma ihåg rutten)~~	X
9. Ska inte kunna backa	✓
~~10. Kunna följa instruktioner för en förbestämd rutt~~	X

Tabell 5

Fordonskomponenter	**Status**
1. ATMEGA-1284	✓
2. DRV8833 Dual h-bridge motor driver	✓
3. 2 st DC motorer	✓
4. GDMI1602K LCD Alfanumerisk display	✓
~~5. Reflective object sensor - 4 st~~	X
5.2. Reflective object sensor - 3 st	✓
6. Hjul - 2x	✓
7. Strömkälla	✓
8. On/Off latch knappar - 2x	✓
9. Tryckknapp	✓
~~10. Resistorer~~	X
10.2. 2 st resistorer, 10k Ω	✓
11. Kablar	✓

Tabell 6

Hemsidans innehåll	**Status**
1. Demonstrationsvideo	✓
2. Rapport	✓
~~3. Ritning~~	✓
4. Kretsschema	✓

Tabell 7

Hemsidans utseende	**Status**
1. Bakgrundsfärg - Flerfärgad	✓
2. Font - Poppins	✓
3. Textfärg - Svart	✓
4. Layout - Vertikal	✓

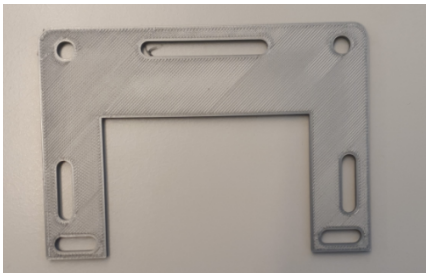
3.3 Utmaningar

Fordonskonstruktion

Vid konstruktionen av fordonet upptäcktes att chassit för bilen var för litet för att kunna stödja kretskortet med dess komponenter. För att lösa detta problem uppstod idén att 3D-printa en brygga som skulle användas för att stödja den delen av kretskortet som stack ut från bilen. För att skapa 3D-ritningingen för komponenten användes programmet *Solid Works*.

Vid första försöket för att printa bryggan, uppstod ett fel som gjorde att bryggan inte blev av god kvalité, se figur 3. Ritningen hade av misstag orienterats i fel riktning, vilket ledde till att bryggan printades stående på högkant istället för liggandes. Efter att har justerat orienteringen inför andra försöket, lades det även till utmätta hål i bryggan för att undvika att behöva borra i komponenten vid konstruktion, resultatet visas i figur 4.

Figur 3. Brygga 1.1



Figur 4. Brygga 1.2

PWM

Varje DC-motor kräver två insignaler från DRV8833 Dual H-Bridge Drive för att kunna driva hjulen på bilen. För att kunna reglera hastigheten på fordonet har PWM-signaler använts för en av de två insignalerna.

För att driva DC-motorerna skulle en PWM-signal samt en logisk 1/0: a skickas från ATmega1284:an till ingångarna för DRV8833 Dual H-Bridge Drive. En stor motgång som uppstod var att få korrekt PWM-signal från ATmega1284:an till dual-H bridge 8833:an. För att lättare kunna felsöka vilka värden som uppstod i de olika leden i kretsen, användes en oscillator. Med hjälp av detta instrument kunde PWM signalen mätas exakt för att få önskat värde.

Vid testning av PWM-signalen var ingen av DC-motorererna fastkopplade till kretsen. Istället hölls kablarna för hand mot DRV8833:ans utgångar för varje värde som önskades testas. Detta tillvägagångssätt användes för att undvika att motorerna var i ständig rörelse vid teststadiet för att minska slitagen samt risken för att skada motorerna.

Efter flera tester lyckades önskat beteende fås på en av de två DC-motorerna, men dock inte den andra. För att närmare kunna jämföra hastigheterna på de båda hjulen som drevs av DC-motorerna, löddades nu komponenterna fast till DRV8833:an.
Vid felsökning upptäcktes att värdet som gick in i ingångarna för DRV8833 Dual H-bridge, inte var samma som gick ut genom dess utgångar. För att säkerställa att denna komponent inte var trasig, kopplades en annan (ny) DRV8833 på ett separat kopplingsdäck, som sedan kopplades ihop med bilens krets. Mätningarna som utfördes från den nya komponenten visade korrekta värden för både utgångar och ingångarna. Slutsatsen drogs då till att den första DRV8833 var defekt och den byttes då ut i kretsen till den nya komponenten. Efter detta utbyte fungerade DC-motorerna som önskat.

Avläsning från sensorerna

En viktig aspekt i konstruktionen av fordonet var att hitta det optimala avståndet för sensorerna för att få så så bra avläsning som möjligt från marken. Detta krävde en del testning och avläsning av de olika värdena man fick från sensorerna, både baserat på avstånd samt underlaget. Enligt informationen given för sensorerna (Pololu Robotics & Electronics (2022)), ska avståndet vara 3-6 mm för att få optimal avläsning från sensorerna. Observera att avståndet för alla 7 sensorerna måste vara så lika som möjligt för att fungera i detta projekt, då det är så koden är uppbyggd. För att kunna komma undan med att sensorerna har olika avstånd måste detta beaktas när koden skrivs, vilket inte har gjorts i detta projekt då det inte ansågs finnas något behov för det. De slutliga avstånden för varje sensor presenteras i tabell 8.

Optimal funktion nås även då batterierna till fordonet är fulladdade med 5 V. Sensorerna är kopplade så att de har en referensspänning till spänningskällan som är kopplad till resten av komponenterna. Detta gör att sensorernas avläsningen påverkas då spänningen för referensen ändras.

