Designspecifikation

Gustaf Sjögemark

Version 1.0

Version	Datum	Utförda förändringar	Utförd av	Granskad
0.1	2024-10-11	Första utkast	Hela gruppen	Gustaf Sjögemark
0.2	2024-10-16	Komplettering	Hela gruppen	Gustaf Sjögemark
1.0	2024-10-17	Färdig första version	Hela gruppen	Gustaf Sjögemark

Projektidentitet

Grupp E-post: rikag489@student.liu.se

Hemsida: gitlab.liu.se/da-proj/microcomputer-project-laboratory-d/2024/g09/docs

Beställare: Linköpings Universitet

Kund: Mattias Krysander
E-post: mattias.krysander@liu.se
Tfn: +4613282198

Handledare: Theodor Lindberg.
E-post: theodor.lindberg@liu.se

Kursansvarig: Anders Nilsson
E-post: anders.p.nilsson@liu.se
Tfn: +4613282635

Projektdeltagare

Namn	**Ansvar**	**E-post**
Kacper Uminski	Implementationsansvarig (IMP)	kacum383@student.liu.se
Gustaf Sjögemark	Dokumentansvarig (DOK)	gussj945@student.liu.se
Samuel Tuvstedt	Projektledare (PROJ)	samtu593@student.liu.se
Rikard Ågren	Kund/Kommunikationsansvarig (KOM)	rikag489@student.liu.se
Axel Nyström	GUI/Test-ansvarig (GUI)	axeny840@student.liu.se
Alfred Sjöqvist	Organisationell Korrespondent (ORG)	alfsj019@student.liu.se

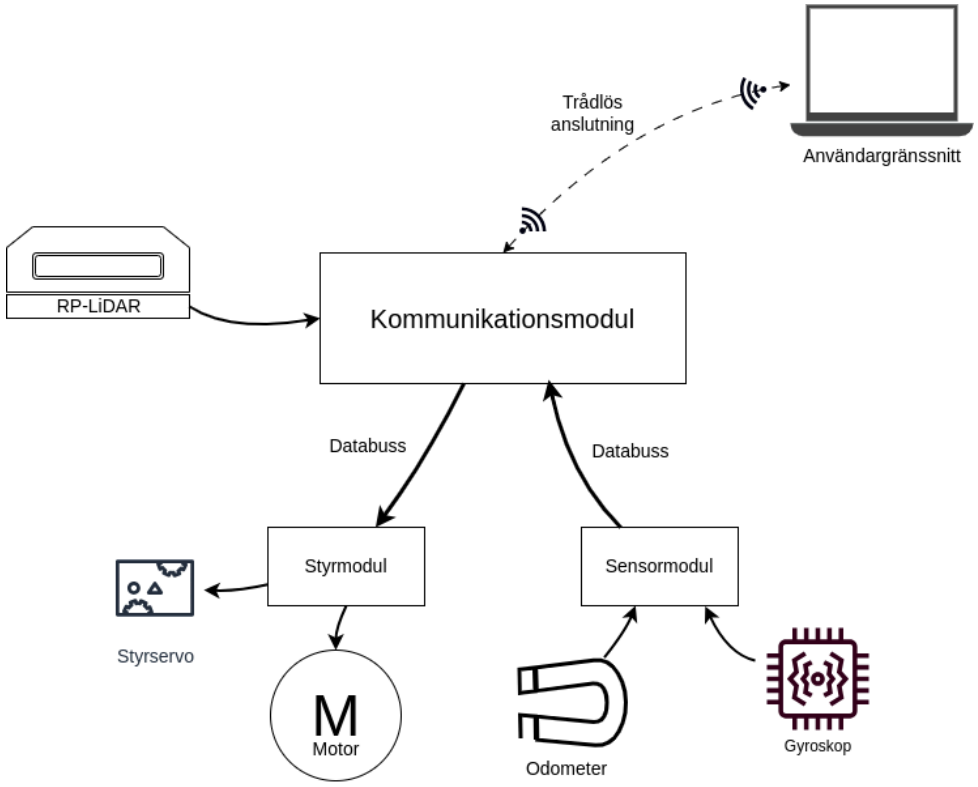
Innehåll

1 Inledning & sammanfattning

I denna del beskrivs en övergripande struktur för bilens uppbyggnad och uppbyggnad av sensorer.

1.1 Systemöverblick

Systemet kommer delas upp i olika moduler som tillsammans skapar bilen. Modulerna kommer vara användargränssnittet, sensormodulen, styrmodulen och kommunikationsmodulen. Användargränssnittet kommer låta utvecklaren styra bilen manuellt och få information, såsom hastighet, tillryggalagd sträcka, vilket varv bilen befinner sig på, med mera. I sensormodulen kommer flera mindre sensorer, såsom hallsensorer och gyroskop, att ingå. Med hjälp av datan från dessa sensorer samt en LiDAR kommer bilen kunna orientera sig och bestämma hur den ska köra den optimala vägen på banan. Styrmodulen kommer har som uppgift att reglera bilens riktning och fart utefter kommandon. Detta kommer göras genom att den kontrollerar en styrservo och drivande motor. Både styr- och sensormodulens klocksignal genereras av en extern oscillator (EXO-3). Kommunikationsmodulen kommer stå i centrum för bilens autonoma funktion och all kommunikation mellan modulerna. Den kommer ha i uppgift att ta in data från LiDAR:n, sensormodulen och användargränssnitt. Baserat på den datan kommer kommunikationsmodulen reglera bilen via styrmodulen. (Se figur 1 för en överblick av systemet.)



Figur 1: Översiktlig systemskiss med samtliga delsystem och gränssnitt.

