Kod

TSEA29

Grupp 8

Exported on 2023-12-21 09:00:27

Table of Contents

1 Kommunikationsmodul kod

1.1 I2C kod

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.

1.1.1 Styr (slave) kod

main.c

```
/*
 * i2c_styr_v1.c
 *
 * Created: 2023-11-16 15:50:42
 * Author : alija148
 */
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/twi.h>

#define STYR_SLAVE_ADDRESS 0x42
#define BUFFER_SIZE 8
volatile uint8_t styr_data_buffer[BUFFER_SIZE];
volatile uint8_t received_data_buffer[BUFFER_SIZE];
volatile uint8_t styr_index = 0;
volatile uint8_t received_index = 0;
/*
TWCR(TWI Control Register): TWINT|TWEA|TWSTA|TWSTO|TWWC|TWEN|- |TWIE|
7=TWI_Intrrupt_Flag, 6= TWI_Enable_Acknoledge_Bit,
5=TWI_Start_Condition_Bit
4= TWI_Stop_Condition_Bit, 3=TWI_Write_Collision_Flag, 2=TWI_Enable_Bit,
1=Reserved_Bit
0 =TWI_Interrupt Enable.

*/


void init_Styr_Slave() {
    // Sätt (TWEN) TWI Enable register=1,(TWEA)TWI Enable Acknowledge
Bit=1,(TWSTA) TWI Start condition Bit=0,(TWSTO) TWI Stop Condition Bit=0,
    TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT) | (1 << TWIE);
    // Sätt den egna I2C-adressen
    TWAR = STYR_SLAVE_ADDRESS << 1;
    sei();
}

void send_Styr_Data() {
    // Check if TWI is ready to transmit
    if ((TWCR & (1 << TWINT))) {
        // Load the data from the buffer to TWDR using styr_index
        TWDR = styr_data_buffer[styr_index];
        // Clear TWINT to start the data transmission
        TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWIE);
        // Increment the styr_index for the next styr's data
        styr_index++;

        // Check if we have reached the end of the buffer
        if (styr_index == BUFFER_SIZE) {
            // Reset the index to start from the beginning of the buffer
            styr_index = 0;
        }
    }
}

void receive_Data() {
    // Receive data and store it in the buffer using received_index
    received_data_buffer[received_index] = TWDR;
    // Increment the received_index for the next received byte
    received_index++;

    // Check if we have reached the end of the buffer
    if (received_index < BUFFER_SIZE) {
        // If it's not the last byte, send ACK and clear TWINT to start
the data transmission
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
```

```
        }
    else {
        // If it's the last byte, prepare to send NACK and clear TWINT
        TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 << TWIE);
        // Reset the index to start from the beginning of the buffer
        received_index = 0;
    }
}



ISR(TWI_vect) {
    //TWI status flags
    switch (TW_STATUS) {
        //Case TWI slave transfer slave_address (0xA8)
        case TW_ST_SLA_ACK:
        // Load the data to be sent
        TWDR = styr_data_buffer[styr_index];
        // Print received SLA for debugging
        printf("Received SLA: %x\n", TWAR >> 1);
        // ...
        //checking if the styr_index reached to BUFFER_SIZE
        if(styr_index == BUFFER_SIZE){
            styr_index = 0;
        }
        else
        // Increment the index for the next byte
        styr_index++;
        // Set TWI status to transmit mode
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;

        //Case (0xB0) Arbitration lost in Slave Transmit, SLA received
with ACK
        case TW_ST_ARB_LOST_SLA_ACK:
            if (styr_index == BUFFER_SIZE - 1) {
                TWDR = styr_data_buffer[styr_index];
                // If it's the last byte, prepare to send NACK
                TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
                }
            else {
                TWDR = styr_data_buffer[styr_index];
                // If it's not the last byte, send ACK
                TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
        break;

        //Case (0xB8) TWI slave transfer data
        case TW_ST_DATA_ACK:
            if (styr_index == BUFFER_SIZE - 1) {
                TWDR = styr_data_buffer[styr_index];
                // If it's the last byte, prepare to send NACK
                TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
            else {
                TWDR = styr_data_buffer[styr_index];
                // If it's not the last byte, send ACK
                TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
        //case (0xC0)
        case TW_ST_DATA_NACK:
```

```
            // Release the TWI peripheral to be ready for a new start
condition
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);

            break;
        //case 0xC8
        case TW_ST_LAST_DATA:

            TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT) | (1 << TWIE);
            break;

        //case 0x60 TWI slave receiver slave_address
        case TW_SR_SLA_ACK:
            // Set TWI status to receive mode with acknowledgment (ACK)
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
            break;

        //Case 0x68
        case TW_SR_ARB_LOST_SLA_ACK:
             // Release the TWI peripheral to be ready for a new start
condition
             TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
        break;
        //case 0x70
        case TW_SR_GCALL_ACK:

        break;
        //case 0x78
        case TW_SR_ARB_LOST_GCALL_ACK:

        break;

        //case 0x80
        case TW_SR_DATA_ACK:
        // Read received data and send ACK for the next byte
        received_data_buffer[received_index] = TWDR;

        // Increment the index for the next received byte
        received_index++;

        if (received_index < (BUFFER_SIZE - 1)) {
            // If it's not the last byte, send ACK
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
            }
        else {
            // If it's the last byte, prepare to send NACK
            TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 << TWIE);
        }
        break;

        //case 0x88
        case TW_SR_DATA_NACK:
        // Receive data
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //case 0x90
        case TW_SR_GCALL_DATA_ACK:
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;

        //case 0x98
```

```
        case TW_SR_GCALL_DATA_NACK:
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //case 0xF8
        case TW_NO_INFO:

        break;
        //case 0x00
        case TW_BUS_ERROR:
        // Release the internal hardware and clear TWSTO
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //In the case where a STOP condition or repeated START condition
has been received while still addressed as a slave
        case TW_SR_STOP:
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);

        break;

        default:
            init_Styr_Slave();

        break;
    }

    // Clear the TWI interrupt flag
    TWCR |= (1 << TWINT);
}

int main () {

    init_Styr_Slave();
    while(1){
        //IF no conversion is active check.

    }
}
```

