Teknisk dokumentation

Projektgrupp 13

14 december 2022

Version 1.0

Status

Granskad	Zackarias Wadströmer	2022-12-14
Godkänd		2022-12-14

Projektidentitet

Grupp E-post: TSEA29_2022HT_E7-Grupp13@groups.liu.se

Hemsida: https://gitlab.liu.se/da-proj/microcomputer-project-laboratory-d/2022/g13

Beställare: Anders Nilsson, ISY, Linköpings universitet
Tfn: 013-28 26 35
E-post: anders.p.nilsson@liu.se

Kund: Anders Nilsson, ISY, Linköpings universitet
Tfn: 013-28 26 35
E-post: anders.p.nilsson@liu.se

Handledare: Peter Johansson
Tfn: 013-28 1345
E-post: peter.a.johansson@liu.se

Kursansvarig: Anders Nilsson, ISY, Linköpings universitetn
Tfn: 013-28 26 35
E-post: anders.p.nilsson@liu.se

Projektdeltagare

Namn	**Ansvar**	**Telefon**	**E-post**
Linus Thorsell	Projektledare	0765612171	linth181@student.liu.se
Oscar Sandell	Testansvarig	0709416866	oscsa604@student.liu.se
Hannes Nörager	Utvecklare	0733118779	hanno696@student.liu.se
Johan Klasén	Dokumentansvarig	0730982555	johkl473@student.liu.se
Zackarias Wadströmer	Utvecklare	0706142029	zacwa923@student.liu.se
Thomas Pilotti Wiger	Konstruktionsansvarig	0761708593	thopi836@student.liu.se

INNEHÅLL

1 Inledning 1
1.1 Den färdiga produkten 1
2 Teori 2
2.1 ”Sobel operator” och ”Gaussian blur” 2
2.2 Sliding windows 2
3 Systemet 3
4 Modulerna 4
4.1 Kommunikationsmodul 4
4.1.1 Hårdvara 4
4.1.2 Start Taxi Script 4
4.1.3 Server Script 4
4.1.4 AVR Kommunikation 4
4.1.5 Pathfinder 4
4.1.6 Grafen och Algoritmen 8
4.2 Sensormodul 10
4.2.1 Odometer 10
4.2.2 Ultraljudsensor 10
4.2.3 Programkod 11
4.3 Styrmodul 12
4.3.1 Pulsmodulering 12
4.3.2 Avbrottsrutiner 13
4.3.3 Mainfunktion 13
4.3.4 Parsern 13
4.3.5 Styrning via PID Reglering 13
4.3.6 Hastighet via PID Reglering 13
4.4 Datorseende 15
4.4.1 Bildomvandling 15
4.4.2 Tolkning av bild 16
4.5 Styrning från bild 17
5 Slutsatser 18
5.1 Datorseende 18
5.2 PID Reglering 18
5.3 Kommunikation 18
A Kopplingsschema 20

DOKUMENTHISTORIK

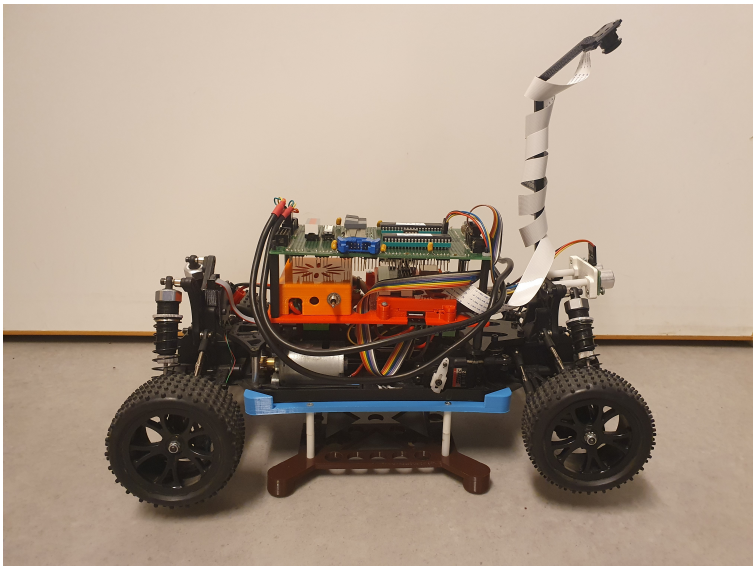
Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2022-12-14	Första versionen	Grupp13	Johan Klasén

1 INLEDNING

Projektet gick ut på att projektgruppen skulle utveckla en taxibil som uppfyller ett flertal krav vilka förhandlades fram emellan kunden och projektgruppen. Dessa krav finns listade i dokumentet ”Kravspecifikation” [1].

1.1 Den färdiga produkten

Projektets produkt är en bil som klarar av att köra autonomt igenom ett vägnät. Bilen ska agera taxi, där den hämtar upp en passagerare vid ett stopp och lämnar av dem vid ett annat. Den använder en extern applikation för att ta in styrparametrar och köruppdrag, vilket den använder för att räkna ut hur den ska navigera igenom vägnätet. Bilen är utrustad med sensorer så som kamera och avståndsmätare för att kunna läsa av sin omgivning och köra säkert på vägen.



Figur 1: Bild på den färdiga produkten.

2 TEORI

Systemet utnyttjar ett flertal kända metoder inom bildbehanling för att kunna navigera sig igenom vägnätet via ett kameraflöde. Merparten av dessa är implementerade med hjälp av externa ramverk som till exempel *”OpenCV”*. Teorin för metoderna som implementeras beskrivs här.

2.1 ”Sobel operator” och ”Gaussian blur”

För att bearbeta bilder i systemet används ”Sobel operator” och ”Gaussian blur” vilket båda utnyttjar filter som ändrar färgen av varje enskild pixel i en bild. De applicerar en matris som kallas för filter och lägger den över varje pixel i bilden för att avgöra dess värde. En pixelns färg blir då en viktning av närliggande pixlar där varje medräknad pixels vikt bestäms av filtret. Genom att välja olika matriser kan olika effekter uppnås och är huvudsakliga metoden för att applicera ”Sobel operator” som hittar kanter och ”Gaussian blur” som smetar ut bilden. För att minska fördröjningen för systemet implementeras dessa genom *”OpenCV”* som redan har skapat en snabb och fungerande lösning av dessa.

2.2 Sliding windows

I datorseendet krävs det att kantlinjernas bana hittas. De pixlar i bilden som representerar någon kant fås från tidigare steg men för dela upp dem för respektive väglinje implementeras *”Sliding windows”*. Denna teknik är simpel och skapar tillräckligt bra fördelningar vilket är lämpligt för systemet då det inte klarar av särskilt tunga processer. Den hittar pixlar tillhörande en linje så den används en gång per vägkant.