Sensorerna som kommer användas i systemet är ett gyroskop för att mäta svängar och två hallsensorer för att mäta hastighet på de bakre hjulen. Gyroskopet placeras inuti bilen. Den exakta platsen är inte relevant då gyroskopet kommer att fungera på rätt sätt så länge den sitter på bilen och inte ligger för nära stora strömkällor. Bilens sensorer kommer lokalt styras med en AVR-kontroller (ATmega1284P) [1]. Denna kommer att använda hårdvarubaserade timers för att med jämna mellanrum ta in data från sensorerna. LiDAR:n kommer placeras högst upp på bilen så dess vy inte obstrueras. Motor och styrservo kommer placeras så de kan styra framhjulen och driva bakaxeln. Kommunikationsmodul, styrmodul och sensormodul kommer placeras så de inte är i vägen för rörliga delar och så att de sitter skyddade mot stötar.

1.2 Sensorer

Nedan beskrivs kort de sensorer som används i projektet:

Odometer På båda av bilens bakre hjul sitter det tio magneter. Dessa blir avlästa med hallsensorer. Hjulens rotation triggar pulsar från sensorerna som sedan kan tolkas till en hastighet.

Gyroskop Gyroskopet mäter bilens rotation i det horrisontella planet.

LiDAR LiDAR:n används för att bilen ska kunna läsa av sin omgivning.

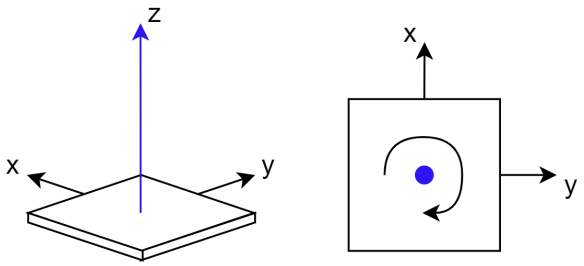
rikag489@student.liu.se

2 Sensormodul

Sensormodulens uppgift kommer vara att med input från sensorerna beräkna hastighet, färdsträcka samt att observera bilens rotation. Denna information ska sensormodulen sedan skicka till kommunikationsmodulen. Sensormodulen kommer bestå av en AVR mikrokontroller (ATmega1284p)@ datasheet:ATmega1284P, ett gyroskop [2], samt två hallsensorer [3], en vid vardera bakhjul.

Hallsensorerna kommer vara kopplade till AVR:ens externa interrupt-portar `INT0` och `INT1` som ligger på pinne `PD2` respektive `PD3` enligt figur 3. Då sensorerna har en öppen kollektor kan de antingen jorda sin utsignal eller släppa den helt. AVR:en kommer därför använda sina interna pull-up resistorer för att sätta pinnarna höga när de inte aktivt jordas. Detta kommer göra det möjligt att läsa från sensorernas output. Resistorerna kan aktiveras genom att sätta `DDRD[2]` och `DDRD[3]` till `0` och sätta `PORTD[2]` och `PORTD[3]` till `1`. Portarna kommer därefter att konfigureras så att ett avbrott triggas på låg flank, vilket sker när en av magneterna i hjulen passerar en hallsensor och dess utsignal blir jordad.

Gyroskop-sensorn kommer vara kopplad till en av AVR:ens portar för AD-omvandling, pinne `PA0` (se figur 3). Gyro-sensorns `'OUTAR'`-pinne skickar ut en analog signal som positivt stiger vid medurs rotation kring axeln normal till sensors platta yta (se figur 2). Denna signal kommer sedan omvandlas till ett digitalt värde i AVR:en. Gyro-sensorns utsignal kan tyvärr påverkas av drift beroende på temperatur. Därför skickar den även ut omgivande temperatur som analog signal `'OUTTEMP'`, vilket kommer användas för att kompensera eventuell drift. Temperaturen kommer tas in i AVR:en på pinne `PA1` för AD-omvandling.



Figur 2:

Till vänster: gyro-sensorn illustrerad i 3D.

Till höger: gyro-sensorn sedd ovanifrån. Pilen kring den blåa pricken illustrerar rotationen som sensorn känner av.

Sensormodulen kommer även ha en JTAG-anslutning[4] för debugg- och programmeringssyften, samt en UART-/FTDI-anslutning[5] för kommunikation med kommunikationsmodulen. Klocksignalen produceras av en EXO-3 extern oscillator.

2.1 Komponentbudget

I tabell 1 återfinns information om vilka komponenter och hur många av vardera som behövs för att konstruera styrmodulen.

Tabell 1: Komponenter i sensormodulen.

Komponent	**Antal**
Atmega1284P	1
Hallsensor (A1120)	2
Gyroskop (MLX90609)	1
JTAG ICE 3	1

2.2 Mjukvara

Nedan beskrivs mjukvarans som implementeras i sensormodulerna.

2.2.1 Timer

AVR:ens interna timers kommer att användas för beräkningar som behöver ta hänsyn till tid, exempelvis hastighetsberäkning och integrering av vinkelhastighet.

2.2.2 Hallsensorer

I listing 1 visas preliminär pseudokod för hallsensorernas avläsning enligt de följande antagandena:

- Hjulens omkrets är känd.
- En interrupt triggas när en magnet passerar en sensor.
- Tiden mellan avbrott mäts.

```
int hall_count_L  //räknar antal avbrott i hall-sensor L
int hall_count_R  //räknar antal avbrott i hall-sensor R
int interrupt_interval
int speed
int distance

main
  Initiera nödvändiga portar, interrupts, etc.

  while-loop med önskad frekvens:
    - beräkna färdsträcka m.h.a antal avbrott och hjulens omkrets
    (Kanske ta genomsnitt av L och R)
    idé: (antal pulser / antal magneter per hjul) * omkrets

    - beräkna hastighet m.h.a tidsintervallet mellan de två senaste avbrotten
    (genomsnitt av L och R)
    idé: hjulens omkrets / (tid mellan de två senaste avbrotten * antalet
    magneter per hjul)

    - skicka data / gör data tillgänglig för komm.-modulen


Avbrottsrutin för hallsensor:   //magnet har passerat sensor
  hall_count_[L/R] ++           //räkna upp antal avbrott
  beräkna tid sen senaste avbrott
```

Listing 1: Pseudokod för hallsensorer

Om vi får problem med heltalsdivision kan vi använda oss av fixed point numbers för att undvika för mycket kvantifiering av signalerna.