Avläsningen från sensorerna är kodat för 8 bitar, vilket innebär att värden som returneras kommer ges i intervallet mellan 0-255. ATmega1284:an har en klockfrekvens på 1Mhz (Atmel Corporation (2016)) som gör att sensorerna kommer avläsa/returnera värden väldigt snabbt. Underlaget som bilen kommer framföras på antas inte vara helt vitt, utan det kan förekomma färgskiftningar samt mörka fläckar. För att göra en tydligare avläsning om sensorerna har identifierat svart tejp eller ej, har därför ett medelvärde utifrån 50 avläsningar gjorts. Sedan har detta medelvärde jämförts med ett gränsvärde för att sedan tilldela sensorn 1/0 baserat på om ljust eller mörkt underlag har identifierats. Gränsvärdet har satts utifrån avlästa värden som presenteras i tabell 8.

Enligt tabell 8 varierar värdena givna från sensorerna mellan 50-120 för ljust underlag och mellan 205-240 för svart underlag. Utifrån dessa värden sattes därmed gränsvärdet för identifiering av tejpen till 190. Överstiger värdet 190 kommer sensorvariabeln tilldelas 1 för identifiering av tejp, och värdet är under 190 tilldelas sensorn värdet 0 för ingen tejp.

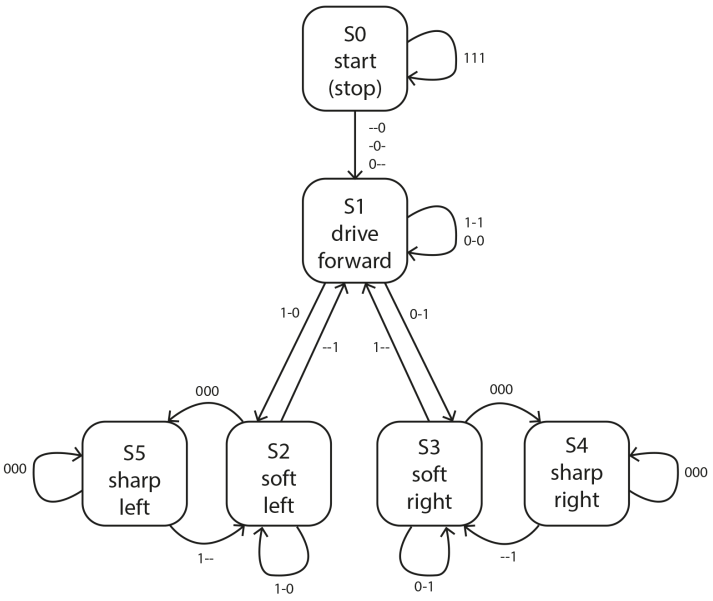
Tabell 8

Sensor	Position på bilen	Avstånd från marken	Medelvärde för 50 avläsningar Ljust underlag	Medelvärde för 50 avläsningar Svart underlag
SF1	Framsensor - Höger	2,9 mm	60	210
SF2	Framsensor - Mitten	2,9 mm	52	226
SF3	Framsensor - Vänster	3,0 mm	63	230
SL1	Vänster sensor - Fram	4,1 mm	112	221
SL2	Vänster sensor - Bak	4,0 mm	97	223
SR1	Höger sensor - Fram	3,5 mm	79	220
SR2	Höger sensor - Bak	3,9 mm	63	225

Sekvenser

Konstruktionen har två huvudfunktioner som är att köra rakt fram och att identifiera en korsning för att sedan svänga till den tillgängliga riktningen.

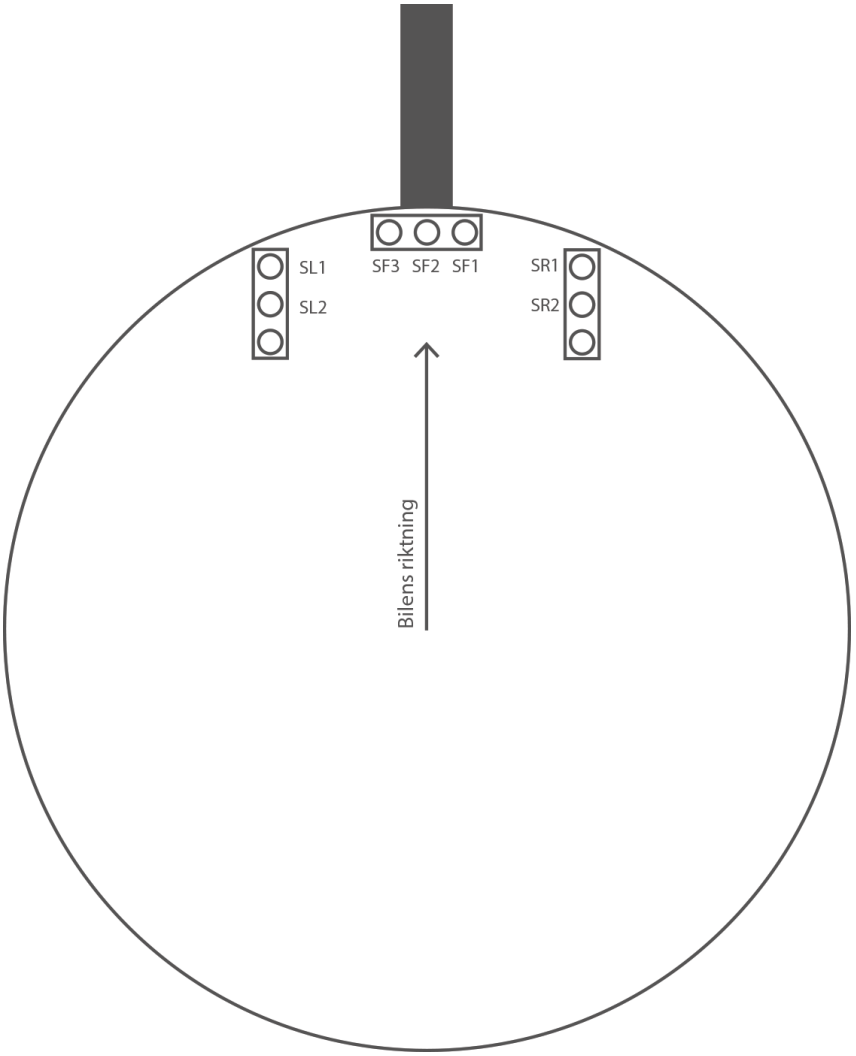
Med hjälp av de främre sensorerna SF1, SF2 och SF3 som uppvisades i figur 6 så kan konstruktionen förflytta sig längs en linje, i det här fallet längs svart tejp. I normalt tillstånd förflyttar konstruktionen på sådant sätt att den svarta linjen ligger mellan sensorerna SF1 och SF3. Båda DC motorer ska rotera med samma hastighet, vilket leder till att bilen kör rakt framåt. När en av de sensorerna SF1 och SF3 kommer över den svarta linjen kommer ena motorn att sakta ned därmed korrigera bilens riktning. Om vänster sensor, i den här fallet SF1 kommer över linjen så minskar hastigheten på vänster motor, vilket leder till att bilen börjar svänga till vänster. På samma sätt, fungerar det när höger sensorn SF3 kommer över linjen. För att demonstrera hur programmeringen är uppbyggd, ritade vi upp en sekvensdiagram i figur 5 för att uppvisa alla tillstånd för främre sensorerna SF1, SF2, SF3.