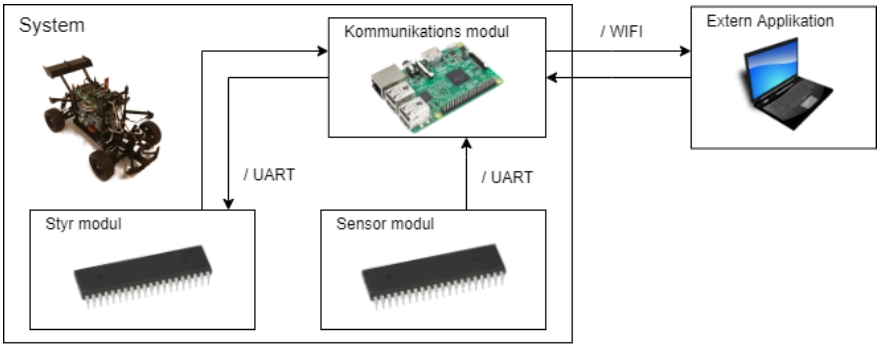
”Sliding windows” tar en två dimensionell matris fylld av intressanta värden och andra värden, i systemet representerade av ettor respektive nollor. Först hittas den troligaste starten av en linje genom att kolla vilken kolumn som har störst förekomst av ettor, denna kolumn utgör procedurens startpunkt. För vägkanterna hittas första vägkantens startposition genom att inspektera bildens vänstra halva och den andra vägkantens startposition genom den högra halvan.

Sedan startar proceduren med att hitta alla punkter med ettor i ett fönster mitt mitten på den kolumnen. Alla dessa punkter läggs till punkter för vägkanten och av dem i fönstret räknas en medelposition ut varav dess kolumn används för att placera mitten av nästa fönster som placeras precis över det aktiva fönstret. I det nya fönstret repeteras processen tills bildens topp har nåtts.

Om man ritar upp fönstrena ser det ut som att dem glider på varandra och på så vis inkapslar alla intressanta pixlar för vägkanten, därav namnet ”Sliding windows”.

3 SYSTEMET

Bilen är uppbyggd utav tre separata moduler som samverkar genom att skicka data mellan varandra. En av modulerna, *Kommunikationsmodulen*, skickar även data till och tar emot data ifrån den externa applikationen som tillåter en användare att styra bilens beteende. De andra två modulerna *Styrmodulen* och Sensormodulen består av två AVR mikroprocessorer av varianten ATmega 1284p.
Sensormodulen ansvarar för att samla in data från de externa sensorer som används och styrmodulen ansvarar för bilens framdrivning genom att styra bilens styrservo och drivmotor. Data från sensormodulen som ska till styrmodulen skickas igenom kommunikationsmodulen och därav utgör kommunikationsmodulen bilens centrum.



Figur 2: Övergripande blockschema på systemet.

De olika delarna i systemet kommunicerar med varandra genom att skicka data över UART. Datan som skickas över UART följer ett specificerat format som beskrivs i Tabell 1.

Commando	Label	Content	Target	Source
keypresses	kp:	f=<0/1>:l=<0/1>:b=<0/1>:r=<0/1>:	Control Module	App
emergencystop	es:	stop if label received	Control Module	ALL
debugpid	db:	esp=<int>:sp=<int>:est=<int>:st=<int>:	App	Control Module
switchmode	sm:	m=<auto(0)/manual(1)>:	Control Module	App
speed sendpid	spp:	p=<int>:i=<int>:d=<int>:	Control Module	App
steering sendpid	stp:	p=<int>:i=<int>:d=<int>:	Control Module	App
error	er:	st=<int>:sp=<int>:	Control Module	OpenCV
telemetry	tm:	s=<int>:d=<int>:	Control Module	Sensor
mission	mi:	s=<ascii>:p=<ascii>:e=<ascii>:	OpenCV	App

Tabell 1: Formatet av textsträngarna som skickas mellan de olika modulerna

4 MODULERNA

Alla moduler i systemet är utbytbara så att en modul som uppfyller samma kommunikations protokoll kan ersätta en modul i systemet. Nedan följer en beskrivning av modulernas olika delar och funktionalitet.

4.1 Kommunikationsmodul

Pågrund av valet av hårdvara utgör kommunikationsmodulen centrum för mycket av systemets funktionalitet.

4.1.1 *Hårdvara*

Kommunikationsmodulens hårdvara består av en processorkärna på en Raspberry Pi 3b. Detta ger den tillräckligt med prestanda för att kunna agera kommunikationsserver mellan alla moduler, även till den externa applikationen som kommunicerar över wifi. Då raspberry pi:n kan kopplas direkt till kameran och kan klara av mera arbete ansvarar den även för datorseendet.

4.1.2 *Start Taxi Script*

Detta är ett python script som startar alla system som körs på kommunikationsmodulens mikrodator. Detta script startas vid startup via en Systemd Daemon, som även agerar förälder till alla underprocesser som skapas. Efter alla processer startats dödas scriptet. Scriptet ansvarar även för att identifiera vilken modul som har vilken UART-port via ett simpelt handskak. AVR skickar vad den är, exempelvis *"Communication-Module"* via UART och scritpet skickar tillbaka *"ACK;"* och sparar vilken modul det var som var kopplad på den UART porten. Sedan startas alla följande script som körs på kommunikationsmodulen: Server Script, AVR Kommunikation (på båda UART portarna) och Datorseende / Pathfinding scripten.

4.1.3 *Server Script*

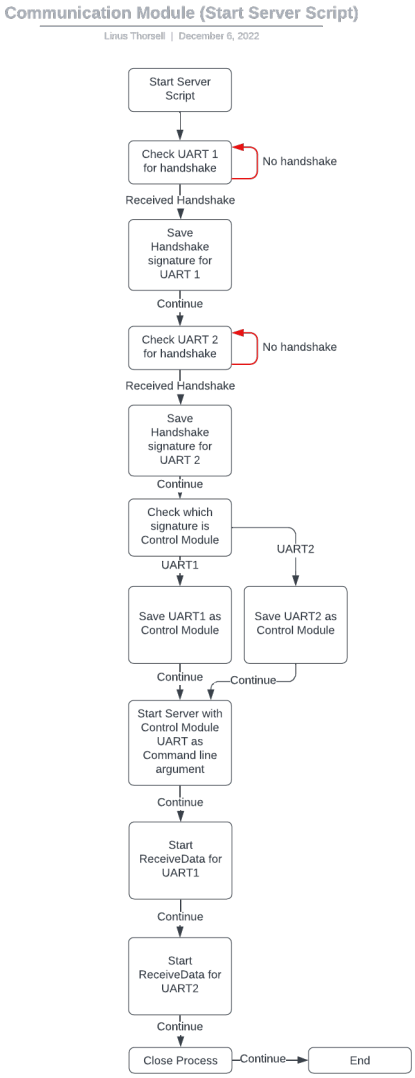
Detta Python script är huvudscriptet som hanterar all kommunikation mellan Sensor-, Styr-, och Kommunikationsmodul samt WebApp, och ett blockschema för programmet kan ses i Figur 4. Det startas med ett kommandradsargument som är vilken UART port som kontrolmodulen är kopplad till. Varje gång servern får in-data via UART eller via en Socket så kollar den vilket label som sitter i början av strängen och skickar endast datan till de som faktiskt behöver använda den. Ett exempel på detta är att resterande kommunikationsmodul inte behöver få emergencystop kommandot, så det skickas bara till kontroll modulen.