2.2.3 Pseudokod för gyroskop

I listing 2 visas preliminär pseudokod för gyroskopets avläsning.

```
  int adc_res

main
  - Initiera nödvändiga portar, ADMUX, referensspänning, samplinghastighet, etc.
  - gyro-sensors analoga output kan MUXas för att välja mellan
  vinkelhastighet och temperatur (där temperaturen kan påverka gyros data med =- 5%)

  while-loop med önskad frekvens:
    - starta AD-omvandling
    - beräkna rotation/riktning utifrån adc_res och förfluten tid
    (integrera)
    - skicka data / gör data tillgänglig för komm.-modulen

Avbrottsrutin för AD-omvandling:
  adc_res <-- "ADCH eller dylikt" (lägg in resultatet av AD-omvandlingen i adc_res)
```

Listing 2: Pseudokod för gyroskop

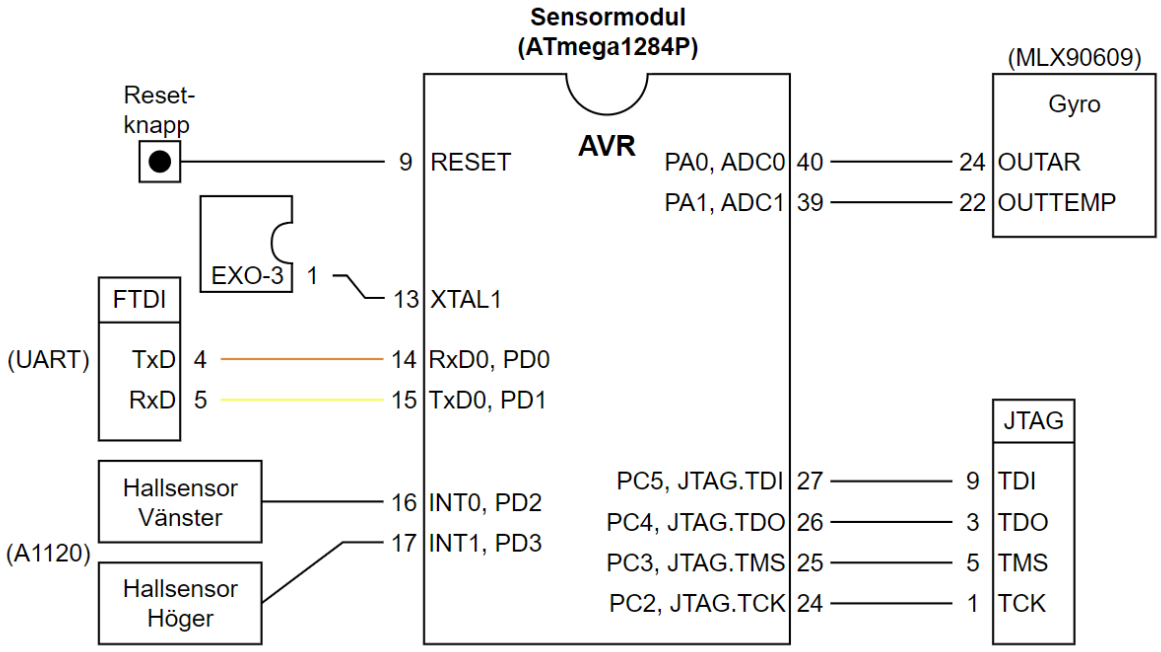
2.3 Prestandabedömning

Minnestillgång och prestanda bör inte vara något problem.

- En 32-bitars integer för räkning av hall-sensor-avbrott.
- En 32-bitars integer för färdsträcka.
- En 8-bitars integer för hastighet.

2.4 Kopplingsschema

I figur 3 presenteras ett kopplingsschema över sensormodulen.



Figur 3: Kopplingsschema för sensormodulen.

3 Styrmodul

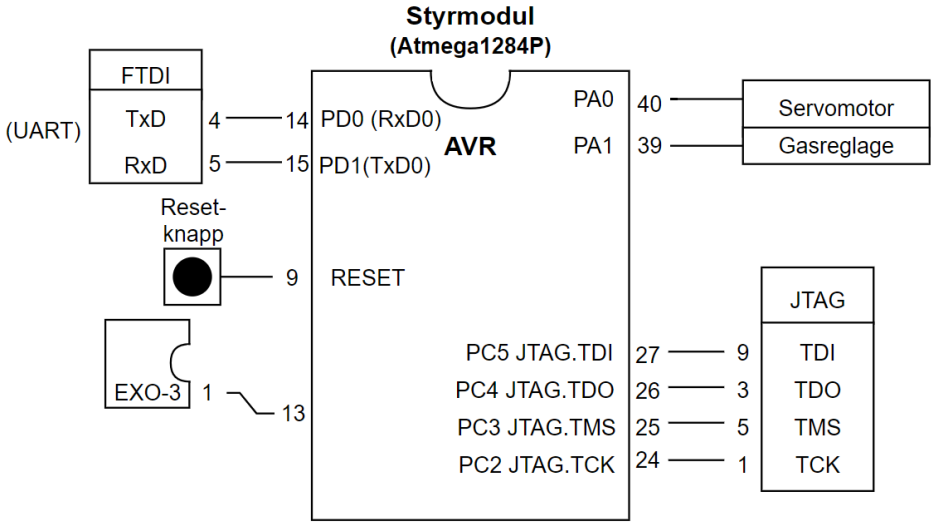
Styrmodulens uppgift kommer vara att reglera bilens styrsystem så att bilen styrs mot ett givet mål. Målet kommer vara en punkt angiven i polär form, med vinkel och avstånd från nuvarande punkt. Genom att signalera till fartreglage och styrservo kommer styrmodulen styra bilen.