Figur 5. Sekvensdiagram “Drive forward”

Med hjälp av sidosensorerna SL1, SL2, SR1, SR2 och de främre sensorerna som uppvisades i figur 6 kan konstruktionen identifiera olika korsningar. När konstruktionen passerar en korsning så registreras alla riktningsalternativ med hjälp av tre variabler. De variablerna tilldelas ett värde som är skiljt från noll för att avgöra vilka riktningar är tillgängliga för konstruktionen. Programmeringen är byggd på ett sådant sätt så att när en av sidosensorerna SL1 och SR1 detekterar en korsning tillämpar konstruktionen en rad olika funktioner. Först ska konstruktionen fortsätta att köra rakt fram tills den passerar korsningen, med andra ord tills sidosensorerna SL1, SL2, SR1 eller SR2 returnerar noll.

Under processen ska konstruktionen registrera olika riktningsalternativ. Därefter exekveras en funktion för att generera ett slumpmässigt tal för att avgöra vilken riktning konstruktionen ska ta. Till sist roterar konstruktionen till den bestämda riktningen, roteringen åstadkommer med hjälp av den främre sensorn SF2 som indikera hur lång konstruktionen ska rotera och två DC-motorer som roterar åt motsatta håll. På bilderna nedan redogörs sensorernas olika sekvenser när konstruktionen passerar olika typer av korsningar. För att se de olika sekvenserna som kan ske se appendix 7.3.



Figur 6. Sekvens raksträcka

Timer

En av målsättningarna för projektet var att tiden skulle mätas från det att bilen startar i början av labyrinten tills att den hittar ut. För att lösa denna metoden för bilen har den inre klockfrekvensen för ATmega1284:an utnyttjats samt har Timer1 använts i mikrokontrollern.

Klockfrekvensen för ATmega1284:an är ställd till 1 Mhz, därefter används en delningsfaktor om 8 i Timer1. Detta ger en frekvens om 125 000 Hz. För att räkna upp till 10 millisekunder, vilket är den minsta möjliga räknade tiden. Därför sätts det att det ska räknas upp till 1250 innan en matchning sker, vilket är det som identifierar att den har räknat klart. Detta gör att man vet att när timer1 har räknat till 1250, har 10ms fortlöpt. Hela denna timern styr uppräkningen av millisekunder(ms), sekunder(sec) och minuter(min), se appendix 7.2 för tidsberäkningskoden och sidan 153 i databladet för ATMega1284.

4. Resultat

Komponenterna monterades enligt kopplingsschemat, programmerades enligt kravspecifikation samt testades vid olika moment. Kodning och konstruktion slutfördes inom den bestämda tidsplanen. Bilen kan köra framåt, identifiera korsningar med hjälp av sensorerna som skickar analoga signaler som i sin tur sedan konverterades till logisk etta eller nolla som sedan utnyttjades i vår sekvensmaskin. Den kan även ta slumpmässiga beslut av vägval dessutom justera hastighet på motorerna i svängar och raksträckor med hjälp pulsbreddsmodulering.

Slutresultatet av projektet blev att det lyckades skapas en linjeföljande bil som har förmågan att kunna starta och stanna med knapptryck samt ta slumpmässiga beslut på svängar i en labyrint för att sedan hitta sin väg ut ur en labyrint och stanna på mållinjen. Under projektets gång implementerades även en display med tidtagarur-funktion som räknar tiden som förlöpt som börjar vid knapptryck och stannar vid mållinjen.

5. Slutsats

Projektgruppen lyckades uppnå det mesta i kravspecifikationen med vissa undantag. För att hinna med tidsplanen fick vissa av funktionerna väljas bort som presenterades i kravspecifikationen.

5.1 Framtida utvecklingsmöjligheter

Projektet kan möjligen vidareutvecklas genom att implementera funktioner som till exempel möjligheten att stanna vid hinder och sedan vända och välja annan rutt. En annan funktion skulle kunna vara bilens förmåga att lägga redan körd rutt på minnet så att den inte gör samma vägval flera gånger i rad vilket kan resultera i att den hittar ut ur labyrinten på betydligt kortare tid. Att programmera en algoritm för att känna av underlag och tejp samt vilka sensorvärden dessa ger kan skapas så att man kan hantera olika typer av underlag.

Konstruktionen kan även vidareutvecklas genom att förbättra programmeringen, lägga till funktioner så det kan verka som ett självkörande fordon i logistikbranschen på en arbetsplats exempelvis ett lager med en förutbestämd rutt. En avståndssensor kan monteras på framsidan av konstruktionen så att den kan stanna vid eventuella hinder.