4.1.4 *AVR Kommunikation*

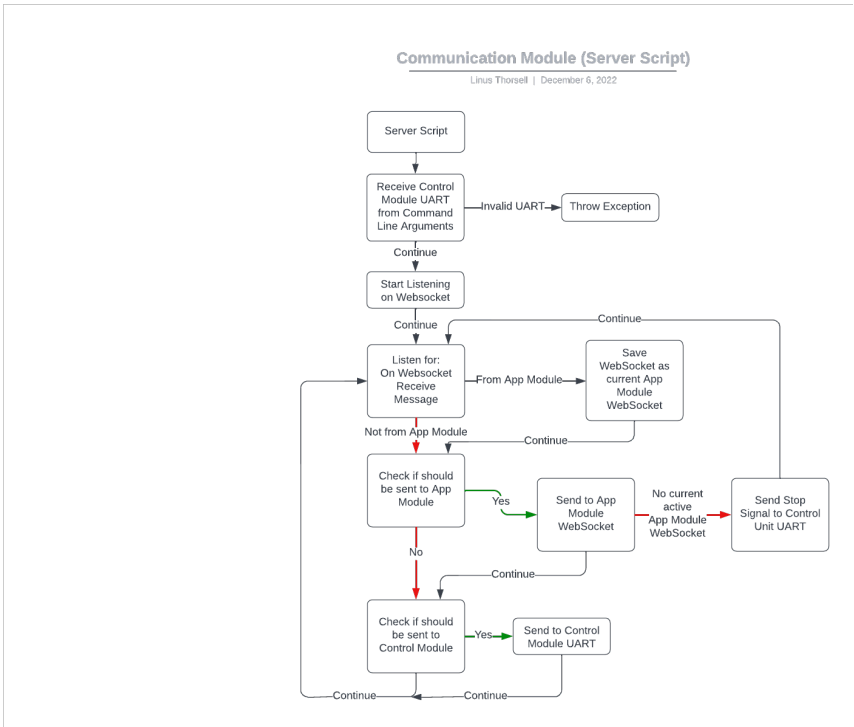
Kommunikationen mellan kommunikationsmodulen och sensor- respektive styrmodulen körs via ett Python script. Scriptet öppnar en UART kommunikation med den andra modulen och skickar all relevant data dit den ska, till exempel vidare till Servern via en Socket. Ett blockschema för förloppet finns i Figur 5.

4.1.5 *Pathfinder*

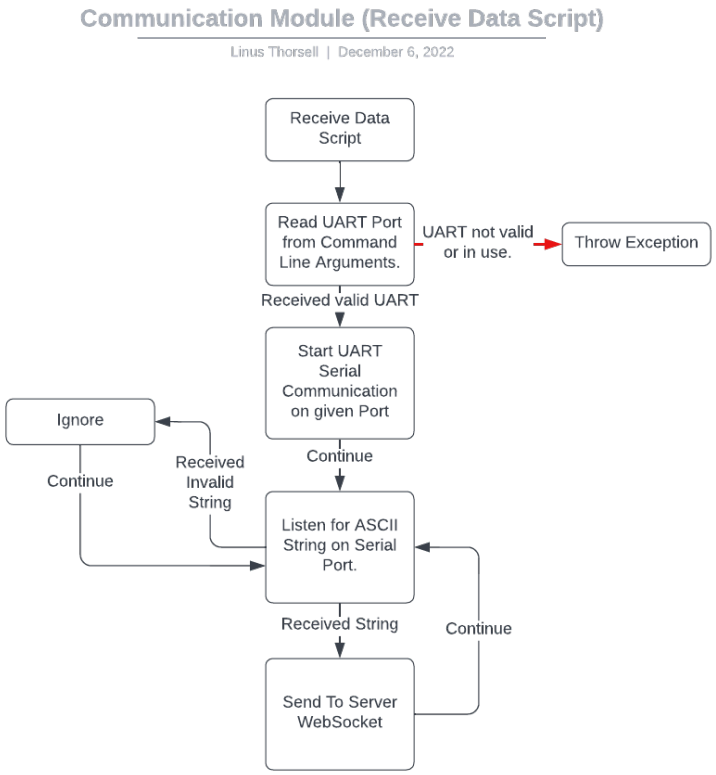
Pathfindern är till för att hitta den kortaste vägen från startplatsen till upphämtningsplatsen och sedan från upphämtningsplatsen till avlämningsplatsen. Pathfindern skrevs i programmeringsspråket python och kan lätt anropas från huvudprogrammet.



Figur 3: Blockschema för kommunikationsmodulens startscript.



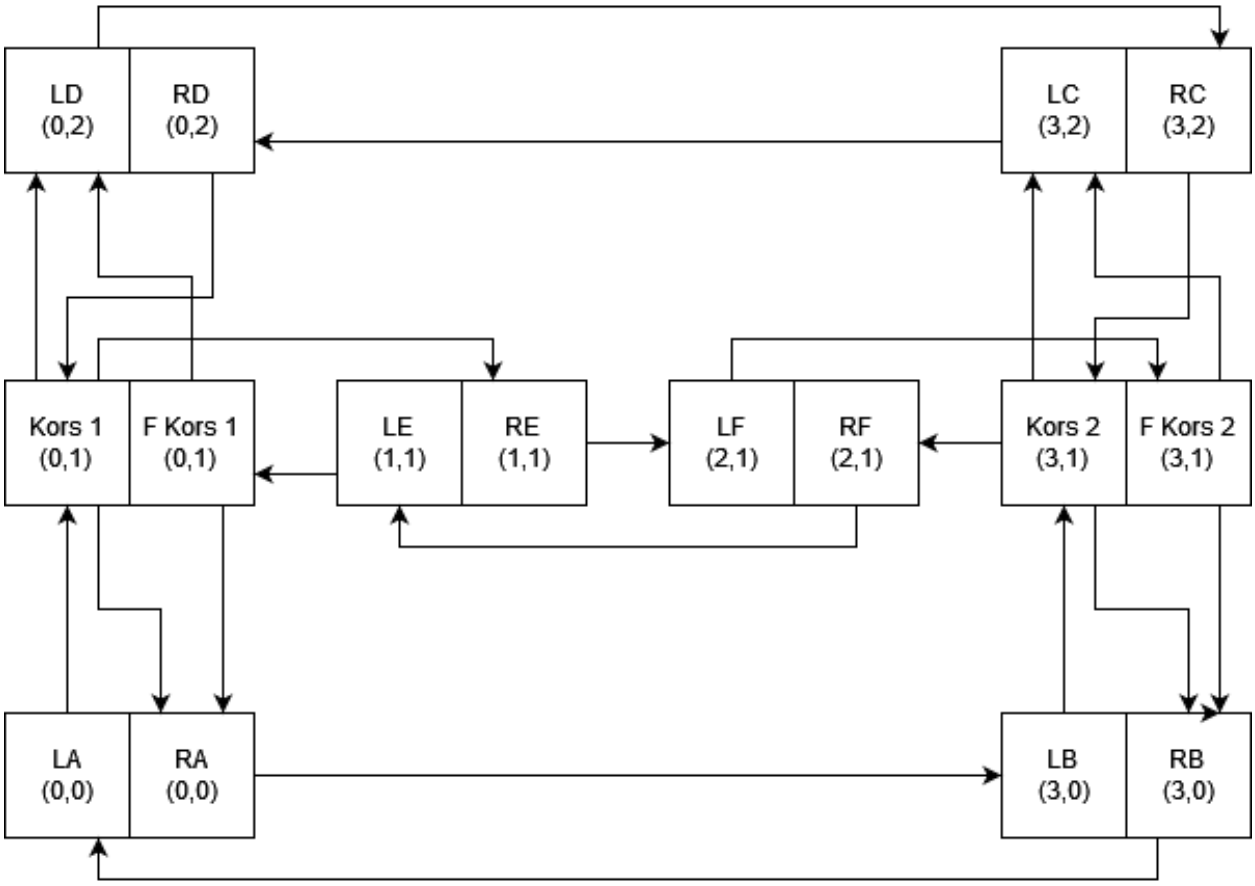
Figur 4: Blockschema för kommunikationsmodulens server.