Styrmodulen kommer anslutas till kommunikationsmodulen för att ta emot data om hur snabbt, och i vilken riktning hjulen ska snurra för att förflytta bilen i önskad riktning.

Styrmodulen behöver ett särskilt gasreglage som skickar en PWM-signal för att ge motorn information om hur mycket gaspådrag den ska ge.

Genom styrservon kommer styrmodulen kunna reglera hur mycket bilen ska svänga. AVR:en kommer att skicka ut en PWM-signal för att styra en servomotor som i sin tur kommer att vrida på framhjulsaxeln. Se figur 4 för att få en helhetsbild hur styrmodulen ska interagera med relevanta komponenter.

Styrmodulen, likt sensormodulen, får en klocksignal av en extern oscillator (EXO-3).



Figur 4: Kopplingsschema för styrmodulen.

3.1 Komponentbudget

I tabell 2 återfinns information om vilka komponenter och hur många av vardera som behövs för att konstruera styrmodulen.

Tabell 2: Komponenter i styrmodulen.

Komponent	**Antal**
Atmega1284P	1
Servomotor	1
Gasreglage	1
JTAG ICE 3	1

3.2 Mjukvara

Styrmodulen kommer att få information från kommunikationsmodulen om hur mycket den ska svänga och vilket gaspådrag den ska ha. Därefter kommer styrmodulen omvandla informationen till PWM-signaler till styrservon och gasreglaget så att detta uppnås.

Det behövs inga särskilda algoritmer då det mer komplicerade beräkningarna kommer ske i den mer kraftfulla Raspberry Pi:n som ingår i kommunikationsmodulen.

I huvudloopen ska information om svängar och gaspådrag omarbetas till PWM-signaler till gaspådraget och styrservot. Om bilen får informationen att den ska köra rakt fram ska exempelvis servomotorn få en impuls på 1500us som motsvarar servots utgångsläge. Längden på de elektriska pulserna kan uppnås med AVR:ens 8-bit `timer0` och dess PWM.

3.3 Prestandabedömning

Eftersom det inte kommer genomföras några mer komplexa beräkningar på styrmodulens AVR så bör minnestillgång och prestanda inte vara något problem.

4 Kommunikationsmodul

Kommunikationsmodulen kommer ha gränssnitt mot samtliga andra moduler, användargränssnitt och LiDAR. För information om koppling till LiDAR, se avsnitt 5. För koppling till övriga moduler, se avsnitt 7. I kommunikationsmodulen kommer bilens autonoma körning att skötas, och det är vad som primärt beskrivs i det här kapitlet.

4.1 Komponentbudget

Se tabell 3 för de komponenter som behövs.

Tabell 3: Komponenter i kommunikationsmodulen.

Komponent	**Antal**
Raspberry Pi 3B+	1
Strömförsörjning	1

4.2 Mjukvara

Den autonoma styralgoritmen kommer ha två huvudfaser. Den första fasen kommer handla om att kartlägga eller orientera sig på banan och den andra kommer handla om att styra bilen så fort som möjligt.

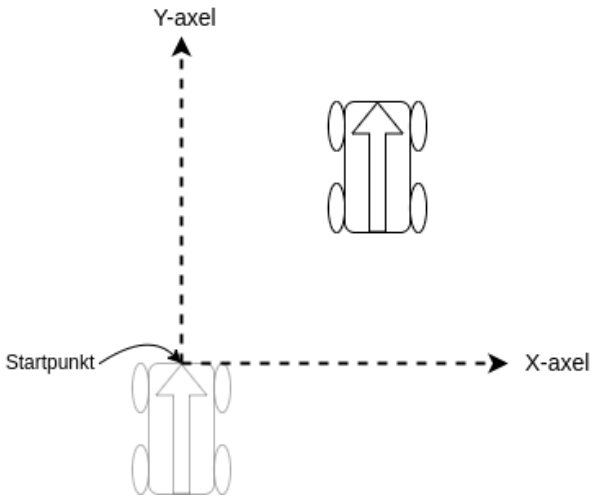
Bilen kommer ha två olika lägen för första fasen som den kan växla mellan kartläggning och orientering.

I kartläggningsläget ska bilen skapa en karta, `ConesMap`, genom att med sin egen position och data från LiDAR bestämma positionen på alla koner i banan. När bilen är i kartläggningsläge kommer den köra långsamt och endast använda sensormodulen samt LiDAR:n för att veta var den är.

I orienteringsläget kommer bilen köra fortare och identifiera sin egen position genom att jämföra kartan den gjorde i kartläggningsläget med verkligheten som den ser med LiDAR. Bilen kommer använda sensormodulen som stöd men förlita sig mest på LiDAR och kartan i orienteringsläget.

4.2.1 Kartläggning

För att kartlägga banan kommer ett koordinatsystem upprättas med origo där bilen startar sin självkörning. Positiva y-axeln kommer vara i riktningen rakt framifrån bilens startposition och positiva x-axeln till höger om bilens startposition (se figur 5). Alla koordinater kan då lagras i datastrukturen `Pos2D`, enligt listing 3, med upplösning på centimeternivå.



Figur 5: Visualisering av koordiinatsystem.

```
struct Pos2D {
  x: i32,
  y: i32,
}
```

Listing 3: Pseudokod för strukturen `Pos2D`

Bilens position och riktning kommer lagras i en datastruktur, `CarPos`, med en `Pos2D` samt grader för rotation. Positiv rotation kommer definieras som rotation motsols. Alltså kommer `CarPos` vara definierad enligt listing 4.

```
struct CarPos {
  pos: Pos2D,
  deg_rotation: i16, // Begränsad mellan 0 och 359.
}
```

Listing 4: Pseudokodsdefinition för strukturen `CarPos`

Koners placering kommer lagras i en datastruktur `ConesMap` som kommer vara en lista av datastrukturer `Cone`. `Cone` kommer i sin tur, enligt listing 5, innehålla en kons `Pos2D` samt en enum som definierar konens storlek.

