Teknisk Dokumentation
Kartrobot

TSEA29

Grupp 8

Publicerad: 2023-12-28

Version: 1.1

Innehåll

1 Inledning

Denna tekniska dokumentation utforskar och presenterar en Autonom Kartläggningsrobot med avancerad Simultaneous Localization and Mapping (SLAM), Frontier och A*-algoritmen. Dokumentationen syftar till att erbjuda en djupgående förståelse för robotens design, dess funktioner och tekniska komponenter. Projektet kombinerar precisionen hos Lidar-sensorer med kraften hos SLAM, Frontier och A* algoritmer för att skapa en autonom robotkapacitet som inte bara navigerar i sin omgivning utan också aktivt kartlägger och lagrar miljöinformation i realtid.

1.1 Bakgrund

Robottekniken fortsätter att driva gränserna för vad som är möjligt, och vår Autonoma Kartläggningsrobot representerar en banbrytande lösning. Genom att integrera avancerade algoritmer som SLAM, Frontier Detection och A* har gruppen strävat efter att skapa en intelligent robot som kan göra informerade beslut för att effektivt utforska och kartlägga okända miljöer.

1.2 Syfte

Robotens mål sträcker sig bortom enbart autonom navigering och inkluderar aktiv kartläggning av omgivningen. Genom att använda SLAM, Frontier och A*-algoritmen strävar gruppen efter att skapa en lösning som är kapabel att inte bara förstå sin position utan också att intelligent utforska och kartlägga den omgivande terrängen.

2 Sensormodul

2.1 Översikt

Detta kapitel kommer att behandla sensormodulens funktionalitet samt struktur. Sensormodulen körs delvis på en Atmega 1284p och en RaspberryPi 3.

2.2 Modulens funktion

Modulens funktion är att genera sensor data, behandla sensordata samt att skicka vidare denna data till de funktioner som är beroende av denna data.

2.3 Komponeneter

- Atmega1284p [1]
- RaspberryPi 3 [2]
- 2x Odometer [3]
- RPLidar A2 [4]
- 3x IR sensorer [5] [6]
- Kopplinstråd

2.3.1 Hårdvarubeskrivning

Atmega1284p

I sensormodulen används en atmega1284p för att hantera odometrarna samt IR sensorerna. Denna används då den har många av de önskade funktionerna; inbyggd timer, avbrottshantering samt analog-to-digital konvertering (ADC)

- För odometrarna används den inbyggda 16bit timer/counter, avbrottshantering samt ADC.
- För IR sensorerna används avrbrottshantering samt ADC.

Raspberry Pi 3

Det används också en Rapsberry Pi 3 till sensormodulen. Denna används för at t koppla RPLidar A2. RPlidar A2 kopplas med en XH2.54-5P - USB adapter till Raspberry Pi:n.

- Raspberry Pi:n används för att snabbt kunna behandla den data som RPLidar A2 samlar in.

Odometer

Odometrarna som används till sensormodulen är så kallade reflexsensorer. Dessa består utav en lysdiod med infrarött ljus som och en mottagardiod som känner av det reflekterade ljuset.

Reflexsensorn är analog och skickar konstant ut ljuspulser. Då underlaget sensorn lyser på reflekterar mycket ljus blir utsignalen från reflexsensorn låg. Absorberar underlaget mycket ljus blir utsignalen hög.

Dessa värden är analoga, och uppmätt spann är mellan 1.2 - 3.5 Volt.

RPLidar A2

Sensorn RPLidar främsta syfte är att kartlägga. Detta då den har ett omfång på 360 grader och kan upptäcka objekt runtomkring den med bra kvalitet upp till 6m. Den har hög skanningshastighet på 10Hz, eller 600RPM.

Den kopplas smidigt upp till Raspberry Pi:n med den USB adapter tidigare nämnd i 2.3.2 och det finns färdigbyggda paket som kan hantera dess kommandon.

IR sensor

De tre IR sensorerna som används är; 2st GP2D120 och en GP2Y0A21.

GP2D120 kan mäta avstånd mellan 4-30cm. GP2Y0A21 Kan mäta avstånd mellan 10-80cm.

Bägge modellerna är analoga och behöver analog inspänning och ger en analog utsignal. Utsignalen varierar med avståndet från det den mäter.

Dessa är kopplade till atmega1284p och använder sig utav ADC samt interruptus.

Dessa har dock prioriterats bort då RPLidar A2 överglänste i precision.

2.4 LIDAR

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

3 Styrmodul

3.1 Översikt

Styrmodulen är det delsystem som ser till att systemet förflyttas. Genom kommandon från huvudprocessorn regleras fyra växlade DC-motorer (7.2V, 291 RPM). Motorerna styrs parvis med två signaler per sida. DIR, som styr motorernas rotationsriktning och PWM [1], pulsbreddsmodulering som styr motorernas hastighet.

Motorparen styrs kan styras oberoende av varandra och tillsammans får de roboten att röra på sig.

Vid autonom styrning är det indata från sensormodulen som behandlas och skickas till huvudprocessorn varefter kommandon skickas till styrmodulen för styrning av enheten via en PID algoritm för att säkerhetsställa att systemet förflyttar sig enligt specifikation. Exempel på sensorer som kommer användas för att styra enheten är odometer för att räkna ut förflyttning samt Lidar för att räkna ut avstånd. Dessa sensorer hör dock till sensormodulen och kommunicerar inte direkt med styrmodulen.

Vid fjärrstyrning skickas kommandon från datorn till huvudprocessorn vidare till styrenheten för att reglera motorerna.

3.2 Manuell styrning

3.2.1 Översikt

Styrmodulen är utformad för att tillhandahålla funktioner för styrning av en enhets rörelse med hjälp av pulsbreddsmodulering (PWM) för hastighetsstyrning och riktningsstyrning. Modulen innehåller följande filer:

- navigation.h: Headerfil som innehåller funktionsdeklarationer och nödvändiga makron.
- navigation.c: fil som implementerar de funktioner som deklareras i navigation.h.
- setDirection.c: fil som implementerar funktionen för att ställa in rörelseriktningen.
- pwm_dir.c: fil som implementerar PWM-initialisering och hastighetskontrollfunktioner.

Användning

För att använda navigeringsbiblioteket måste headerfilen ”navigation.h” inkluderas.

Globala Variabler

- `wheelpair_speed_right` och `wheelpair_speed_left`: Variabler för att bestämma hastigheten för höger och vänster hjulpar.
- `right_speed` och `left_speed`: Variabler för att spåra hastigheten för höger och vänster hjul separat.

Macros

F_CPU:

Beskrivning: Definierar CPU-frekvensen (8MHz som standard).

3.2.2 Funktionsbeskrivning

`move_forward(uint8_t DIRECTION)`

Funktion: Rör sig framåt genom att sätta hjulhastigheter och riktning för att köra framåt.

Parametrar:

Retur: dfdsf

move_backward(uint8_t DIRECTION)

Funktion: Rör sig bakåt genom att sätta hjulhastigheter och riktning för att köra bakåt.

Parametrar:

Retur:

move_right(uint8_t DIRECTION)

Funktion: Rör sig åt höger genom att sätta hjulhastigheter och riktning för att svänga åt höger.