Figur 5: Blockschema för kommunikationsmodulens AVR kommunikationsscript.

4.1.6 *Grafen och Algoritmen*

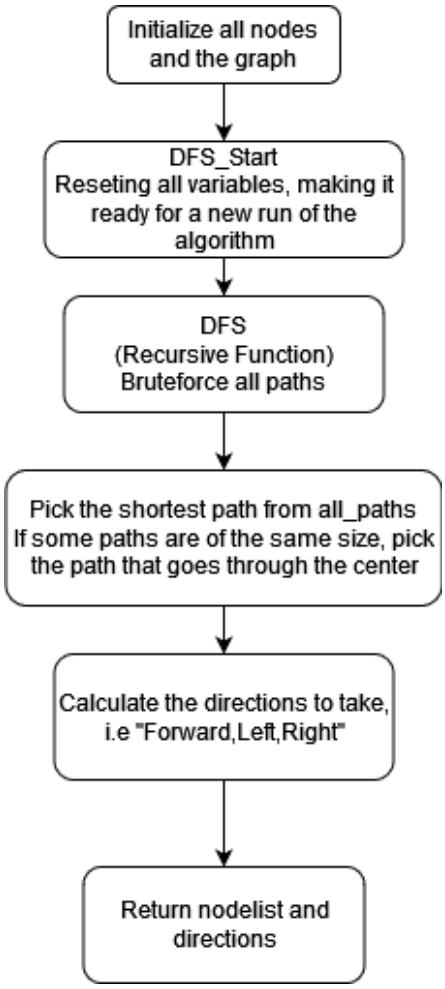
För att lösa problemet med att få fram den kortaste vägen i grafen används en uppsättning noder, där varje plats i banan (Station A, B, C, D, E, F, Kors 1 och Kors 2) representeras av två noder och en uppritad variant av grafen kan ses i Figur 10. Varje nod representerar en position och en körriktning då vi vill kunna vara på samma fysiska plats två gånger per körrunda, detta då bilen skulle kunna vända om och köra tillbaka genom punkten för att komma fram med rätt sida vänd mot målet.



Figur 6: Grafen som används i Pathfindern.

När det kommer till självaste pathfinding-algoritmen används en DFS som söker fram alla vägar som når till målet, för att sedan välja den väg som är kortast. Om två vägar är lika långa så väljs den väg som går igenom ”E” och ”F” vilket garanterar den kortaste vägen i denna graf. Anledningen till att DFS valdes var för att vi ville ha en rekursiv funktion då det kändes lättare att få fram ett bra resultat. Anledningen till detta var för att vi behövde ett extra kondition i if statementet i DFS när man ska utforska alla noder. I en normal DFS algoritmen så utforskar man bara noden om den noden man ska utforska inte har utforskats tidigare, i detta fall vill vi lägga till ett fall som är att vi utforskar noden om

den noden vi vill utforska inte ligger på samma plats som den tidigare noden vilket då eliminerar problemet att bilen skulle vilja vända om 180°. Programflödet av Pathfindern syns i Figur 10:



Figur 7: Ett simpelt programflöde över Pathfindern.

4.2 Sensormodul

Sensormodulen består av en ATmega 1284P och dess uppgift är att samla och bearbeta den data som bilen genererar när den åker, d.v.s. hastighet via en odometer med hallsensorer och detektion av eventuella föremål framför bilen som detekteras med hjälp av en ultraljudsensor. Datan skickas vidare i jämna intervaller till kommunikationsmodulen via UART som sammanställer den med data från övriga moduler och för den vidare till styrmodulen och webb-applikationen.

4.2.1 *Odometer*

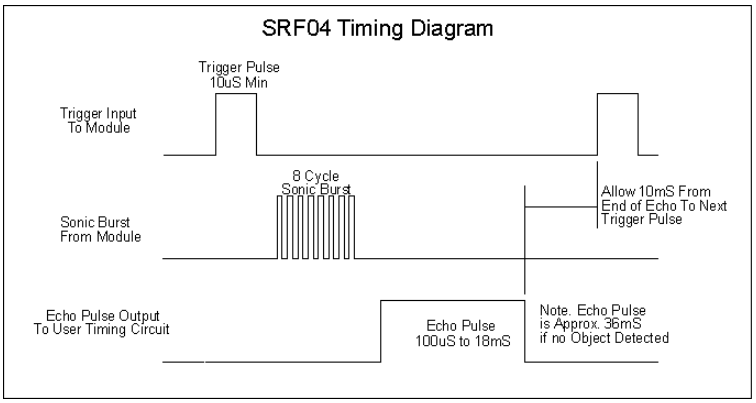
Bilen är utrustad med två halleffektsensorer monterade i anslutning till bakhjulen. På bakhjulen sitter tio magneter placerade med jämnt mellanrum och med hjälp av dessa kan en hastighet mätas upp. När ett tillräckligt starkt magnetfält passerar hallsensorn ger den utslag och ger en hög utsignal. Signalen är kopplad till en extern avbrotts-pin (INT1) på AVR:en. Tillhörande avbrottsrutin är sedan inställd att köras på varje stigande flank på signalen och ökar varje gång en räknare som håller koll på hur många gånger (antal ”tick”) en magnet har passerat sensorn under ett visst intervall. Detta används sedan för att räkna ut hastigheten som frekvensen av magneter per tidsintervall.

4.2.2 *Ultraljudsensor*

Ultraljudsensorn består av en SRF04 och fungerar genom att skicka ut ekopulser som kan studsa på eventuella föremål framför sensorn och sedan mäta tiden det tar för ekot att studsa tillbaka för att på så sätt avgöra avstånd till föremålet. Sensorn styrs med signalen *Trigger Input* och tid fås via signalen *Echo Pulse Output* och timingen kan studeras närmre i Figur 8. En mätning startas via att en 10 s lång puls skickas till *Trigger Input* för att signalera starten på en mätning. En kort tid därefter skickar ultraljudsensorn ut eko-pulserna och drar *Echo Pulse Output* hög för att visa att den har börjat lyssna på studset av ekot. När ekot når tillbaka (eller man når en timeout på 36 ms) dras *Echo Pulse Output* låg igen och bredden på pulsen som skickas motsvarar alltså avståndet till föremålet. För att underlätta implementeringen mäts dock den totala tiden räknat från den fallande flanken på startpulsen fram tills fallande flank på Echo Pulse Output.