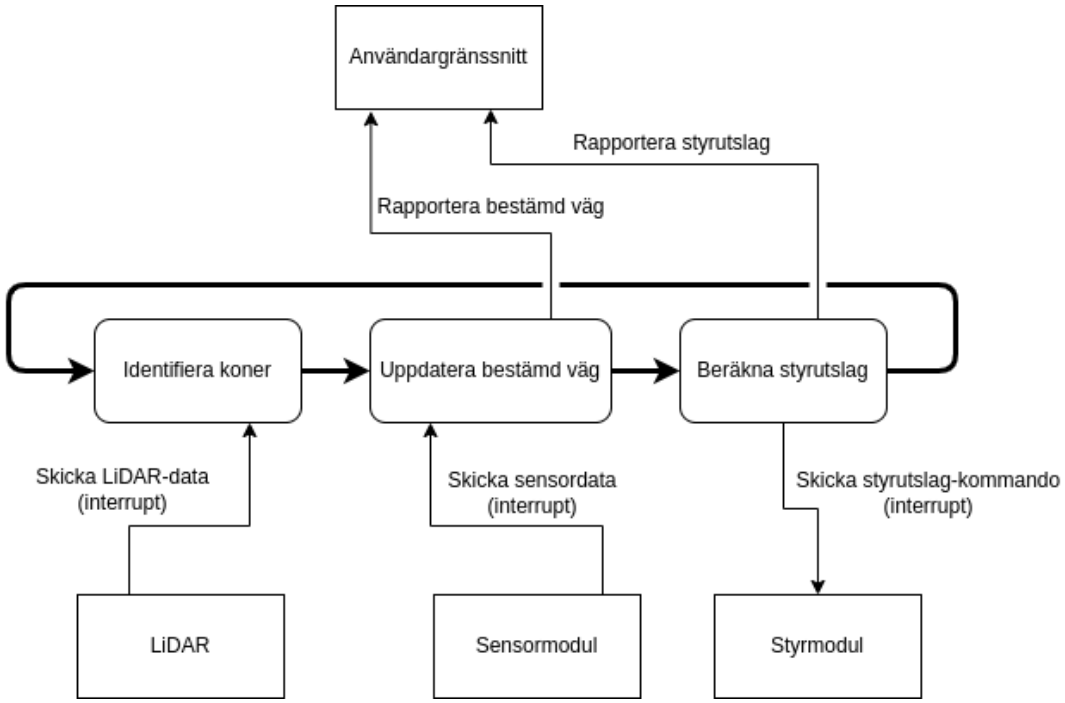
```
enum ConeSize {
  Large, // Stor kon
  Small, // Liten kon
}

struct Cone {
  pos: Pos2D
  size: ConeSize,
}
```

Listing 5: Pseudokodsdefinition för strukturen `Cone`

4.2.2 Styrning

Innan bilen har kartlagt banan kommer den (i kartläggningsläge) åka långsamt mellan konparen genom att sikta mot mitten av öppningen.

Bilen kommer efter att ha kartlagt banan räkna ut en optimal väg att köra med en vägsökningsalgoritm. Den kommer därefter byta till orienteringsläge och öka farten. Bilen kommer att använda en ”pure pursuit”-algoritm för att hålla sig till den optimala vägen mellan konerna. Pure pursuit-algoritmen fungerar genom att ”titta framåt” på en punkt i önskade körvägen och köra mot den.



Figur 6: Blockschema för mjukvara i kommunikationsmodulen.

4.2.3 Main-loop

För en illustration av main-loopen i kommunikationsmodulen, se figur 6.

Identifiera koner Beroende på om bilen är i kartläggningsläge eller orienteringsläge har den här fasen olika syften:

Kartläggning Bilen kommer kartlägga de koner den ser och matcha in dem i `ConesMap`. Om en kon inte fanns där innan lägger algoritmen till den.

Orientering Bilen kommer bestämma sin egen position med hög precision genom att jämföra konerna som den ser med konerna i `ConesMap` och använda sensorerna i sensormodulen.

Uppdatera bestämd väg Med hjälp av sin position mot konerna i `ConesMap` beräknar bilen en rutt med styralgoritmen.

Beräkningar Beräkna styrutslag och skicka kommando till styrmodul.

4.2.4 Datastrukturer

Datan för senaste varvet av LiDAR:n kommer sparas som en lista med punkter av avstånd. (Vinkeln kan kalkyleras från en punkts index i listan.) Varje varv mäter LiDARen 400 gånger. Avstånd består av en 32-bitars unsigned vilket ger $\frac{400 \cdot 32}{8} = 1600$ byte, eller 1,6kB.

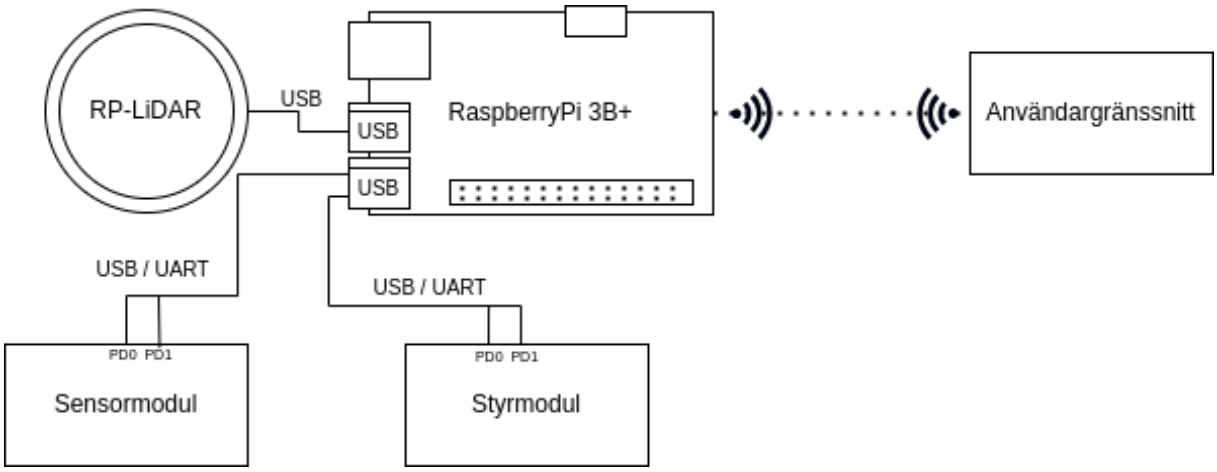
Den kartlagda banan `ConesMap` kommer enbart lagra ett fåtal koner på kommunikationsmodulen. Den kommer därav vara trivialt liten.