Parametrar:

Retur:

move_left(uint8_t DIRECTION)

Funktion: Rör sig åt vänster genom att sätta hjulhastigheter och riktning för att svänga åt vänster.

Parametrar:

Retur:

move_slight_r(uint8_t DIRECTION)

Funktion: Gör en lätt högersväng genom att justera höger hjulhastighet jämfört med vänster hjulhastighet.

Parametrar:

Retur:

move_slight_l(uint8_t DIRECTION)

Funktion: Gör en lätt vänstersväng genom att justera vänster hjulhastighet jämfört med höger hjulhastighet.

Parametrar: dsd

Retur: lsad

set_right_speed(uint8_t duty_threshold)

Funktion: Sätter hastigheten för det högra hjulet genom att justera duty cycle.

Parametrar: Inga.

Retur: Ingen.

set_left_speed(uint8_t duty_threshold)

Funktion: Sätter hastigheten för det vänstra hjulet genom att justera duty cycle.

Parametrar: Inga.

Retur: Ingen.

InitPWM(void)

Funktion: Denna funktion initierar konfigurationen för pulsbreddsmodulering (PWM). Den ställer in nödvändiga register och konfigurationer för att möjliggöra PWM-utgång.

Parametrar: Inga.

Retur: Ingen.

setSpeed(int leftSpeed, int rightSpeed)

Funktion: Ställer in hastigheten på motorerna oberoende av varandra. Tar två heltalsargument, leftSpeed och rightSpeed, som representerar hastigheten för vänster och höger motor, respektive.

Parametrar: int leftSpeed (0-255), int rightSpeed (0-255)

Retur: Ingen.

setDirection(int direction)

Funktion: Ställer in riktningen för roboten baserat på det angivna direction-parametern.

Parametrar: direction (1 för frammåt, 0 för bakåt)

Retur: Ingen.

move_man(int direction)

Funktion: Förflyttar roboten manuellt baserat på det angivna direction-parametern.

Parametrar: direction (1 för frammåt, 0 för bakåt)

Retur: Ingen.

Koden för den manuella styrningen är sammanslaget med koden för den autonoma delen. Läsare hänvisar därför till RunAuto.py.

3.3 RunAuto.py

Pythonkoden är en del av programmet som styr roboten för autonom utforskning.

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

4 Kommunikationsmodul

4.1 Översikt

Kommunikationsmodulen gör det möjligt att överföra data mellan alla moduler som ingår i konstruktionen. Kommunikation mellan systemets enheter sker med hjälp av en intern I2C-buss samt Wifi-uppkoppling för överföring av data från och till en dator.

4.2 I2C

I2C (*Inter-Integrated Circuit*) är ett seriellt kommunikationsprotokoll som används för att ansluta och möjliggöra kommunikation mellan olika integrerade kretsar (IC), sensorer och andra elektroniska enheter.

4.2.1 Översikt

Roboten har en intern I2C-buss som hanterar överföringen mellan styrmodulen och Raspberry Pi samt mellan sensormodulen och Raspberry Pi. Där Raspberry Pi är master, styr och sensormodulen är slave. I2C är uppbygt av nedanstående delar.

Python-skriptet underlättar I2C-kommunikation på Raspberry Pi som agerar som en I2C-master. Det kommunicerar med två I2C-enheter: en I2C-sensor med adress 0x35 och en I2C-styrning med adress 0x42. Skriptet använder smbus2-biblioteket för att interagera med I2C-bussen. I2C protokollet beskrivs i tabell 1.

4.2.2 I2C beskrivning

Del	Beskrivning
Start-Stop	Kommunikationen i I2C inleds av en startsignal(S) från mastern och avslutas med en stop signal(P). Startsignalen indikerar att en ny kommunikationssession börjar, medan stoppsignalen indikerar att sessionen är över.
Addressering	Varje I2C enhet (slave-enhet) har en unik adress som används av mastern för att identifiera vilken som ska kommunicera med. Den vanligaste adressen är 7 bitar som kan adressera 128 slave enheter.
Dataöverföring	Data överförs i I2C som 8 bitars datapaket. Kommunikationen är synkroniserad, och båda enheterna måste vara överens om överföringshastigheten.
Läs- och skrivåtergärder	I2C-överföringar kan vara läsningar (från slav till master) eller skrivningar (från master till slave). En startsignal följs av adress och en bit som indikerar om mastern vill läsa eller skriva data. Denna bit kallas "Read/Write"-bit.
ACK och NACK	Efter varje 8-bitars överföring svarar mottagaren (slave-enheten) med en ACK eller NACK-signal. En ACK signal (logiskt låg) anger att överföringen lyckades, medan en NACK signal (logiskt hög) indikerar ett fel eller att överföringen avslutades.
Byte sändning och mottagning	2C-överföringar sker byte för byte. Efter varje byte måste mottagaren skicka ett ACK eller NACK beroende på om den vill fortsätta överföringen eller inte.
Klocka och datapinnar	I2C använder två signalpinnar: en klocka (SCL - Serial clock) och en data (SDA - Serial data). Klockpinnen, kallad SCL, används för att synkronisera timingen för dataöverföringen mellan master och slave-enheter. Klocksignalen bestämmer när data ska avläsas eller sändas. Kommunikationen sker i pulser, och data byts normalt ut på varje stigande flank (övergång från låg till hög) eller fallande flank (övergång från hög till låg) av klocksignalen. Datapinnen kallad SDA, är den faktiska ledaren för datan i I2C-kommunikationen. Master och slave-enheter använder denna pinne för att sända och ta emot data. SDA är en tvåvägsbuss, vilket innebär att både master och slave-enheter kan sända och ta emot data på samma ledare.

Tabell 1: Beskrivning av I2C-protokollet

4.2.2.1 Master

Raspberry Pi förbestämd som master. Sensorenheten och styrenheten är förbestämda som slave.

Python-skriptet underlättar I2C-kommunikation på Raspberry Pi som agerar som en I2C-master. Det kommunicerar med två I2C-enheter: en I2C-sensor med adress 0x35 och en I2C-styrning med adress 0x42. Skriptet använder smbus2-biblioteket för att interagera med I2C-bussen.

Beroenden

smbus2 Bibliotek:

Skriptet är beroende av smbus2-biblioteket för I2C-kommunikation.

4.2.2.2 Kodförklaring

I2C Enhetsadresser:

· i2c_styr_address = 0x42: I2C-adress för styrenheten (0x42).

· i2c_sensor_address = 0x35: I2C-adress för sensorenheten (0x35).

4.2.2.3 Läsning från Sensor (read_from_sensor-funktion):

· Öppnar en I2C-buss (SMBus(1)).

· Initierar en läs-operation från sensorenheten (i2c_sensor_address).

· Skriver ut den mottagna data och returnerar den som en lista.

· Hanterar undantag och returnerar en standardlista [0, 0, 0, 0] vid fel.

4.2.2.4 Skrivning till Styrning (write_to_styr-funktion):

· Öppnar en I2C-buss (SMBus(1)).

· Initierar en skrivoperation till styrenheten (i2c_styr_address) med den angivna data.

· Hanterar undantag och skriver ut ett felmeddelande om skrivoperationen misslyckas.