Detta hanteras via AVR:en genom att koppla *Trigger Input* till en pin länkat till en intern timer (OC0A) och *Echo Pulse Output* till en extern avbrotts pin (INT2). Timer0 har ställts in så att den skickar hög output i 10 s och sedan stänger av sig själv för att hantera startsignalen till *Trigger Input* och sätter en starttid för mätning. Och för *Echo Pulse Output* är INT0 inställt så att man får en interruptrutin på fallande flank för att få en sluttid på mätningen.

Det uppmätta värdet används sedan helt enkelt som kollisionsdetektion för bilen. Blir den uppmätta tiden tillräckligt låg för att hamna under ett satt gränsvärde vet man att det finns föremål framför bilen och att den behöver stanna, men denna tolkning görs dock i styrmodulen.

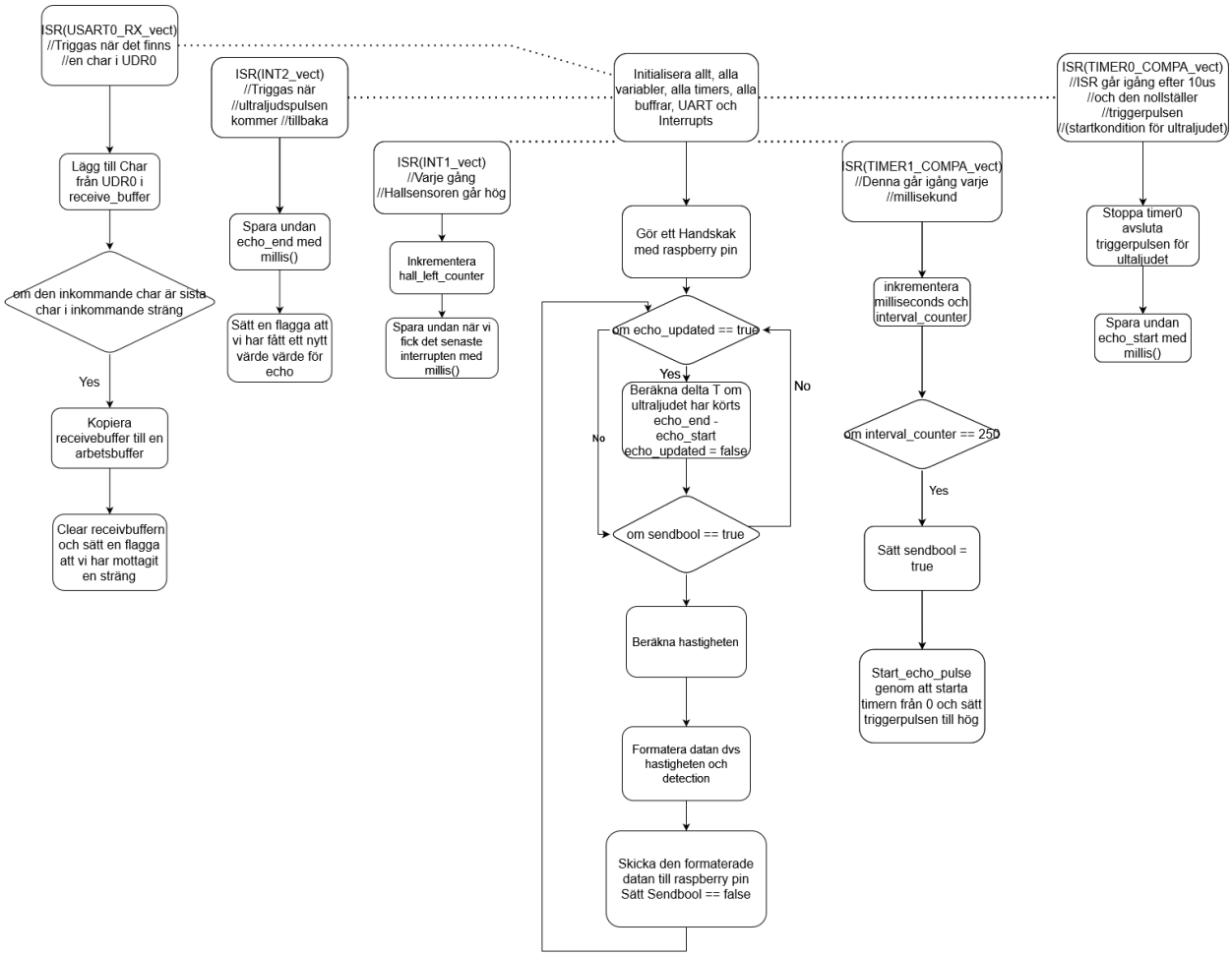


Figur 8: Tidsdiagram för ultraljudsensorns signaler. *Trigger Input* innebär starten av en mätning där sensorn svarar med att dra *Echo Pulse Output* hög så fort den är redo att lyssna på ett eko och låg igen efter 36 ms (timeout) eller så fort ekot har tagits emot, vilket som kommer först.

4.2.3 *Programkod*

Vid uppstart av modulen körs först en initieringsrutin som konfigurerar och startar bland annat portinställningar, minnesbuffertar, timers och och de fem avbrottsrutinerna. Den första avbrottsrutinen är en timer kopplat till AVR:ens interna räknare Timer1. Den körs en gång i millisekunden från och med start och används för att kolla hur mycket tid som har förflutit och på så sätt räkna ut tillsskillnader. Den ser också till att vid satta tidsintervall starta Timer0 för att aktivera ultraljudssenorns startpuls. En andra avbrottsrutin ansvarar sedan för att stänga av Timer0 och avsluta startpulsen. Den tredje avbrottsrutinen hanterar mottagning av data från UART och de sista två hör till respektive sensor och har nämnts ovan.

Huvudprogrammet är inte särskilt stort, den har ett handskak i början som upprepat skickar ”sensor_module” till kommunikationsmodulen som sen skickar ett ”ACK” tillbaka. Därefter hamnar sensormodulen i en inre loop där den i satta tidsintervall beräknar tiden det tar för ljudvågen att färdas fram till ett hinder och sedan tillbaka till sensorn om den har fått färsk data. Loopen beräknar sedan även ut hastigheten och sammanställer all data för att sedan skicka den vidare till kommunikationsmodulen enligt specificerad standard i Tabell 1.



Figur 9: Blockschema över sensormodulen

4.3 Styrmodul

Styrmodulen består av en ATmega 1284P som är kopplad till Raspberry Pin via UART och är även kopplad till bilens motor och styrservo via ett kontaktdon i skelettet. Vi använder oss utav pulsmodulering för att styra motorn och styrservot.

4.3.1 *Pulsmodulering*

Bilens motor och servo styrs pulsmodulering. Detta sker med hjälp av styrmodulens inbyggda timrar och pulsbreddsmodulatorer. Pulserna kommer från Timer1 och Timer3, den ena timern har en periodtid på 0.5 ms och styr motorn medans den andra har en periodtid på 20 ms och styr styrservot. Pulsbredden sätts genom att skriva till registret OCR1A respektive OCR3A. Pulsen som styr motorn kommer från port PB5 och den som styr styrtservot kommer ifrån PB6 kopplade till bilens skelett.