4.3 Prestandabedömning

Mängden minne som behövs är försumbar jämfört med den 1GB som en Raspberry Pi 3B+ har. Programmet för bilens autonoma körning borde inte heller vara alltför krävande för Raspberry Pi:ns processor.

4.4 Kopplingsschema

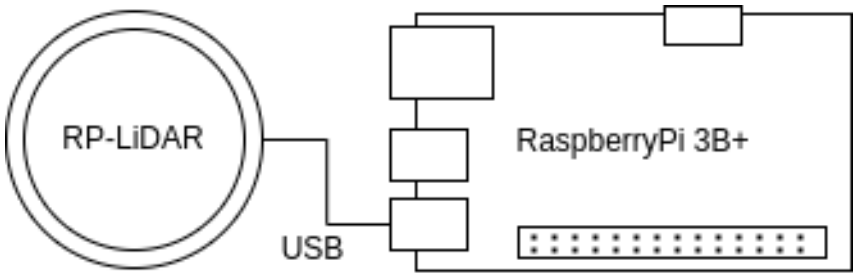
I figur 7 syns ett något avskalat kopplingsschema för alla kommunikationsmodulens gränssnitt. LiDAR, styrmodul och sensormodul är kopplade på Kommunikationsmodulens (Raspberry Pi 3B+) USB-portar. Användargränssnittet är kopplat med Wi-Fi eller Bluetooth till kommunikationsmodulen.



Figur 7: Kopplingsschema för kommunikationsmodulen som består av en Raspberry Pi 3B+.

5 LiDAR

LiDAR:en används för att kartlägga omgivningen och orientera bilen på banan. Kapitel 5 täcker både LiDARen i sig men också delar av kommunikationsmodulen som rör LiDARen. I figur 8 Syns en uppkoppling mellan kommunikationsmodulen och LiDAR. Den har en USB-sladd som kopplas direkt in i en av Raspberry Pi:ens USB-portar.



Figur 8: Ritning för koppling mellan LiDAR och Kommunikationsmodulen.

5.1 Komponentbudget

Se tabell 4 för de komponenter som behövs.

Tabell 4: Komponenter i LiDAR-delsystem.

Komponent	Antal
LiDAR	1

5.2 Mjukvara

Mjukvaran som är relaterad till modulen, alltså Kartläggningsprogrammet, kommer preliminärt att skrivas i Python. Vid behov kan vissa delar skrivas i ett snabbare språk som sedan körs på Raspberry Pi:n.

5.2.1 Programöversikt

Programmet är uppdelat i två olika faser och detta är de delar av faserna som är relevanta för LiDAR:ns funktion:

Uppstartsfas

Initialiseringssekvens Initialiseringen ska utföras enligt protokollet från Slamtec [6] för att starta LiDAR-enheten på ett säkert sätt. Detta inkluderar att skicka korrekta kommandon för att initiera kommunikationen med LiDAR:n via UART-gränssnittet

Säkerställning Systemet verifierar att hårdvaran är korrekt konfigurerad innan exekvering. Detta innebär kontroll av att LiDAR-enheten svarar korrekt på särskilda kommandon, exempelvis `GET_INFO` och `GET_HEALTH`.

Exekveringsfas - del av ”Identifiera koner” kommunikationsmodulens huvudloop

Inläsning av LiDAR-data Systemet registrerar det kontinuerliga flödet av avstånds- och vinkeldata från LiDAR-enheten. LiDAR skickar data i olika format beroende på skanningsläge.

Mottagning av positionsdata Systemet tar emot bilens positionsförändringar (förflyttning och rotation) från kommunikationsmodulen via standardinströmmen.

Kommunikation med processorn Systemet skickar LiDAR-datan till processorn via standardutströmmen.

Datajustering Systemet justerar LiDAR-data baserat på bilens rörelse för att få en korrekt inre representaion av omgivningen. Detta innebär transformation av de polära koordinaterna till ett globalt referenssystem.

Identifiering av koner Systemet använder en algoritm för att kunna identifiera konerna baserat på den justerade LiDAR-datan. Algoritmen baseras på klusteranalys och formigenkänning för att särskilja koner från andra objekt.

Logging Systemet sparar alla mottagna datapunkter och kommunicerar med den externa datorn där allting dumpas till en loggfil. Bufferten rensas efter den dumpat datan.

5.2.2 Algoritmer

Konidentifieringsalgoritm Algoritm baserad på klusteranalys för att momentant kunna avgöra konernas positioner.

1. Filtrering av rådata: Eliminera brus och ogilitiga mätvärden.
2. Klustring: Dela in datapunkterna baserat på dess polära koordinater.
3. Formigenkänning: Identifiera kluster som matchar konens geometriska egenskaper.
4. Beräkning av positioner: Bestäm positionerna och radien för varje kon.

Kartritningsalgoritm En sammanslagning av all momentan data från konidentifieringsalgoritmen efter ett kalibreringsvarv för att rita en karta.