4.2.2.5 Användning:

· Anropa read_from_sensor-funktionen för att läsa data från sensoren.

· Anropa write_to_styr-funktionen med data för att skicka till styrenheten.

4.2.3 Styr (slave)

4.2.3.1 Översikt

Python-skriptet underlättar I2C-kommunikation på Raspberry Pi som agerar som en I2C-master. Det kommunicerar med två I2C-enheter: en I2C-sensor med adress 0x35 och en I2C-styrning med adress 0x42. Skriptet använder smbus2-biblioteket för att interagera med I2C-bussen.

Beroenden

Skriptet är beroende av smbus2-biblioteket för I2C-kommunikation.

4.2.3.2 Kodförklaring

I2C Enhetsadresser:

- i2c_styr_address = 0x42: I2C-adress för styrenheten (0x42).
- i2c_sensor_address = 0x35: I2C-adress för sensorenheten (0x35).

4.2.3.2.1 Läsning från Sensor (read_from_sensor-funktion):

- Öppnar en I2C-buss (SMBus(1)).
- Initierar en läsoperation från sensorenheten (i2c_sensor_address).
- Skriver ut den mottagna datan och returnerar den som en lista.
- Hanterar undantag och returnerar en standardlista [0, 0, 0, 0] vid fel.

4.2.3.2.2 Skrivning till Styrning (write_to_styr-funktion):

- Öppnar en I2C-buss (SMBus(1)).
- Initierar en skrivoperation till styrenheten (i2c_styr_address) med den angivna datan.
- Hanterar undantag och skriver ut ett felmeddelande om skrivoperationen misslyckas.

4.2.3.2.3 Användning:

- Anropa read_from_sensor-funktionen för att läsa data från sensoren.
- Anropa write_to_styr-funktionen med data för att skicka till styrenheten.

main.c

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

4.2.4 Sensor (slave)

4.2.4.1 Översikt

Koden implementerar I2C-protokollet för seriell kommunikation mellan enheter på en gemensam buss, med fokus på sensor-slavenhetens funktioner. Den möjliggör pålitligt datautbyte och har strukturerats modulärt för enkel integrering i olika slavenheter. Här presenteras de viktigaste funktionerna och globala variablerna.

4.2.4.2 Globala Variabler

Globala variabler är definerade i header filen

sensor_data_buffer

- Typ: Array av uint8_t
- Beskrivning: Buffer för att lagra sensordata som ska skickas från sensor-slaven till huvudenheten.

received_data_buffer

- Typ: Array av uint8_t
- Beskrivning: Buffer för att lagra mottagen data från huvudenheten till sensor-slaven.

4.2.4.2.1 sensor_index

- Typ: Heltal
- Beskrivning: Indexvariabel för att hålla koll på positionen i sensor_data_buffer vid sändning till huvudenheten.

4.2.4.2.2 received_index

- Typ: Heltal
- Beskrivning: Indexvariabel för att hålla koll på positionen i received_data_buffer vid mottagning från huvudenheten.

4.2.4.3 Funktioner

init_Sensor_Slave()

Denna funktion initialiserar I2C-sensorslaven och är avgörande för att etablera kommunikation med andra enheter på I2C-bussen.

- **TWCR Setup:** Konfigurerar kontrollregistret TWCR för att aktivera TWI, möjliggöra *acknowledgment* och initiera ett TWI-avbrott.
- **TWAR Setup:** Sätter adressregistret TWAR med en unik I2C-adress för sensorslaven.
- **Globala Avbrott (Interrupts):** Aktiverar globala avbrott med **sei()** för att möjliggöra hantering av avbrott när TWI-gränssnittet signalerar händelser.

receive_Data()

Denna funktion används för att ta emot data från en master-enhet.

- **Lagra Mottagen Data:** Lagrar mottagna datat från TWDR i *received_data_buffer* baserat på aktuellt index.
- **Indexhantering:** Inkrementerar index för nästa mottagna byte och återställer index om bufferten är full.
- **Kontrollera ACK/NACK:** Beroende på om det är sista byte i överföringen eller inte, skickas ACK eller NACK som svar.

ISR(TWI_vect)

Interrupt Service Routine (ISR) är kärnan i I2C-sensorslavkommunikationen. Nedan beskrivs en Interrupt Service Routine (ISR) för Two-Wire Interface (TWI) på en AVR-mikrokontroller. ISR är utformad för att hantera händelser relaterade till slavlägesoperation. ISR stödjer olika TWI-statusflaggor och utför specifika åtgärder baserat på mottagna TWI-statusar.

4.2.4.3 Funktionalitet

TWI Slave ISR är strukturerad med en switch-case-sats som utvärderar TWI-statusflaggor och utför motsvarande åtgärder. Huvudfunktionerna inkluderar hantering av SLA (Slave Address) godkännande, dataöverföring, datamottagning och hantering av olika statusvillkor.

Kodgenomgång

- Statusflagg TW_ST_SLA_ACK utnyttjas för att signalera att en slavadress har godkänts av TWI-slaven. Rutinen kontrollerar om det nuvarande bytet är det sista i bufferten (BUFFER_SIZE - 1).
    - Om ja, förbereder sig för att skicka det sista databytet och förbereder att skicka NACK efteråt.
    - Om nej, skickar det aktuella databytet och förbereder att skicka ACK för att indikera att fler databyten är att vänta.
- Statusflagg TW_ST_ARB_LOST_SLA_ACK utnyttjas för att signalera att arbitrering har förlorats under överföringen av SLA. Rutinen kontrollerar om det nuvarande bytet är det sista i bufferten (BUFFER_SIZE - 1).
    - Om ja, förbereder sig för att skicka det sista databytet och förbereder att skicka NACK efteråt.
    - Om nej, skickar det aktuella databytet och förbereder att skicka ACK för att indikera att fler databyten är att vänta.
- Statusflagg TW_ST_DATA_ACK utnyttjas för att signalera att dataöverföring sker och ACK har mottagits från master-enheten. Rutinen kontrollerar om det nuvarande bytet är det sista i bufferten (BUFFER_SIZE - 1).
    - Om ja, förbereder sig för att skicka det sista databytet och förbereder att skicka NACK efteråt.

    - Om nej, skickar det aktuella databytet och förbereder att skicka ACK för att indikera att fler databyten är att vänta.