4.3.2 *Avbrottsrutiner*

Styrmodulen har två avbrott med tillhörande avbrottsrutiner. Den ena avbrottsrutinen är relaterat till UART och tar en char ifrån motagningsregistret UDR0 och sparar charen i en buffer. Avbrottet triggas när det finns data i UDR0 vilket avgörs av RX pinnen blir hög vilket då triggar avbrottet. Den andra avbrottsrutinen är en intern avbrottsrutin och aktiveras varje gång det har gått 1 millisekund och används till att få en fungerande millis() funktion så att vi kan få hur många millisekunder det har gått sedan programmet startades.

4.3.3 *Mainfunktion*

Styrmodulens huvudprogram består av tre delar, Den första delen av programmet initierar variabler, interrupts och buffertar. Den andra delen är en handskakning där styrmodulen upprepat skickar ”control_module” till Raspberry Pin så att den kan avgöra vilken som är Styrmodulen och den tredje delen av mainfunktionen är en whileloop som har hand om allt annat. Det som den sista whileloopen tar hand om är att parsea meddelandet som kom från Pi’n och sedan baserat på detta gör den olika kommandon såsom att köra i olika riktningar, gasa och stoppa.

4.3.4 *Parsern*

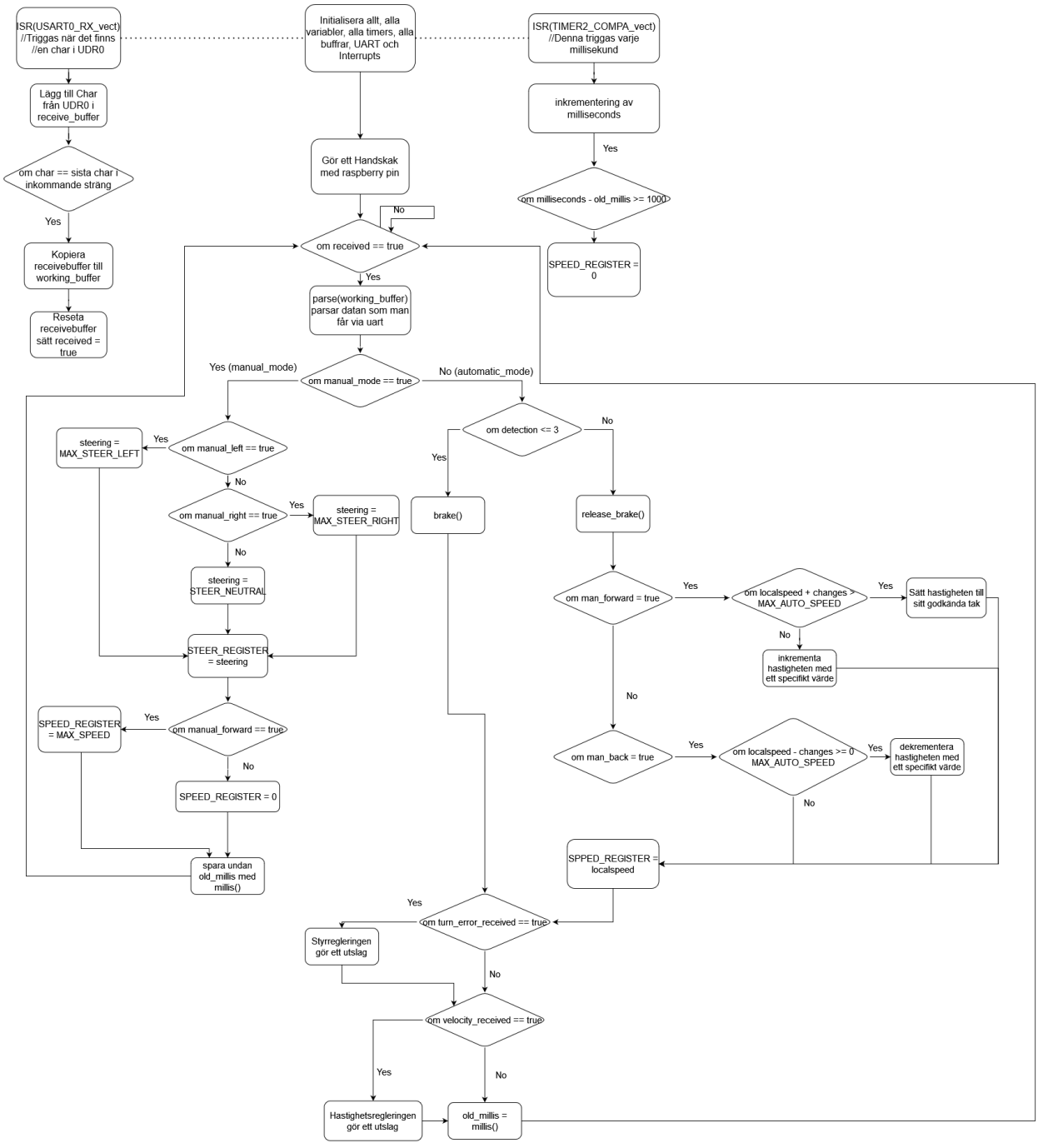
Parsern får data i form av en sträng via UART sedan plockar styrmodulen ut den relevanta datan som finns i strängen. Datan är i formetet som syns i tabell 1 på sida 3. Denna data används för allt ifrån till att styra bilen, gasa, få info om eventuella hinder och får även hastigheten.

4.3.5 *Styrning via PID Reglering*

För att hålla sig på vägen vid autonom körning kör styrmodulen en simpel PID loop. Denna körs varje gång modulen får ett nytt fel från kommunikationsmodulen via UART. Det betyder att regleringen körs så ofta den kan på det nya felet.

4.3.6 *Hastighet via PID Reglering*

För att hålla en konstant hastighet under körning så har bilen en PID loop som uppdaterar hastigheten varje gång sensormodulen skickar data till kontroll modulen, detta ger om den tunas för det en mycket mjuk start och mycket mjuk stopp vid upphämtning / avlämningsplatserna.



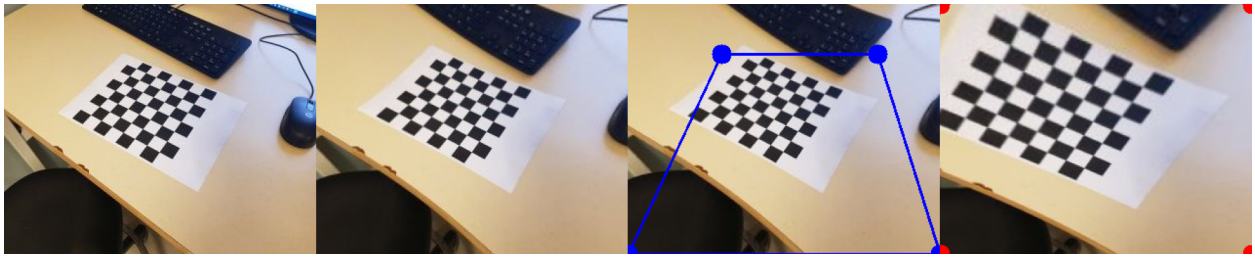
Figur 10: Ett detaljerat blockschema över styrmodulen.