6. Källförteckning

1. Atmel Corporation, “Atmel-42718C-ATmega1284_Datasheet_Complete-10/2016”, Datasheet, 2016

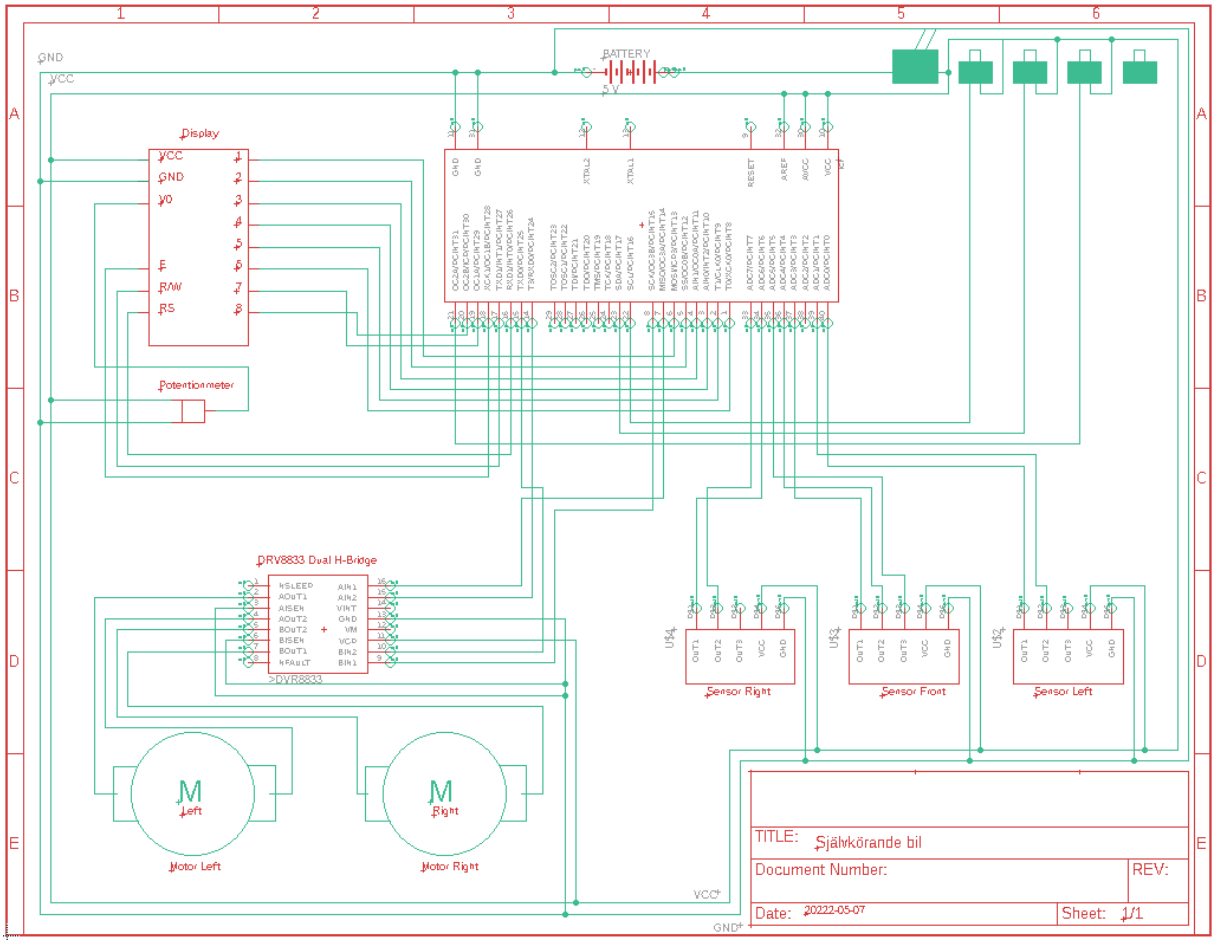
2. Texas Instruments, “DUAL H-BRIDGE MOTOR DRIVER”, Datasheet, 2011

3. Pololu Robotics & Electronics (2022), QTR-3A Reflectance Sensor Array. https://www.pololu.com/product/2456 [2022-05-12]

7. Appendix

I detta avsnitt av rapporten kommer det sammanställas bakgrund- och konkreta ideér som använts för att färdigställa projektarbetet.

7.1 Kretsschema



7.2 C-Kod - Bil

1. Main-metoden

```
int main(void){
setUp();
stopEngine();
init_adc();
init_stopwatch();
Lcd8_Init();
Lcd8_Set_Cursor(1,1);
while (1) {
        readButtonGREEN();
        readButtonRED();
        read_sensors();
        current_state();
        if (redButton==1){
                writeSavedTime();
        }
        if((greenButton==1) && (redButton==0)){
                if(!PossibilityForDirectionChange()){
                switch (state) {
                        case 0:
                                stopEngine();
                                break;
                        case 1:
                                driveForward();
                                break;
                        case 2:
                                adjustToLeft();
                                break;
                        case 3:
                                adjustToRight();
                                break;
                        case 4:
                                turnAroundLeft();
                                break;
                        case 5:
                                turnAroundRight();
                                break;
                        default:
                                turnAroundLeft();
                        }
                }
                if(PossibilityForDirectionChange()){
                        LookingforDirection();
                        ChangingDirection();
                        }
                }
        }
}
```