- Statusflagg TW_ST_DATA_NACK utnyttjas för att signalera att det sista databytet har överförts och NACK har mottagits från master-enheten. Släpper TWI-periferin för att vara redo för ett nytt startvillkor.
- Statusflagg TW_ST_LAST_DATA utnyttjas för att signalera att det sista databytet har överförts och ACK har mottagits från master-enheten. Släpper TWI-enheten för att vara redo för ett nytt startvillkor
- Statusflagg TW_SR_SLA_ACK används för att signalera att en slavadress har godkänts av TWI-slaven och att den är redo att ta emot data. Sätter TWI-status till mottagningsläge med acknowledgment (ACK).
- Statusflagg TW_SR_ARB_LOST_SLA_ACK används för att signalera att arbitrering har förlorats under mottagningen av SLA, men SLA har ändå mottagits och ACK har skickats tillbaka. Släpper TWI-periferin för att vara redo för ett nytt startvillkor.
- Statusflagg TW_SR_GCALL_ACK används för att signalera att en generell anrops-SLA har godkänts av TWI-slaven, och den är redo att ta emot data. Sätter TWI-status till mottagningsläge med acknowledgment (ACK).
- Statusflagg TW_SR_ARB_LOST_GCALL_ACK används för att signalera att arbitrering har förlorats under mottagningen av en generell anrops-SLA, men SLA har ändå mottagits och ACK har skickats tillbaka. Släpper TWI-periferin för att vara redo för ett nytt startvillkor.
- Statusflagg TW_SR_DATA_ACK används för att signalera att data har mottagits, och ACK har skickats tillbaka till master-enheten. Läser mottaget data och skickar ACK för att indikera att slaven är redo att ta emot mer data.
- Statusflagg TW_SR_DATA_NACK används för att signalera att det sista databytet har mottagits, och NACK har skickats tillbaka till master-enheten. Mottar det sista databytet och förbereder TWI-periferin för ett nytt startvillkor.
- Statusflagg TW_SR_GCALL_DATA_ACK används för att signalera att data har mottagits efter att en generell anrops-SLA har skickats, och ACK har skickats tillbaka till master-enheten. Läser mottaget data och skickar ACK för att indikera att slaven är redo att ta emot mer data.
- Statusflagg TW_SR_GCALL_DATA_NACK används för att signalera att data har mottagits efter att en generell anrops-SLA har skickats, och NACK har skickats tillbaka till master-enheten. Läser mottaget data och förbereder TWI-periferin för ett nytt startvillkor.
- Statusflagg TW_NO_INFO Indikerar att TWI-slaven inte svarade.
- Statusflagg TW_BUS_ERROR indikerar att TWI-busfel har detekterats. Det kan vara en situation där det har uppstått ett fel på TWI-bussen under kommunikationen.
- TWI Slave Receiver Stop (TW_SR_STOP)

  Sätter TWI-status för att vara redo för ett nytt startvillkor efter ett stoppvillkor.
- Default:

Slutsats

TWI Slave ISR är utformad för att hantera olika TWI-statusflaggor och utföra specifika åtgärder därefter. Den stöder SLA-godkännande, dataöverföring, datareception och hanterar olika statusvillkor. ISR är en viktig komponent för att möjliggöra kommunikation mellan RPI och TWI-enheter.

4.2.4.4 Användning

För att använda I2C-sensorslaven kan **init_Sensor_Slave()** anropas från huvudprogrammet. Funktionen kontrollerar kontinuerligt om det inte pågår några I2C-överföringar och väntar på att nya kommandon ska tas emot.

main.c

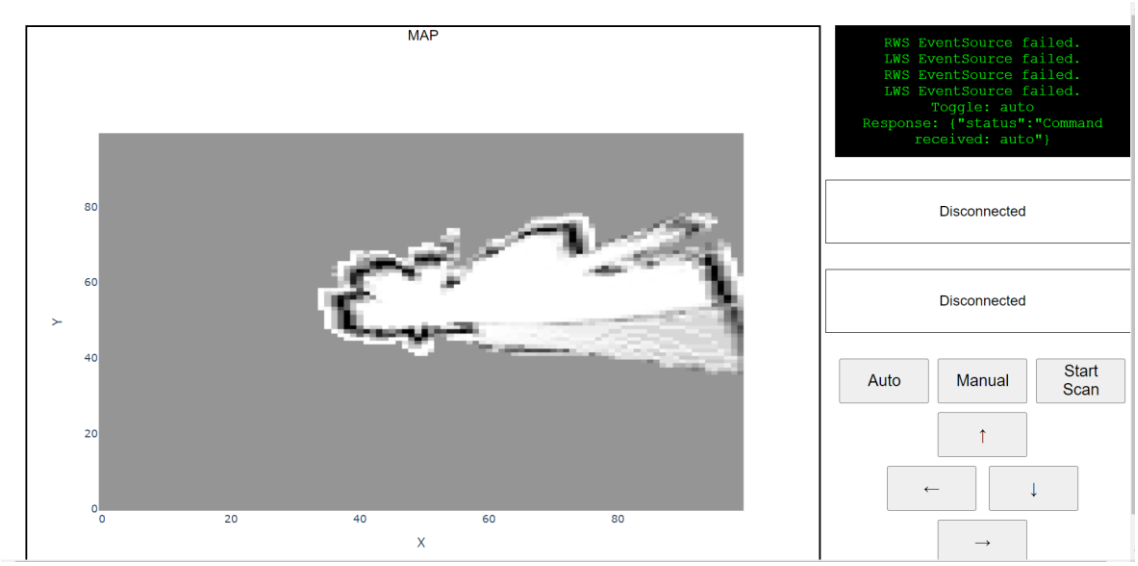
För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

5 Gränssnitt

5.1 Översikt

Detta kapitel beskriver gränssnittets funktionalitet samt struktur. Gränssnittet körs på en webapplikation.

På hemsidan finns det kontroller för att styra roboten manuellt, knappar för att byta manuellt läge samt automatiskt, en knapp för att manuellt göra en scanning med lidar, informationsrutor för sensor data och en visualisering över det rum roboten skannar av.



Figur 1: Gränsnitt

5.2 Instruktioner

1. För att köra venv kör "source venv/bin/activate" i mappen "W.I.P-JATAKA/1_gränssnitt".
2. Nu när hemsidan körs på pin. Först koppla upp pin och en dator på samma nätverk, ex dela från telefon.
3. Sen kör hemsidan, inte i vertual enviromenten (venv) "uvicorn main:app --host 0.0.0.0 --port 8000"
4. Adressen användaren ska ansluta till är http://192.168.72.3:8000/ om inte, kör ifconfig i pins terminal, IP-addressen användaren söker är:
    - wlan0: inet xxx.xxx.xx.x sen porten 8000

Just nu är det fortfarande fel på vissa funktioner som visar sensor data. FastAPI ger möjligheten att köra en websida på RPI:n, denna sida kommer användaren kunna nå genom mobilen, laptop etc.

5.3 Översikt

För att göra gränssnittet tillgängligt över många olika enheter används python paketet *FastAPI* [1]. FastAPI är ett webbapplikations paket för att bygga APIs. Med detta paket kan raspberry Pi:n hosta en server på ett lokalt nätverk, där hemsidan är tillgänglig för alla uppkopplade på samma nätverk. Webbapplikationen hanterar http requests mellan front-end och back-end.

FastAPI hanterar uppstarten och nedstängningen av robotens funktionalitet, när servern startar initialiseras 4 olika parallella processer; Lidar-processen, SLAM-processen, Autostyrningsprocessen, samt den process som kör hemsidan.

För kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

5.4 Frontend

5.4.1 Översikt

Denna del går igenom front-end delen av gränssnittet, där till html, css och javascript. Front-end delen har I Ansvar att rendera kontroll gränssnittet, visa visuella data samt skicka kommandon till back-end

5.4.2 HTML

HTML strukturen ger layouten till hela gränssnittet. Detta inkluderar en karta utav SLAM-visualiseringen, en kommandologgning för kommandohistorik och kontroll knappar för roboten.