4.4 Datorseende

För att navigera sig efter vägen använder bilen sig av datorseende i form av ett script där den kontinuerligt tar bilder och bearbetar dem för att producera fel i form av ett numeriskt värde och sedan förse styrmodulen med den. Utöver kontrollerar scriptet parametrar i samarbete med ”pathfindingen” som berättar hur scriptet ska hantera särskilda sammanhang och om styrmodulen ska stanna. Dessa är i syfte att navigera sig genom vägnätet och särskilt att hantera korsningar korrekt.

4.4.1 *Bildomvandling*

För att tolka bilden börjar scriptet med att omvandla den så att endast relevant information visas upp och på ett relevant sätt. Detta görs i separata steg som representeras av egna funktioner i koden.



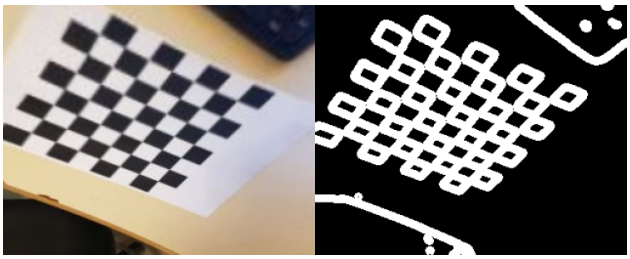
Figur 11: Bildsekvens som visar på omvandlingarna som görs. Första bilden är obehandlad. Andra har tagit bort böjning i linsen. Tredje har ’region of interest’ utmarkerats och sista är transformerad för att det området ska täcka skärmen

Först förvrängs bilden för att dess böjning ska försvinna. Denna korrigering är beroende av kamerans lins och dess inställningar och sker genom att omvandla bilden med en förvrängnings matris som beskriver bildens böjning. Då parametrarna för en särskild kamerauppsättning är demsamma räknas dessa ut utanför aktiv exekvering utifrån en uppsättning kalibreringsbilder. Parametrarna sparas sedan undan i en dedikerad fil som sedan hämtas när bilden ska förvrängas under exekvering.

Näst förvrängs bildens perspektiv. Detta är viktigare för att hantera linjer korrekt och att deras tjocklek samt avstånd från varandra ska representera vägens egenskaper mera korrekt. Detta görs genom *”OpenCV”* som tar koordinater som beskriver det område som ska vridas upp.

Efteråt hanteras bilden så att kanter markeras ut. Först plockas pixlar med låg färg ut. Dessa representerar vägmarkeringar då dem är svarta och markeras som ettor medan allt annat blir nollor. Sedan används ”Gaussian blur” för att göra bilden suddig då detta hjälper i nästa steg där alla kanter markeras med hjälp av Sobel operatorn. En suddig bild har jämnare kanter och minskar små störningar som kan framkomma i bilden.

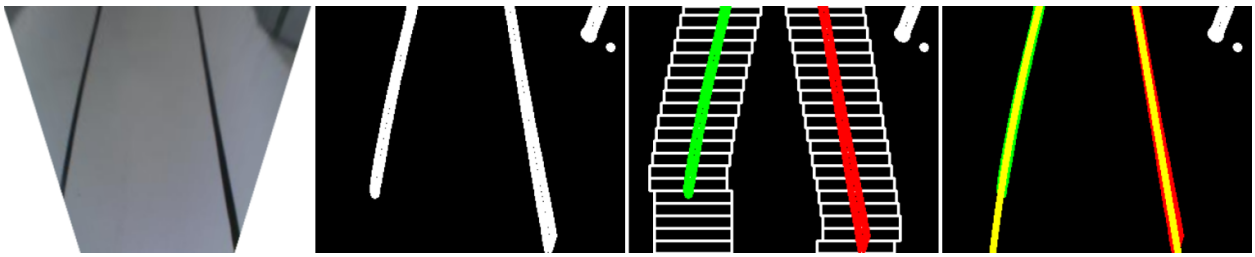
Efter all bildomvanlding har bilden omvandlats till en bitmap där ettor utgör vägmarkeringar.



Figur 12: Bildsekvensen visar konverteringen av en bild. Första bilden är den ingående bilden och den andra är där kanter har markerats.

4.4.2 *Tolkning av bild*

Utifrån en bitmap där ettor markerar vägmarkingar utförs olika beräkningar för att producera de styrparametrar som sedan förs till styrmodulen.



Figur 13: Bildsekvens som visar på hur en bild tolkas. Första bilden är den bild som ska tolkas efter bildomvandling, som visat i figur 11. Det vita området är tillagt för att perspektivet ska bli korrekt men inte kapa bort linjerna. I andra bilden har den omvandlats till en bitmap, som i 12. I tredje används sliding windows för att finna de pixlar som tillhör en av de två vägkanterna som målats i två olika färger. Sista bilden visar linjerna som fås över punkterna för respektive vägkant.

Först markeras positionen av den vänstra och högra väglinjen. Detta görs genom att förekomsten av ettor i varje kolumn markeras ut i ett histogram och sedan får högsta förekomsten i kolumnens första halva markera den vänstra kantens start och högsta förekomsten i den andra halvan för den högra linjens kant.

Startpunkterna används sedan för att beräkna linjerna i form av andragradskurvor. Detta görs med ”sliding window” tekniken där förekomsten av punkter inom ett mindre fönster läggs till i punkter för respektive linje och sedan flyttas fönstret upp så att det placeras över det förra fönstret och är centrerad runt förekomsten av punkter i den. Detta repeteras till toppen av skärmen nås. För att finna de två kantlinjerna genomförs processen två gånger om. För varje kantlinje startas processen med att placera första fönstret vid startpunkten för respektive kantlinje, hämtat från tidigare uträkningssteg. Alla punkter för en kantlinje används för att räkna ut en andragradskurva som passar dem så bra som möjligt och representerar vägen.

Utifrån de två kurvorna räknas två vridgningsfel ut, vilket är hur mycket bilen måste rotera för att träffa banan igen och hur mycket den måste rotera för att ligga parallellt med banan. Eftersom dessa fel ska representera hur bilen ska röra sig framåt förhåller uträkningarna sig till en punkt längre bort på vägen. Exakta avstånd definieras i koden. Felet

räknas ut i radianer alltså den exakta vinkeln som bilen måste korrigera.

Sedan tas de två vridningsfelen för att räkna ut ett enda fel som kan skickas till styrmodulen. Detta fel ska representera hur mycket bilen ska vridas.

4.5 Styrning från bild

För att kunna navigera i vägnätet så arbetar datorseendet tillsammans med pathfinding. Från pathfinding får datorseendet information om vilken kant som ska följas i olika skeden i körningen. Under normal körning påverkar båda linjerna bilens styrning men när bilen stöter på en korsning används enbart en. Datorseendet får information om vilken linje som ska följas och kan därmed strunta i den felaktiga linjen och köra åt rätt håll. Vid sväng kan båda linjer hamna utanför kamerans synfält och då ges styrhjälp i form av dödkörning tills dess att linje hittats.