2. Initieringskod samt värdetilldelning

```
// Kod för initiering, kommer endast köras en gång
void setUp (){
DDRA |= 0x00;
        DDRB |= 0xFF;
        DDRD |= 0x7F;
        TCCR3A |= 0b10100010;
        TCCR3B |= 0b00011100;
}

// Tilldela puls-värde, nbr1 för Motor 1, nbr 2 för Motor 2
void set_pulse(int nbr1,int nbr2){
OCR3A = nbr1;
OCR3B = nbr2;
TCNT3 = 0;
}

// Tilldela periodvärde
void set_period (int nbr){
ICR3 = nbr;
}

//Initierar ADC
void init_adc(){
DIDR0 = 0b11111111;
ADMUX = 0b00100000;
ADCSRA = 0b10000011;
}

//Gör en analog till digital konvertering och returnerar 8-bitars värdet
unsigned char read_ADC(){
        uint16_t adcvalue;
        ADCSRA = ADCSRA | 0b01000000;                                   //
        while((ADSC == 1) );
        adcvalue = ADCH;
        return adcvalue;
}

//Sätter  gränsvärdet och avkodar för hur de 8 bitarna från adc:n ska tolkas som antingen 1 eller 0.
unsigned char sensorvalue()     {
counter = 0;
        avaragevalue = 0;
        while (counter < 50){
                avaragevalue += read_ADC();
                counter++;
}
avaragevalue = avaragevalue/50;
        if ( avaragevalue < 190 ){
                return 0;
        }else {
                return 1;
}
}
//Cykel för att köra runt på de olika sensorerna igenom ADC och lagra deras värden.
void read_sensors (){
         ADMUX = 0b00100000;
 _delay_us(1);
        SL1 = sensorvalue();
ADMUX = 0b00100001;
        _delay_us(1);
```

```
            SL2 = sensorvalue();
            ADMUX = 0b00100011;
            _delay_us(1);
            SF1 = sensorvalue();
            ADMUX = 0b00100100;
_delay_us(1);
             SF2 = sensorvalue();
            ADMUX = 0b00100101;
            _delay_us(1);
            SF3 = sensorvalue();
            ADMUX = 0b00100111;
            _delay_us(1);
            SR1 = sensorvalue();
            ADMUX = 0b00100110;
            _delay_us(1);
            SR2 = sensorvalue();
if (SL1==1 && SL2==1 && SR1==1 && SR2==1 && SF1==1 && SF2==1 && SF3==1){
stopEngine();
            redButton = 1;
greenButton= 0;
goal_is_reached = 1;
 }
  }

//Kontrollerar vilket tillstånd bilen är i och vilket tillstånd den går till
void current_state(){
            if (state == 0){
                        if ( (SF3==1) && (SF2==1) && (SF1==1) ){
                                    state = 0;
                        }else {
                                    state = 1;
                        }
            }
            if (state == 1){
                        if ( ((SF3==1) && (SF1==1)) || ((SF3==0) && (SF1==0))){
                                    state = 1;
                        }else if ((SF3==1) && (SF2==0) && (SF1==0)){
                                    state = 2;
                        }else if ((SF3==0) && (SF2==0) && (SF1==1))       {
                                    state = 3;
                        }
            }
            if (state==2){
                        if((SF3==1) && (SF1==0)){                                                   state=2;
                        } else if ((SF2==1) && (SF1==1)){
                                    state=1;
                        } else if ((SF3==0) && (SF2==0) && (SF1==0)){
                                    state=4;
                        }
            }
            if (state==3){
                        if((SF3==0) && (SF1==1)){
                                    state=3;
                        }else if ((SF3==1) && (SF2==1)){
                                    state=1;
                        }else if ((SF3==0) && (SF2==0) && (SF1==0)){
                                    state=5;
                        }
            }
            if (state==4){
                        if ((SF3==0) && (SF2==0) && (SF1==0)){
```

```
                        state=4;
                } else if(SF3==1){
                        state=2;                                        }
        }
        if (state==5){
                if ((SF3==0) && (SF2==0) && (SF1==0)){
                        state=5;
                }else if(SF1==1){
                        state=3;
                }
        }
}
```

3. Motorfunktioner

```
//Kör framåt
void driveForward (){
                set_period(18);
                set_pulse(8, 8);
                PORTD |= (1<<AIN1);
                PORTD |= (1<<BIN1);
}
//Stannar bilen
void stopEngine(){
                set_period(65000);
                set_pulse(0, 0);
                PORTD &= (0<<AIN1);
                PORTD &= (0<<BIN1);
}
// Justerar till vänster
void adjustToLeft (){
                set_period(18);
                set_pulse(10, 8);
                PORTD |= (1<<AIN1);
                PORTD |= (1<<BIN1);
}
//Justerar till höger
void adjustToRight (){
                set_period(18);
                set_pulse(8, 10);
                PORTD |= (1<<AIN1);
                PORTD |= (1<<BIN1);
}
//Svänger skarpt till höger – Används vid raksträcka
void sharpTurnRight (){
                set_period(18);
                set_pulse(8, 18);
                PORTD |= (1<<AIN1);
                PORTD |= (1<<BIN1);
}
// Svänger skarpt till vänster – Används vid raksträcka
void sharpTurnLeft (){
                set_period(18);
                set_pulse(18, 8);
                PORTD |= (1<<AIN1);
                PORTD |= (1<<BIN1);
```

```
}

// Svänger runt bilen runt sin egen axel mot höger. Sätter ena motorn framåt och andra bakåt.
void turnAroundRight (){
                        set_period(18);
                        set_pulse(8, 8);
                      PORTD = 0b00000001;
}
//Svänger runt bilen runt sin egen axel mot vänster.
void turnAroundLeft (){
                        set_period(18);
                        set_pulse(8, 8);
                        PORTD = 0b00000010;
}

//Roterar till vänster
void turningLeft(){
                          if( (SF1==1) || (SF2==1) || (SF3==1)){
  while((SF1==1) || (SF2==1) || (SF3==1)){
                                                              read_sensors();
                                                              turnAroundLeft();
                                          }
                                          while(SF2 == 0){
                                                              read_sensors();
                                                              turnAroundLeft();
                                          }
                        }else if((SF1==0) && (SF2==0) && (SF3==0)){
                                          while(SF2 ==0 ){
                                                              read_sensors();
                                                              turnAroundLeft();
                                          }
                        }
  }

//Roterar till höger
void turningRight(){
  // funktionnen gör att bilen kan rotera till höger
                        if((SF1==1) || (SF2==1) || (SF3==1)){
                                          while((SF1==1) || (SF2==1) || (SF3==1)){
                                                              read_sensors();
                                                              turnAroundRight();
                                          }
                                          while(SF2 == 0){
                                                              read_sensors();
                                                              turnAroundRight();
                                          }
                        }
                        else if((SF1==0) && (SF2==0) && (SF3==0)){
                                          while(SF2 == 0 ){
                                                              read_sensors();
                                                              turnAroundRight();
                                          }
                        }
  }

//Avgör om det finns möjlighet till riktningsändring
int PossibilityForDirectionChange() {
                      if ((SL1 == 1) || (SR1 == 1)) {
                                          return 1;
                        }
```

```
                return 0;
}

//Söker efter olika riktningar
void LookingforDirection() {
                if (SL1 == 1) {
                                left = 1;
                              left1 =1;
                                while (((SL1 == 1) || (SL2 == 1)) && goal_is_reached==0) {
                                                read_sensors();
                                                driveForward();
                                                if (SR1 == 1) {
                                                                right = 3;
                                                                 right1 =1;
                                                }
                                }
                                if ((SF2 == 1)||(SF1 == 1)||(SF3 == 1)) {
                                                straight = 2;
                                             straight1 =1;
                                }
                }else if (SR1 == 1) {
                                right = 3;
                                 right1 =1;
                                while
                                (((SR2 == 1) || (SR1 == 1)) && goal_is_reached==0) {
                                                read_sensors();
                                                driveForward();
                                                if (SL1 == 1) {
                                                                left = 1;
                                                                left1 =1;
                                                }
                                }
                                if ((SF2 == 1)||(SF1 == 1)||(SF3 == 1)) {
                                                straight = 2;
                                                straight1 =1;
                                }
                }
                sum = straight1+left1+right1;
}

//Ändrar riktning
void ChangingDirection() {
                stopEngine();
              if ((sum == 1) && (goal_is_reached==0)){
                                              if(left == 1){
                                                                turningLeft();
                                              }else if (right == 3){
                                                                turningRight();
                                }else {
                                                                driveForward();
                                              }
                                }else if((sum > 1) && (goal_is_reached==0)){
              uint8_t number = (ms % 3)+1;
                while((number != left)&&(number != right )&&(number != straight)){
                                number = (number % 3) +1;
  }
 if(left == number && goal_is_reached==0){
  turningLeft();
  }
 else if (right == number && goal_is_reached==0){
  turningRight();
```

```
    }
  else if (straight == number && goal_is_reached==0){
    driveForward();
    }
  }
if (goal_is_reached==0)
{
                        driveForward();
}
}
```