- Karta (div#map-container)
    - Visar SLAM-visualiseringen av kartan
    - Positionerad som ett av huvudelementen på hemsidan
- Kommandologger (div#log-container)
    - Visar en historia av de kommandon som gränssnittet kommunicerar
    - Implementerat med en scroll funktion för att minimera plats
- Kommandoknappar (div#.command-container)
    - Två knappar, *Manual* och *Auto* för att byta mellan manuell och automatisk-styrning
    - *Scan* knappen aktiverar Lidar, främst använts för felsökning
    - Använder sig av *Event listeners* för att utföra motsvarande Javascript funktion
- Riktningsknappar (div#.control-container)
    - Fyra knappar för att manuellt styra roboten. Framåt, bakåt, vänster och höger.
        - Kan användas i kombination för att till exempel köra fram-vänster, fram-höger etc.
    - Använder sig av *Event listeners* för att hantera knapptryck eller mustryck.

5.4.2.1 index.html

HTML-koden definierar strukturen på en webbsida för ett gränssnitt för robotstyrning.

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

5.4.3 CSS

Anpassar det HTML koden presenterar. Säkerhetsställer att hemsidan anpassar sig efter olika plattformar. Sätter stilen för hur knappar, loggning och inforutor ser ut visuellt.

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

5.4.4 JavaScript

JavaScript koden hanterar interaktiva element på hemsidan så som knapptryck, realtids uppdatering utav element som HTML koden presenterar.

Hanterar kommunikationen mellan back-end och front-end, detta genom API slutpunkter. Implementation av *Server-sent Events (SSE)* för att skicka och ta emot data från och till back-end samt uppdaterar SLAM-visualiseringen som karta.

- Globala variabler
    - Används för att hålla koll på olika stadier och kommandon.
- Event hanterare (Event Handlers)
    - Olika funktioner för att hantera knapptryck eller piltangenter.
        - Ex. När framåt knappen hålls nere skickas endast ett kommando att köra framåt, när knappen släpps skickas ett kommando att stanna.
    - Garanterar ett intuitivt användande.
- Riktnings översättning och Robot styrning
    - Funktioner för att översätta de olika användare interaktionerna till robotkommandon.
        - Ex. Om användaren klickar framåt och vänster, översätts dessa kommandon så att roboten får rätt riktningskommandon. Släpper användaren vänster knapp, men håller kvar framåt skickas inte ett stoppkommando, utan ett framåt kommando.
    - Funktioner för att skicka styrkommandon till back-end
- *Server-Sent Events* (SSE)
- Hanterar en kontinuerlig ström av data från back-end.
    - Ex. SLAM-visualiseringen i ”div#map-container”.
    - Sensor data som presenteras i inforutorna.
- Asynkron kommunikation med back-end

Använder sig utav *Fetch* API för asynkron kommunikation med back-end

Hanterar svar och förfrågningar från back-end

- Loggning och Respons
    - Implementation utav ett loggning system.
    - Loggning systemet skriver ut de API svar som skickas mellan front och back-end.
        - Ex. När användaren byter mellan *Auto* och *Manual*, skrivs ett meddelande ut att servern har mottagit kommandot eller inte.
- SLAM Kartvisualisering
    - Använder sig utav Plotly.js för en dynamisk kart.
    - Regelbundet uppdaterar kartan baserad på data skickat genom SSE

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

5.5 Backend

5.5.1 Översikt

I denna del kommer allt som inkluderas i back-end genomgås. Detta inkluderar hur back-end kommunicerar med front-end, de applikationer som används; main.py, rp_master.py, lidar.py, rpslam.py, autoMode.py.

I *main.py* hanteras FastAPI initieringen, HTTP förfrågningar samt svar. Skickar och ter emot data från och till front-end, med API endpoints.

I uppstarten utav hemsidan startas alla parallella processer, detta med python modulen *multiprocessing*. Detta paket gör det möjligt att köra parallella processorer genom at utnyttja alla RaspberryPi:ns kärnor.

5.5.2 Variabler

- App.FastAPI()
    - app är en instans utav klassen FastAPI. Denna variabel fungerar som kärnan till hela webapplikationen.
    - Denna variabel innehåller serverns inställningar, ruter och funktionalitet
- Process ()
    - Följande variabler initieras till *None* i början för att sedan tilldelas till ett *multiprocess process* objekt.
        - autoProcess
            - Kör den autonoma styrprocessen.
        - lidarProcess
            - Kör insamling utav data från Lidar.
        - slamProcess
            - Kör SLAM algoritmen.
- Queue()
    - *Queue()* är en variabel för att skicka data ström mellan processer. Är en del utav modulen *multiprocessing*.
    - sensorQueue:
        - Denna variabel hanterar sensor data från Lidar och odometer.
    - xythetaQueue
        - Denna variabel skickar robotens nuvarande position som beräknas i rpslam.py
    - figQueue
        - Denna variabel skickar den genererade kartan, visualiserad utav SLAM.
- Manager()
    - Manager() är en variabel som kan delas mellan parallella processer. Är en del utav modulen *multiprocessing*.
    - autoVariable
        - En processglobal *boolean* för att hålla koll på om roboten är i manuellt läge eller autonomt.
- lidarDict

    - En processglobal *dictionary*. Denna dict sparar lidar data. Grader som *keys* och korresponderande distans som *value.* Även behandlad data från odometer lagras här.
- controlVariable
    - En processglobal *dictionary* som uppdateras med styrkommandon till roboten.
- Event()
    - Används för att skicka signaler mellan processer. Kan skicka flaggor ”True” eller ”False”.
    - shutdownEvent
        - Används för att stänga ner alla parallella processer snyggt för att minimera konflikter

5.5.3 Dekoratör

Dekoratörer är en del av FastAPIs funktionalitet. Definierar slutpunkter för *HTTP requests*.

- @app.get(”/”, response_class=HTMLResponse)
    - Definerar en *HTTP GET request* till funktionen *read_root(request: Request).*
- @app.on_event(”startup”)
    - Hanterar event då applikationen startar. Definierar att funktionen *on_startup()* ska köras vid start.
- @app.on_event(”shutdown”)
    - Hanterar event då applikationen stängs av. Definierar att funktionen *on_shutdown()* ska köras vid nedstängning.
- @app.get(”/slam-visualization”)
    - Definerar en *HTTP GET request* till funktionen *get_slam_vis()*.
- @app.get(”/lws-stream”)
    - Definerar en *HTTP GET request* till funktionen *lws_stream(request: Request)*.
- @app.get(”/rws-stream”)
    - Definerar en *HTTP GET request* till funktionen *rws_stream(request: Request)*.
- @app.post(”/command/{action}”)
    - Definerar en *http POST request* till funktionen *command(action: str).*
- @app.post(”/stopDirection/{stop}”)
    - Definerar en *http POST request* till funktionen *stopDirection(stop: str).*
- @app.post(”/direction/{direction}”)
    - Definerar en *http POST request* till funktionen *sendDirection(direction: str).*
- @app.post(”/scan”)
    - Definerar en *http POST request* till funktionen *scan().*