5 SLUTSATSER

5.1 Datorseende

Som datorseendet är utformat just nu måste varje bild gå igenom ett antal processer för att producera styrdata. Dessa har en minimum tid de kan genomföras på för en viss bilduppsättning och efter att ha arbetat med att optimera dem är det klart att denna tidsgräns är ganska stor.

Det lättaste sättet att minska tiden är genom att minska bildstorleken men det resulterar i förlust av data som förvärrar bilens fel. Detta eftersom linjerna ska passas emot färre punkter och av dessa kommer inte lika många ligga på vägkanten som därav inte kommer väga lika tungt i beräkningarna och kunna resultera i större fel. För att få tillräcklig upplösning men ändå bra tid testades bildens storlek fram men en lämpligare storlek är nog möjligt.

5.2 PID Reglering

Med hjälp av datorseendets felvärde kan kontroll modulens PID loopar både hålla bilen i konstant hastighet som bestäms av datorseendet och även reglera styrningen för att hålla sig mitt i vägen.

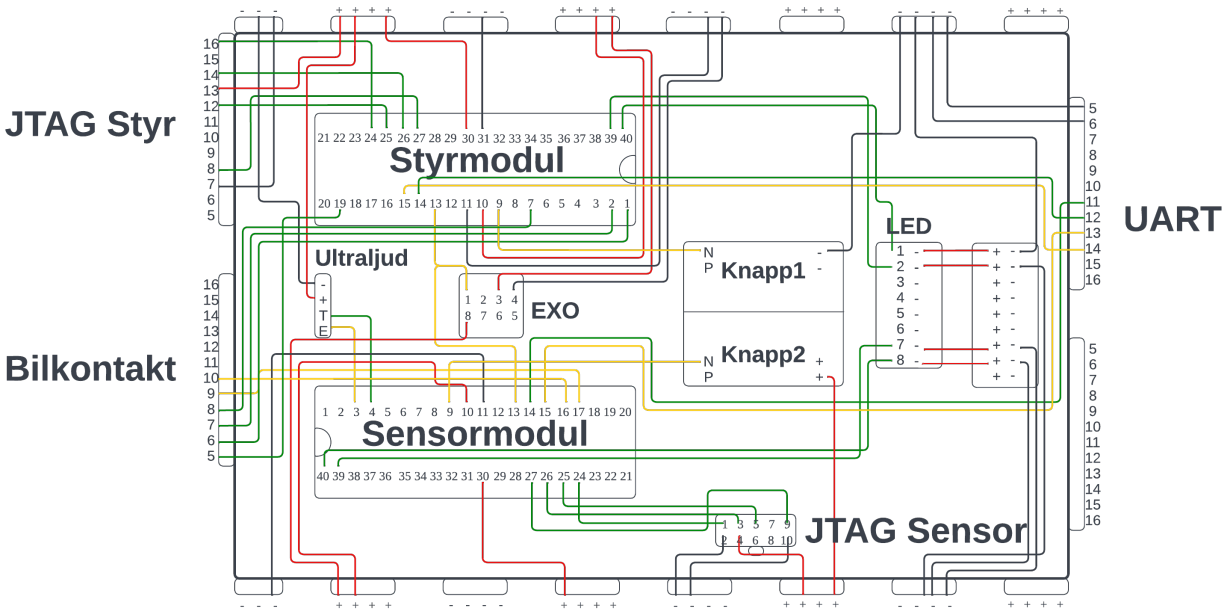
5.3 Kommunikation

Kommunikationen mellan de olika modulerna fungerar bra mellan olika typer av protokoll och överföringstyp, detta på grund av att alla kommandon är baserade på ASCII som alla system på bilen kan förstå. Kommunikationsmodulens huvudserver som använder WebSockets implementerade i Python hanterar all data snabbt och dirigera all data till de mottagarna som skall få denna data.

REFERENSER

[1] Projektgrupp 13, “Kravspecification,” https://gitlab.liu.se/da-proj/microcomputer-project-laboratory-d/2022/g13/docs/-/blob/master/kravspec/Kravspec_1.0_grupp13.pdf, internt projektdokument i samråd med beställaren.

A KOPPLINGSSCHEMA



Tabell 2: Portlista Styrmodul

Från	Namn	Till
1	DIR	Bilkontakt6
2	BRAKE	Bilkontakt7
7	SERVO	Bilkontakt8
9	RESET	Knapp1N
10	VCC	VCC
11	GND	GND
13	CM_XTAL1	EXO1
14	CM_RXD0	UART12
15	CM_TXD0	UART14
19	PWM	Bilkontakt5
24	CM_TCK	JTAG_Styr16
25	CM_TMS	JTAG_Styr12
26	CM_TDO	JTAG_Styr14
27	CM_TDI	JTAG_Styr8
30	VCC	VCC
31	GND	GND

39	LED2	LED2
40	LED1	LED1

Tabell 3: Portlista Sensormodul

Från	Namn	Till
3	Echo	UltraljudE
4	Trigger	UltraljudT
9	RESET	Knapp2N
10	VCC	VCC
11	GND	GND
13	SM_XTAL1	EXO1
14	SM_RXD0	UART11
15	SM_TXD0	UART13
16	INT0	Bilkontakt10
17	INT1	Bilkontakt9
24	SM_TCK	JTAG_Sensor1
25	SM_TMS	JTAG_Sensor5
26	SM_TDO	JTAG_Sensor3
27	SM_TDI	JTAG_Sensor9
30	VCC	VCC
31	GND	GND
39	LED7	LED7
40	LED8	LED8

Tabell 4: UART

Från	Namn	Till
5	GND	GND
6	GND	GND
11	SM_RXD0	Sensor14
12	CM_RXD0	Styr14
13	SM_TXD0	Sensor15
14	CM_TXD0	Styr15

Tabell 5: Portlista Bilkontakt

Från	Namn	Till
1	GND	GND
2	GND	GND
3	VCC	VCC

4	VCC	VCC
5	PWM	Styr19
6	DIR	Styr1
7	BRAKE	Styr2
8	SERVO	Styr7
9	HALL_L	Sensor17
10	HALL_R	Sensor16

Tabell 6: Portlista Ultraljud

Från	Namn	Till
-	GND	GND
+	VCC	VCC
T	Trigger	Sensor 4
E	Echo	Sensor 3

Tabell 7: Portlista EXO

Från	Namn	Till
1	F Output	Styr/sensor 13
2	D	-
3	ST	VCC
4	GND	GND
5	A	-
6	B	-
7	C	-

Tabell 8: Portlista JTAG-styr

Från	Namn	Till
16	TCK	Styr 4
15	GND	GND
14	TDO	Styr 26
13	VTG	Vcc
12	TMS	Styr 25
11	nSRST	Styr 9
10	N.C.	-
9	nTRST	-
8	TDI	Styr 27
7	GND	GND

Tabell 9: Portlista JTAG-sensor

Från	Namn	Till
1	TCK	Sensor 24
2	GND	GND
3	TDO	Sensor 26
4	VTG	Vcc
5	TMS	Sensor 25
6	nSRST	Sensor 9
7	N.C.	-
8	nTRST	-
9	TDI	Sensor 27
10	GND	GND