5.2.3 Datastrukturer

`LidarData` Lagrar alla inkommande datapunkter från LiDAR:n under en definierad tidsperiod. Storlek beror på samplingsfrekvensen men kommer att väljas så att det finns plats inom Raspberry Pi:ns tillgängliga minnesrymd. Efter att buffern dumpats till loggfilen rensas den för att frigöra minne.

5.2.4 Avbrottshantering

Inga hårdvaruavbrott används i Kartläggnings-programmet. All mottagning av data sker via en separat process som skriver datan till en buffert. Koden ämnar generellt sett att följa en sekventiell struktur där varje steg i huvudloopen utförs i tur och ordning. Detta minimerar behovet av komplex synkronisering och gör koden mer lättläst och underhållsvänlig.

5.3 Prestandabedömning

Processorn måste kunna hantera behandling och lagring av 4000 datapunkter/sekund från LiDAR:n. Denna kopplas in i Raspberry Pi via kommunikationsmodulen via USB.

5.4 Signaldefinitioner

Se figur 9 för definitioner av alla in- och utsignaler.

Color	Signal Name	Type	Description	Min	Typical	Max
Red	VCC	Power	Total Power	4.9V	5V	5.5V
Yellow	TX	Output	Serial port output of the scanner core	0V	3.3V	3.5V
Green	RX	Input	Serial port input of the scanner core	0V	3.3V	3.5V
Black	GND	Power	GND	0V	0V	0V
Blue	MOTOCTL	Input	Scan motor /PWM Control Signal (active high, internal pull down)	0V	3.3V	5V

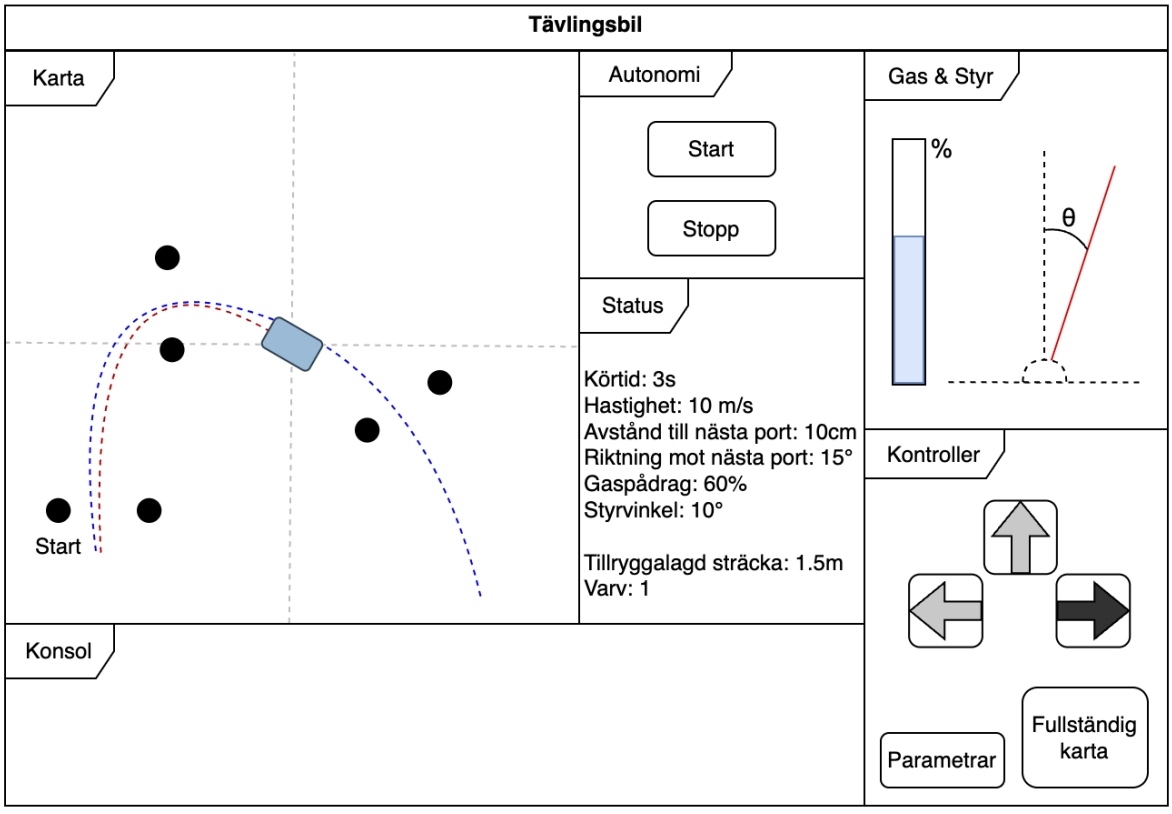
Figur 9: Signaldefinitioner till externt gränssnitt för LiDAR.

6 Användargränssnitt

Användargränssnittet kommer preliminärt vara skrivet i Python. Kommunikation mellan användargränssnitt och bilen kommer ske via WiFi. Användargränssnittet utgör länken mellan bilen och användaren genom att visa relevant data för användaren och låta användaren styra bilen manuellt.

I användargränssnittet ska användaren kunna göra följande (se figur 10 för exempel av användargränssnitt):

- Öka gaspådrag genom att hålla ner tangenten W, svänga till vänster genom att hålla ner A och svänga till höger om användaren håller ner D. Om användaren inte håller ner W ska bilens gaspådrag minskas.
- Gränssnittet ska visa avstånd och riktning till nästa port.
- Gränssnittet visar procentuell andel av gaspådrag relativt max-gaspådrag.
- Gränssnittet visar styrvinkel.
- Gränssnittet visar den totala tiden bilen har kört banan.
- Gränssnittet visar vilket varv den befinner sig på (kartläggningsvarv eller tävlingsvarv).
- Gränssnittet har en knapp som ändrar från autonom till manuell körning (och vice versa).
- Bilens nuvarande hastighet i m/s.
- Gränssnittet ska visa den kartlagda banan där både den tillryggalagda vägen och den planerade vägen ritas ut.