5.5.4 Funktioner

- *async def read_root(request: Request)*
    - En asynkron function som hämtar och renderar HTML filen index.html
- *async def on_startup()*
    - En asynkron funktion som körs vid start utav applikationen.
    - Går igenom 3 if satser för att kontrollera och starta de parallella processerna lidarProcess, slamProcess och autoProcess.
        - För vardera if sats kontroleras om motsvarande process redan är aktiv, om inte, starta den.
- *def on_shutdown()*
    - Denna funktion hanterar nedstängningen utav applikationen. Främst stänger den av de parallella processerna.
        - Vardera processen kontrolleras om de är aktiva. Är de aktiva, kommer de stängas av.
- *async def get_slam_vis()*
    - Denna funktion hanterar den nestlade funktionen *map_gen()* och returnerar *yield* som funktionen genererar eller ett felmeddelande. Returen hanteras i front-end JavaScript.
    - *async def event_map_gen()*
        - Denna funktion körs då autoVariable är satt till ”True”, alltså autonomt läge.
        - Så länge autoVariable = True, hämtas kartan från figQueue.
        - Kartan görs till ett json objekt
- *async def periodic_task()*
    - Denna funktion hämtar data från odometrarna. Används främst i felsöknings syfte.
    - Kallar på sensor funktionen *read_from_sensor()*
    - Placerar korrekt den returnerade data i lwsData och rwsData.
- *async def lws_stream(request: Request)*
    - Denna asynkrona funktion hanterar den nestlade funktionen *event_stream()* och returnerar den yield funktionen genererar. Returen hanteras i front-end JavaScript. Används främst i felsökningssyfte.
    - *async def event_stream()*
        - Updaterar data till front-end hämtat av i funktionen *periodic_task().* Denna uppdaterar med lwsData.
- *async def rws_stream(request: Request)*
    - Denna asynkrona funktion hanterar den nestlade funktionen *event_stream()* och returnerar den yield funktionen genererar. Returen hanteras i front-end JavaScript. Används främst i felsökningssyfte.
    - *async def event_stream()*
        - Updaterar data till front-end hämtat av i funktionen *periodic_task().* Denna uppdaterar med rwsData.
- *async def command(action: str)*

    - En asynkron funktion som uppdaterar variabeln autoVariable. En kommandosträng skickas från front-end.
        - Strängen kan vara ”auto” eller ”manual”.
    - Beroende på kommandosträng så uppdateras autoVariable till ”True” eller ”False”.
- *async def stopDirection(stop: str)*
    - Denna asynkrona funktion hanterar stopkommando skickat från front-end.
    - Skickar vidare kommando till kommunikations funktionen write_to_styr()
        - Skickar en lista med följande värden; [0,0,0,0]
- *async def sendDirection(direction: str)*
    - Denna synkrona funktion hanterar styrkommandon skickade från front-end.
    - Denna funktion används under det manuella styr läget.
    - Använder sig utav en if-sats för att skicka vidare korrekt kommando till styrmodulen.
        - Skickar med kommunikations funktionen write_to_styr().
        - Finns 8 kommandon; forward, back, right, left, forward-right, forward-left, back-left och back-right.
- *async def scan()*
    - Denna asynkrona funktion hanterar kommando att utföra en scanning med lidar.
    - Kallar sensor funktionen scanDistanceAngle().
    - Används främst i felsökningsyfte och produktion.

5.5.4.1 main.py

Kodförklaring: ett Python-skript som definierar en FastAPI-webbapplikation för styrning och övervakning av en robot.

För fullständig kod se relevant dokumentationssektion på Confluence alternativt Gitlab.

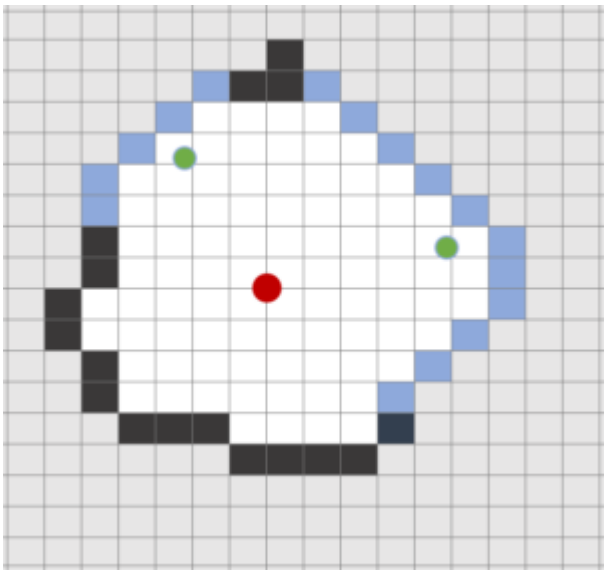
6 Algoritmer

6.1 Översikt

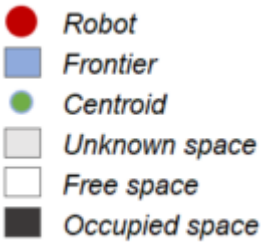
Frontier based exploration handlar om två koncept, “Occupation grid“ och “Frontiers“ [1]

6.1.1 Occupation grid

Occupation grid är ett sätt att representera miljön robotens sensorer (Lidar oftast för bäst resultat) skapar. Den delar in utrymmet i lika stora celler som sen kan representeras som fria, okupierade, Frontiers eller oskannade.



Figur 2: Occupation grid



Figur 3: Occupation grid: color symbols

Den representationen innehåller 6 variabler:

- **Robot:** robotens plats i jämförese med den skannade kartan
- **Frontier:** arean som separerar mellan “Free space“ och “Unknown space“
- **Centroid:** mellanpunkten av en “Frontier“
- **Unknown space:** arean som inte har skannats än
- **Free space:** arean som är skannat och är fri från hider
- **Occupied space:** arean som är skannat och är okupierad av ett hinder

6.1.2 Frontiers

En frontier är arean som delar mellan de fria och de oskannade cellerna. Frontier representeras av 6 variabler:

- En lista av angränsande fria celler
- storleken, dvs hur många celler en frontier består av
- ett avstånd. Beskrivs som “the Euclidean distance“, som är den kortaste distansen mellan roboten och frontier:en i meter.
- en orientation, som är hur många grader mellan roboten håll och frontier:en
- en centerpunkt, som är en punkt som är i mitten av en frontier
- ett pris/kostnad. Priset i den här sammanhanget är hur intressant det är att gå till en viss frontier.

6.1.3 Algoritmen

Algoritmen funkar så att roboten skannar och får ett val av frontiers, sen väljer den en frontier baserad på kostnaden mellan de olika frontiers. Den fortsätter så tills det finns inga frontiers kvar.

Valet av frontiers baseras på dens kostnad. Den frontier:en som har lägst kostnad väljs, där kostnaden beräknas med formeln:

$$C(f) = \lambda 1 Distance(f) - \lambda 2 Size(f) + \lambda 3 Orientation(f)$$

Ekvation 1: Frontiers kostnadsberäkning

där f är en frontier av kartan och $\lambda 1, \lambda 2, \lambda 3$ är variabler för att bestämma prioteringen. Formeln som bas (om $\lambda 1, \lambda 2, \lambda 3 = 1$) betyder att ju närmare, ju större och ju mindre vinkel som roboten behöver vända för en frontier ju mer intressant en frontier är. Variablerna är justerbara för att priotera de parametrar som är viktigare.