4. Knappfunktion

```
//Läser av GRÖN knapp
void readButtonGREEN (){
                    if ((PIND & 0b10000000) == 0b10000000){
                                        lengthOfButtonGREEN++;
                      }
                      if(lengthOfButtonGREEN>3){
                                        lengthOfButtonRED = 0;
                                        lengthOfButtonGREEN = 0;
                                        greenButton = 1;
                                        redButton = 0;
                                        goal_is_reached = 0;
                                        timeCounter = 0;
                                        ms = 0;
                                        sec = 0;
                                        min = 0;
                                        Lcd8_Clear();
                      }
}
//Läser av RÖD knapp
void readButtonRED (){
                    if ((PINC & 0b00000001) == 0b00000001){
                                        lengthOfButtonRED++;
                      }
                      if(lengthOfButtonRED>3){
                                        lengthOfButtonRED = 0;
                                        lengthOfButtonGREEN = 0;
                                        greenButton = 0;
                                        redButton = 1;
                                        stopEngine();
                                        goal_is_reached = 1;
                      }
}
```

5. LCD-skärm

```
// För att beräkna hur många siffror i talet för millesekund/sekund/minut
void countDigit(int n){
		if ((n/10) == 0){
				noDigits = 1;
		}
		else if (n/10 >= 1){
				noDigits = 2;
		}
}
//Utskriftsfunktion för tiden som är just nu
void writeTime(){
		itoa(ms, buffer_ms, 10);
		itoa(min, buffer_min, 10);
		itoa(sec, buffer_sec, 10);
		Lcd8_Set_Cursor(1,1);
		Lcd8_Write_String("Time: ");
		Lcd8_Set_Cursor(1,8);
		Lcd8_Write_String(buffer_min);
		countDigit(sec);
		if(noDigits == 1){
				Lcd8_Set_Cursor(1,9);
				Lcd8_Write_String(":0");
				Lcd8_Set_Cursor(1,11);
				Lcd8_Write_String(buffer_sec);
		}else if(noDigits > 1)		{
				Lcd8_Set_Cursor(1,9);
				Lcd8_Write_String(":");
				Lcd8_Set_Cursor(1,10);
				Lcd8_Write_String(buffer_sec);
		}
		Lcd8_Set_Cursor(1, 12);
		Lcd8_Write_String(":");
		Lcd8_Set_Cursor(1, 13);
		Lcd8_Write_String(buffer_ms);
}
//Utskriftsfunktion för tiden som är sparad efter mål eller röd knapp är nedtryckt
void writeSavedTime(){
				msSaved = ms;
				secSaved = sec;
				minSaved = min;
				if (timeCounter==0){
						itoa(msSaved, buffer_ms, 10);
						itoa(minSaved, buffer_min, 10);
						itoa(secSaved, buffer_sec, 10);
						Lcd8_Set_Cursor(2,1);
						Lcd8_Write_String("Time: ");
						Lcd8_Set_Cursor(2,8);
						Lcd8_Write_String(buffer_min);
						countDigit(sec);

				if(noDigits == 1){
						Lcd8_Set_Cursor(2,9);
						Lcd8_Write_String(":0");
						Lcd8_Set_Cursor(2,11);
						Lcd8_Write_String(buffer_sec);
				}else if(noDigits > 1){
						Lcd8_Set_Cursor(2,9);
						Lcd8_Write_String(":");
```