Figur 10: Exempel på hur användargränssnittet skulle kunna se ut.

6.1 Komponentbudget

Se tabell 5 för komponenter relaterade till användargränssnittet

Tabell 5: Komponenter i användargränssnittet.

Komponent	**antal**
Bärbar dator	1

6.2 Mjukvara

Användargränssnittet består av två olika delar. Den ena delen visar information och den andra kan användaren interagera med.

```
Pseudokod till gränssnittet:
def on_press(key):
  match (key):
    case 'w': goFaster()
    case 'a': turnLeft()
    case 'd': turnRight()
    case _: pass

def on_release(key):
  match (key):
    case 'w': goSlow()
    case 'a': stopTurningLeft()
    case 'd': stopTurningRight();
    case _: pass
```

7 Kommunikation mellan delsystemen

All kommunikation mellan moduler på bilen kommer ske via direkt UART. Vardera ATmega1284P kommer ha en direkt seriell anslutning till Raspberry Pi:n. Relevant information kommer skickas kontinuerligt för att motverka effekterna av eventuella störningar. Båda AVR:ena kommer använda `USART0` (`PORTD[1:0]`) för kommunikation. Raspberry Pi:n kommer att använda två adaptrar från USB till UART för att komma undan behovet av att skifta spänningar från 3.3V till 5V och tillbaka.

Raspberry Pi:n kommer även kommunicera med LiDAR:n via USB enligt [6].

Kommunikationen mellan Raspberry Pi:n och den externa datorn sker via en TCP-anslutning över wifi. Data kommer enbart förmedlas när det behövs då TCP är felkorrigerande. Istället för ett eget protokollspråk kommer språkkonstrukt serialiseras, skickas och deserialiseras. Detta kan exempelvis göras med `pickle` för python eller `serde` för rust. Ifall något sådant ej finns för det använda språket kan ett format som json användas.

8 Implementationsstrategi

Då detta är ett omfattande projekt är det nödvändigt att etablera strukturer och hur implementationen ska ske för att förebygga potentiella fallgropar. Detta avsnitt har i avseende att tydliggöra hur projektet är tänkt att bedrivas.

8.1 Dokumentation

Då stor mängd kod kommer att skrivas är det av yttersta vikt att koden är lättläst och välskriven. Därav kommer projektet använda linting och formatteringsverktyg i de använda språken.

8.2 Implementationsordning

Efter att bilen är ihopmonterad är det första som behöver göras att säkerställa att kommunikationen mellan modulerna fungerar. Tanken är att skriva så lite kod som möjligt till en början, men tillräckligt för att kunna se att de modulerna kan kommunicera med varandra. Eftersom all kommunikation går via kommunikationsmodulen är det nödvändigt att alla andra moduler testas med den.

När kommunikationen är etablerad ska den riktiga implementationen av modulerna påbörjas, alltså att få in rätt data och se till att datan från modulerna förmedlas felfritt. I detta skede är nyckeln till framgång att testa så mycket som möjligt (av det som går att testa.) Detta kommer att göras genom att på ett primitivt sätt säga till bilen ”gasa” i 5 sekunder, sväng vänster i 2 sekunder, och så vidare.

Nästa steg blir att fortsätta utveckla den manuella körningen av bilen och då behöver åtminstone sensormodulen, styrmodulen, kommunikationsmodulen och användargränssnittet vara delvis implementerade. När manuell körning uppnås handlar det till stor grad om testning av de olika sensorerna för att se att den tänka implementationen faktiskt fungerar som det är tänkt.

Till sist (men detta är ett spår som har arbetats på löpande) ska den autonoma körningen implementeras med hjälp av LiDAR. LiDAR:n kan testas separat för att först säkerställa att tolkningen av data fungerar och sedan för att tolka den datan för att kunna visualisera banan.

8.3 Test

Test av modulerna och delsystem sker löpande under utvecklingsprocessen. Till hjälp för testning av signaler mellan komponenter kommer signalanalysatorn Diligent Discovery användas [7]. Följande är tester som kommer utföras på systemet i helhet:

1. Skicka meddelanden mellan de tre modulerna på bilen.
2. Bilen kan utföra enkla körkommandon från externa datorn.
3. Köra bilen manuellt från användargränssnittet.
4. Bilen kan kartlägga omgivning och visa i användargränssnittet.
5. Bilen kan köra kalibreringsvarv.
6. Bilen kan efter kalibreringsvarv, köra tävlingsvarv.

8.4 Feedback

Feedback av systemet kommer att ske primärt via användargränssnittet. För de mindre modulerna (styrmodulen och sensormodulen) kan exempelvis signalanalysatorn Diligent Discovery kopplas för att ge en visuell feedback när någon signal är hög.

Bibliografi

[1] A. Corporation, ”8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash”. 09 november 2009.

[2] Melexis, ”MLX90609, Angular Rate Sensor (Standard version)”. februari 2008.

[3] L. Allegro MicroSystems, ”Chopper-Stabilized Precision Hall-Effect Switches”. 04 november 2016.

[4] I. för Systemteknik, ”JTAG”. Åtkomstdatum: 04 oktober 2024. [Online]. Tillgänglig vid: https://da-proj.gitlab-pages.liu.se/vanheden/page/handledningar/avr_jtag/

[5] L. Future Technology Devices International, ”TTL to USB Serial Converter Range of Cables Datasheet”. 23 maj 2016.

[6] L. Shanghai Slamtec.Co., ”RPLIDAR, 360 Degree Laser Range Scanner, Interface Protocol and Application Notes”. 28 mars 2019.

[7] ”Digital Discovery - Getting Started Guide”. Åtkomstdatum: 10 oktober 2024. [Online]. Tillgänglig vid: https://digilent.com/reference/test-and-measurement/digital-discovery/getting-started-guide?redirect=1