För mer info se: Autonom styrning

6.2 A* (A-star)

6.2.1 Översikt

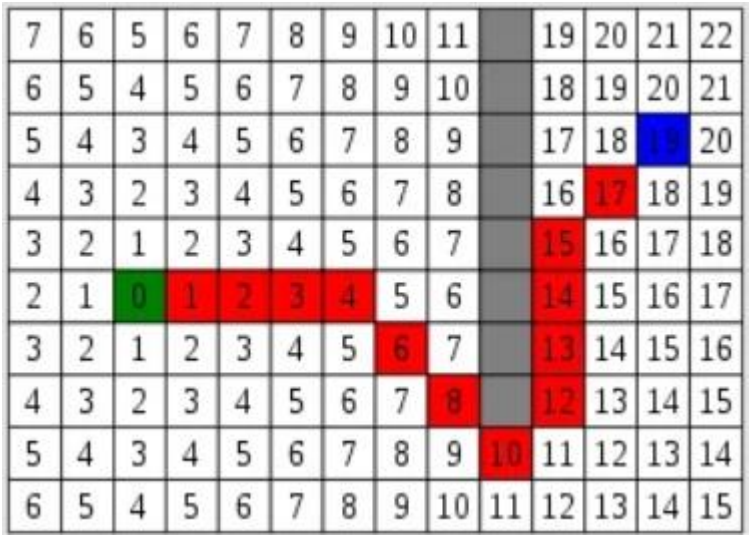
A* (uttalas "A-star") [1] [2] är en algoritm för graf-traversering och sökning efter vägar, vilken används inom många områden inom datavetenskap som följd av dess fullständighet, optimalitet och effektivitet. Med en viktad graf, en start-nod och en mål-nod hittar algoritmen den kortaste vägen (med avseende på de givna vikterna) från start till mål.

6.2.2 F = G + H

En viktig aspekt av A* är att F = G + H. Variablerna F, G och H finns i en nod-klass och beräknas varje gång en ny nod skapas. Dessa variabler betyder:

- F är den totala kostnaden för noden.
- G är avståndet mellan den aktuella noden och start-noden.
- H är heuristiken — den uppskattade sträckan från den aktuella noden till slut-noden.

Figur 3 illustrerar processen.



Figur 4: Värdering av olika noder

6.2.3 A* Metodsteg

1. Lägg till startrutan (eller noden) i öppna listan.
2. Upprepa följande:

A) Sök efter den ruta med lägst F-kostnad på öppna listan. Detta benämns som den aktuella rutan.

B) Flytta den till den stängda listan.

C) För varje av de 8 rutorna intill den aktuella rutan...

- Om den inte går att passera eller om den redan finns på den stängda listan, ignorera den. Annars utför följande steg.
- Om den inte redan finns på den öppna listan, lägg till den där. Gör den aktuella rutan till förälder till denna ruta. Registrera F-, G- och H-kostnaderna för rutan.
- Om den redan finns på den öppna listan, kontrollera om vägen till rutan är bättre, med G-kostnaden som mått. En lägre G-kostnad innebär att detta är en bättre väg. I sådant fall ändra föräldern till rutan till den aktuella rutan och beräkna om G- och F-poängen för rutan. Om öppna listan behålls sorterad efter F-poängen kan det vara nödvändigt att omorganisera listan för att beakta förändringen.

D) Stanna när algoritmen:

- Lägger till målrutan i den stängda listan, i vilket fall vägen har hittats, eller
- Inte kan hitta målrutan och den öppna listan är tom. I detta fall finns ingen väg.

3. Spara vägen. Arbeta bakåt från målrutan, gå från varje ruta till dess förälder-ruta tills startrutan är nådd.

För mer info se: Autonom styrning

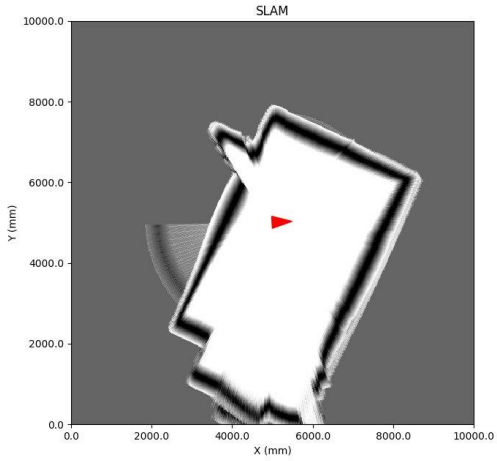
6.3 BreezySLAM

6.3.1 Översikt

BreezySLAM [1] är en enkel, effektiv, plattformsoberoende och öppen Python-bibliotek för Simultaneous Localization and Mapping (SLAM). Genom att använda ett Python C-tillägg för att omsluta befintliga implementationer av SLAM-algoritmer erbjuder BreezySLAM en Python API för SLAM som körs nästan lika snabbt som den ursprungliga C-koden. Genom att göra SLAM API tillgänglig i Python kan studenter och andra intresserade användare snabbt och effektivt få tillgång till SLAM.

6.3.2 Användningsområde

En av de framstående funktionerna är dess effektivitet vid bearbetning av SLAM-funktionalitet. Genom att omfamna befintliga SLAM-algoritmer och implementeringar och tillhandahålla en Python API möjliggör BreezySLAM snabb och effektiv simultan kartläggning och lokaliseringsfunktionalitet.



Figur 5: Rum kartlagt med BreezySLAM

6.3.3 BreezySLAM Metodsteg

1. **Inläsning av Sensordata:**
    - BreezySLAM börjar genom att läsa in sensordata från den givna Lidar.
    - Data från sensor, som består av avståndsmätningar från omgivande objekt, används som ingångsinformation för kartläggning och lokalisering.
2. **Initialisering:**
    - En karta och en partikeluppsättning (för att representera möjliga positioner) initialiseras.
    - Varje partikel representerar en möjlig position och orientering för roboten.
3. **Partikelfiltrering (Particle Filtering):**
    - BreezySLAM använder partikelfiltrering för att uppskatta robotens position och orientering i förhållande till den initiala kartan.
    - Partiklar flyttas och uppdateras baserat på data från sensor för att optimera positionen.
4. **Association av Data:**
    - Data från sensor associeras med den befintliga kartan för att bestämma vilka mätningar som hör till vilka objekt i kartan.
    - Detta steg bidrar till att uppdatera partiklarna och förbättra noggrannheten i positionsskattningarna.
5. **Uppdatering av Kartan:**
    - Kartan uppdateras med nya mätningar från Lidar.
    - Detta inkluderar att lägga till nya strukturer eller uppdatera befintliga objekt baserat på de nya sensormätningarna.
6. **Optimering:**
    - För att förbättra noggrannheten och konsekvensen i positionsskattningarna genomförs optimering.
    - Optimeringen kan innefatta justering av partikelpositioner och orienteringar för att minska fel och förbättra överensstämmelsen med data från sensor.
7. **Återupprepning:**
    - De föregående stegen upprepas kontinuerligt när nya sensormätningar blir tillgängliga.
    - Genom att uppdatera partiklar, kartan och positionen regelbundet anpassar sig algoritmen till förändringar i omgivningen.
8. **Resultat och Slutlig Position:**
    - Efter ett antal iterationer ger BreezySLAM en slutgiltig uppskattning av robotens position och en kartbild över dess omgivning.