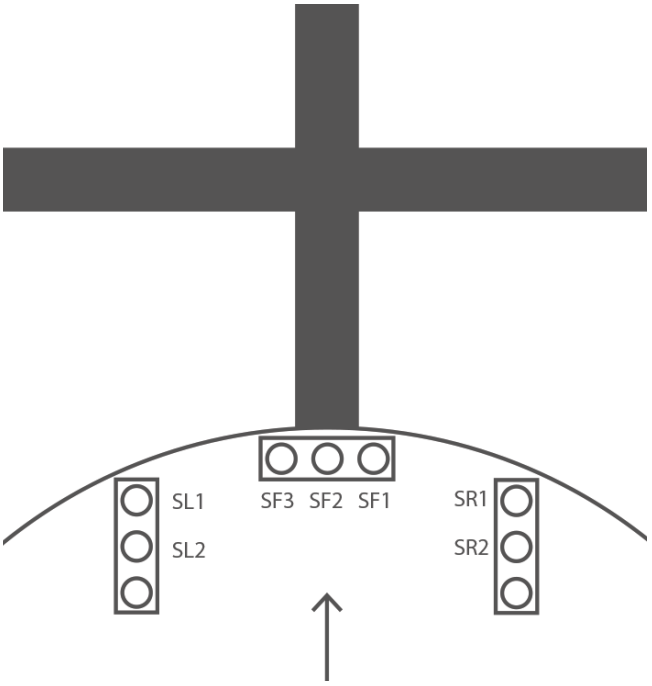
```
                                                Lcd8_Set_Cursor(2,10);
                                                Lcd8_Write_String(buffer_sec);
                                        }
                                        Lcd8_Set_Cursor(2, 12);
                                        Lcd8_Write_String(":");
                                        Lcd8_Set_Cursor(2, 13);
                                        Lcd8_Write_String(buffer_ms);
                                        timeCounter++;
                                        }
                        }
//Kommando som ska skickas till displayen (finns fördefinierade funktioner)

void Lcd8_Cmd(char a)
{
        pinChange(RS,0);            // => RS = 0
        Lcd8_Port(a);             //Data transfer
        pinChange(EN,1);            // => E = 1
        _delay_ms(1);
        pinChange(EN,0);            // => E = 0
        _delay_ms(1);
}

//Skickar kommandot för att rensa displayen

void Lcd8_Clear()
{
        Lcd8_Cmd(1);
}

//Bestämmer läget för nästa input på displayen.

void Lcd8_Set_Cursor(char a, char b)
{
        if(a == 1)
        Lcd8_Cmd(0x80 + b);
        else if(a == 2)
        Lcd8_Cmd(0xC0 + b);
}

//Initierar displayen

void Lcd8_Init()
{
        pinChange(RS,0);
        pinChange(EN,0);
        _delay_ms(20);
        Lcd8_Cmd(0x30);
        _delay_ms(5);
        Lcd8_Cmd(0x30);
        _delay_ms(1);
        Lcd8_Cmd(0x30);
        _delay_ms(10);

        Lcd8_Cmd(0x38);    //function set
        Lcd8_Cmd(0x0C);    //display på,cursor off,blink off
        Lcd8_Cmd(0x01);    //clear display
        Lcd8_Cmd(0x06);    //entry mode, set increment
}

//Skriver ut ett tecken
```

```
void Lcd8_Write_Char(char a)
{
	pinChange(RS,1);		// => RS = 1
	Lcd8_Port(a);		//Data transfer
	pinChange(EN,1);		// => E = 1
	_delay_ms(1);
	pinChange(EN,0);		// => E = 04
	_delay_ms(1);
}

//Funktion för att skriva ut flera tecken i följd via en string

void Lcd8_Write_String(char *a)
{
	int i;
	for(i=0;a[i]!='\0';i++)
	Lcd8_Write_Char(a[i]);
}
```

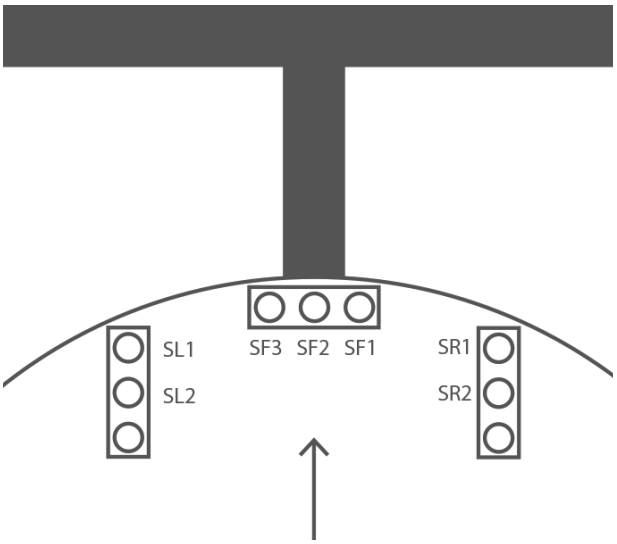
6. Timerfunktion

```
// Initierar timer1 i mikrokontrollern. Sätter prescaler och OCR1A så att en match (interrupt) motsvarar 10ms
void init_stopwatch(){
		TCCR1B |= (1<<WGM12)|(1<<CS11);
		TCNT1 = 0;
		OCR1A = 1250;
		TIMSK1 |= (1<<OCIE1A);
		sei();
}

// Vid varje interrupt från match på timer1 adderas 10ms
ISR (TIMER1_COMPA_vect){
		ms+=10;
		if(ms>999){
				ms=0;
				sec++;
				if(sec>59){
						sec=0;
						min++;
						}
				}
}
```