7 Referenser

Uppdaterad kod finns på gruppens Gitlab repo.

https://gitlab.liu.se/da-proj/microcomputer-project-laboratory-d/2023/g08/docs/-/tree/main?ref_type=heads

7.1 Kommunikationsmodul

7.1.1 12C

[1] "ATmega1284P." Tillgänglig: https://ww1.microchip.com/downloads/en/DeviceDoc/doc8059.pdf

[2] "I2C - http://learn.sparkfun.com ," http://Sparkfun.com , 2018. https://learn.sparkfun.com/tutorials/i2c/all

[3] K.-P. Lindegaard, "smbus2: smbus2 is a drop-in replacement for smbus-cffi/smbus-python in pure Python," *PyPI*. https://pypi.org/project/smbus2/

7.2 Sensormodul

[1] Microchip Technologies, "ATmega1284p," Tillgänglig: https://ww1.microchip.com/downloads/en/DeviceDoc/doc8059.pdf

[2] Raspberry Pi LTD, "Raspberry Pi 3 Model B," Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[3] Linköpings universitet, "Reflexsensormodul," Tillgänglig: https://da-proj.gitlab-pages.liu.se/vanheden/pdf/reflexsensormodul.pdf

[4] "Slamtec RPLIDAR A2 - SLAMTEC Global Network," https://www.slamtec.ai/product/slamtec-rplidar-a2/

[5] "GP2D120 Optoelectronic Device FEATURES." Tillgänglig: https://www.pololu.com/file/0J157/GP2D120-DATA-SHEET.pdf

[6] "Sharp GP2Y0A21 IR Distance Sensor (10-80cm) For Arduino - DFRobot," http://www.dfrobot.com . https://www.dfrobot.com/product-328.html

7.3 Styrmodul

[1] "Basics of PWM (Pulse Width Modulation) | Arduino Documentation," *docs.arduino.cc*. https://docs.arduino.cc/learn/microcontrollers/analog-output

7.4 Gränsnitt

7.4.1 FastAPI

[1] FastAPI, "FastAPI," http://fastapi.tiangolo.com . https://fastapi.tiangolo.com/

7.5 Algoritmer

7.5.1 BreezySLAM

[1] S. D. Levy, “BreezySLAM,” *GitHub*, Dec. 20, 2023. https://github.com/simondlevy/BreezySLAM (Åtkomst Dec. 21, 2023).

7.5.2 A*(A-star)

[1] N. Swift, “Easy A* (star) Pathfinding,” *Medium*, Feb. 27, 2017.

[2] Wikipedia Contributors, “A* search algorithm,” *Wikipedia*, Mar. 10, 2019.

7.5.3 Frontier

[1] Frontier: awabot (2021), Autonomous Mobile Robot (AMR): focus on autonomous exploration

8 Appendix

8.1 Kopplingsschema

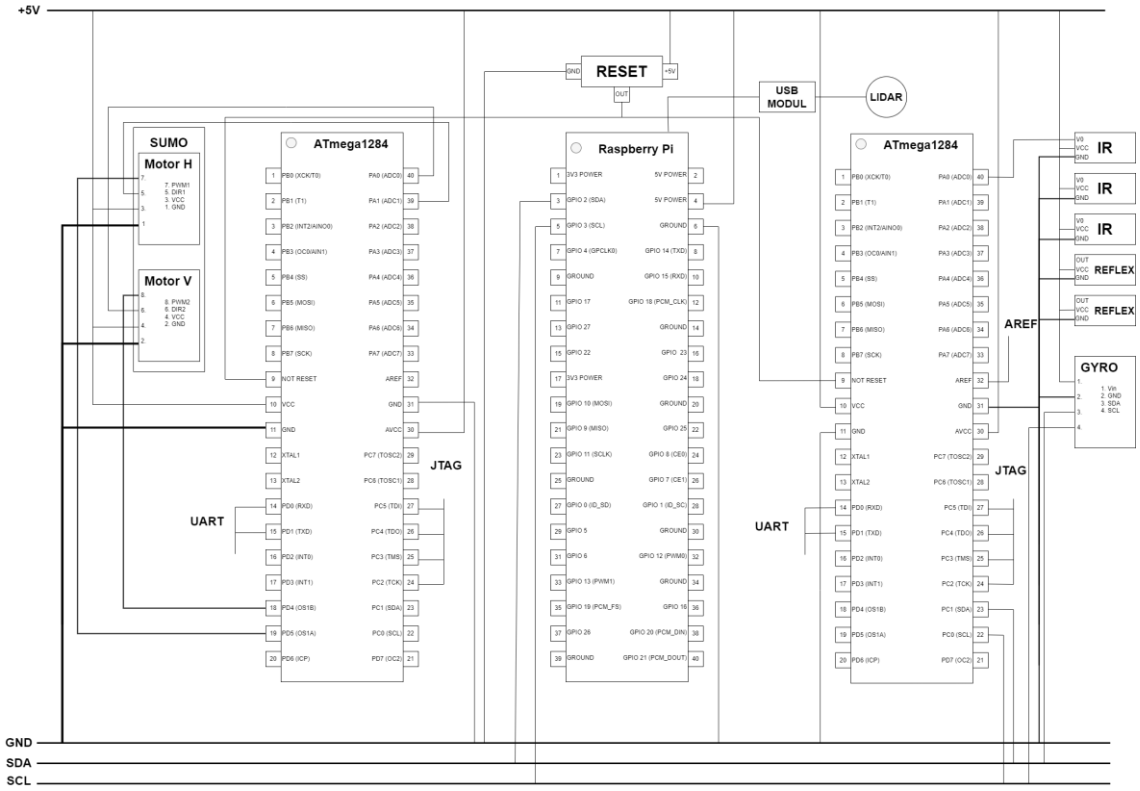
8.1.1 Översikt

- ATmega1284p (vänster) - Styrmodul
- ATmega1284p (höger) - Sensormodul
- Raspberry Pi 3 - Kommunikationsmodul

8.1.2 Diagram

8.1.2.1 Översikt

Figur 5 visar en översikt av kopplingsschemat för systemet. Raspberry Pi 3 fungerar som systemets hjärna och är processorn som bearbetar all information som systemet behandlar via kommunikationsprotokollet I2C. Det finns två ATmega1284p i systemet. Den ATmega1284p som kan ses i den vänstra delen av kopplingsschemat är det delsystem som ansvarar för att systemet rör sig via fyra växlade DC-motorer (7.2V, 291 RPM). Motorerna styrs parvis med två signaler per sida. DIR, som styr motorernas rotationsriktning och PWM, pulsbreddsmodulering som styr motorernas hastighet. Den ATmega1284p som kan ses på den högra sidan av kopplingsschemat är det delsystem som ansvarar för de sensorer som systemet behöver för att genomföra dess autonoma styrning (tillsammans med styrmodulen). Mer ingående beskrivning om vad varje modul gör finns beskrivet i respektive delkapitel.



Figur 5: Översikt av kopplingsschemat