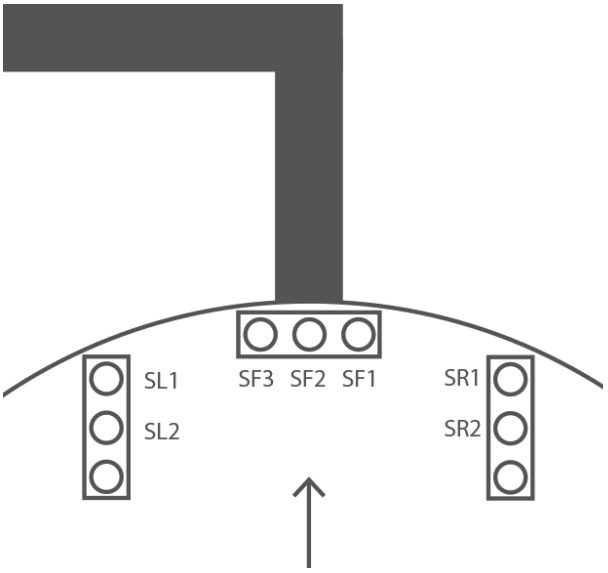
7.3 Sensorernas sekvenser



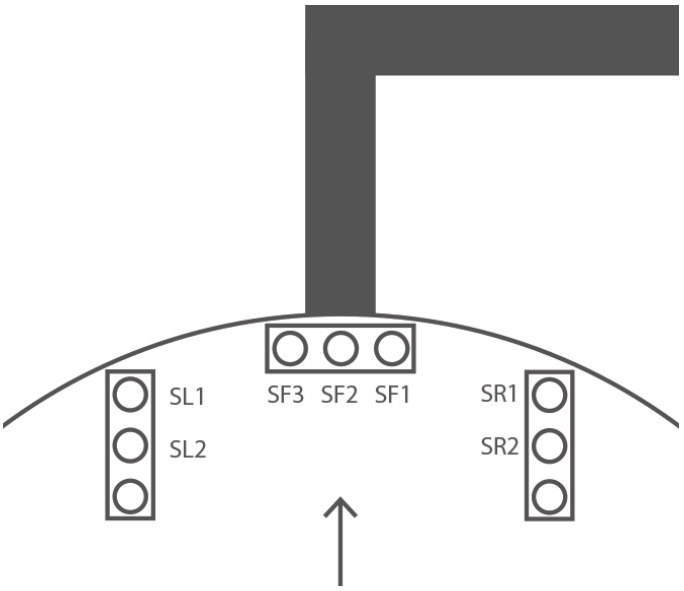
Figur 7. Sekvens fyrvägskorsning



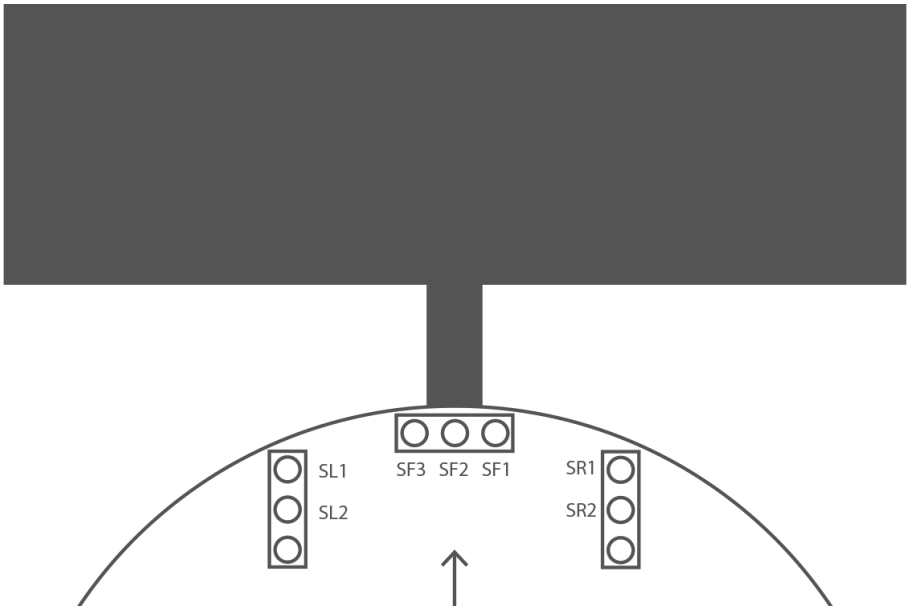
Figur 8. Sekvens trevägskorsning



Figur 9. Sekvens vänstersväng



Figur 10. Sekvens högersväng



Figur 11. Sekvens stopp

Förkortningar

SF - Sensor front (1, 2, 3)

SL - Sensor left (1, 2)

SR - Sensor right (1, 2)

Sekvens fyrvägskorsning, figur 7

SF: 010 → 111 → 010

SL: 00 → 10 → 11 → 01 → 00

SR: 00 → 10 → 11 → 01 → 00

Sekvens trevägskorsning, figur 8

SF: 010 → 111 → 000

SL: 00 → 10 → 11 → 01 → 00

SR: 00 → 10 → 11 → 01 → 00

Sekvens vänstersväng, figur 9

SF: 010 → 110 → 000

SL: 00 → 10 → 11 → 01 → 00

SR: 00

Sekvens högersväng, figur 10

SF: 010 → 011 → 000

SL: 00

SR: 00 → 10 → 11 → 01 → 00

Sekvens stopp, figur 11

SF: 010 → 111

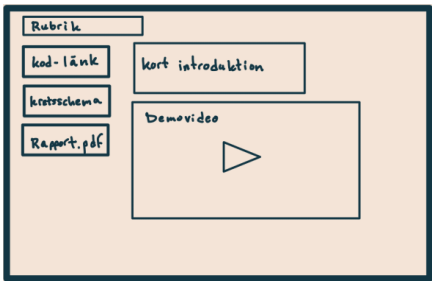
SL: 00 → 10 → 11

SR: 00 → 10 → 11

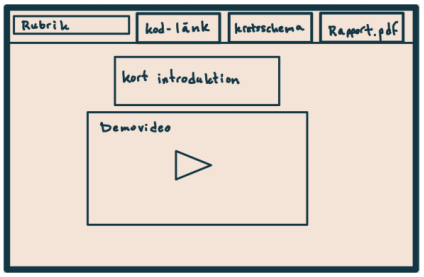
7.4 Mallar för hemsida

Mallar som har tagits fram för att basera hemsidan på.

Vertikal



Horisontell



Bubbelformat