1.1.2 Sensor (slave) kod

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

```c
/*
 * i2c_sensor_v1.c
 *
 * Created: 2023-11-16 15:50:42
 * Author : alija148
 */
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/twi.h>

#define SENSOR_SLAVE_ADDRESS 0x42
#define BUFFER_SIZE 32
volatile uint8_t sensor_data_buffer[BUFFER_SIZE];
volatile uint8_t received_data_buffer[BUFFER_SIZE];
volatile uint8_t sensor_index = 0;
volatile uint8_t received_index = 0;
/*
TWCR(TWI Control Register): TWINT|TWEA|TWSTA|TWSTO|TWWC|TWEN|- |TWIE|
7=TWI_Intrrupt_Flag, 6= TWI_Enable_Acknoledge_Bit,
5=TWI_Start_Condition_Bit
4= TWI_Stop_Condition_Bit, 3=TWI_Write_Collision_Flag, 2=TWI_Enable_Bit,
1=Reserved_Bit
0 =TWI_Interrupt Enable.

*/


void init_Sensor_Slave() {
    // Sätt (TWEN) TWI Enable register=1,(TWEA)TWI Enable Acknowledge
Bit=1,(TWSTA) TWI Start condition Bit=0,(TWSTO) TWI Stop Condition Bit=0,
    TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT) | (1 << TWIE);
    // Sätt den egna I2C-adressen
    TWAR = SENSOR_SLAVE_ADDRESS << 1;
    sei();
}

void send_Sensor_Data() {
    // Check if TWI is ready to transmit
    if ((TWCR & (1 << TWINT))) {
        // Load the data from the buffer to TWDR using sensor_index
        TWDR = sensor_data_buffer[sensor_index];
        // Clear TWINT to start the data transmission
        TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWIE);
        // Increment the sensor_index for the next sensor's data
        sensor_index++;

        // Check if we have reached the end of the buffer
        if (sensor_index == BUFFER_SIZE) {
            // Reset the index to start from the beginning of the buffer
            sensor_index = 0;
        }
    }
}

void receive_Data() {
    // Receive data and store it in the buffer using received_index
    received_data_buffer[received_index] = TWDR;
    // Increment the received_index for the next received byte
    received_index++;

    // Check if we have reached the end of the buffer
    if (received_index < BUFFER_SIZE) {
        // If it's not the last byte, send ACK and clear TWINT to start
the data transmission
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
```

```
        }
    else {
        // If it's the last byte, prepare to send NACK and clear TWINT
        TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 << TWIE);
        // Reset the index to start from the beginning of the buffer
        received_index = 0;
    }
}



ISR(TWI_vect) {
    //TWI status flags
    switch (TW_STATUS) {
        //Case TWI slave transfer slave_address (0xA8)
        case TW_ST_SLA_ACK:
        // Load the data to be sent
        TWDR = sensor_data_buffer[sensor_index];
        // Print received SLA for debugging
        printf("Received SLA: %x\n", TWAR >> 1);
        // ...
        //checking if the sensor_index reached to BUFFER_SIZE
        if(sensor_index == BUFFER_SIZE){
            sensor_index = 0;
        }
        else
        // Increment the index for the next byte
        sensor_index++;
        // Set TWI status to transmit mode
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;

        //Case (0xB0) Arbitration lost in Slave Transmit, SLA received
with ACK
        case TW_ST_ARB_LOST_SLA_ACK:
            if (sensor_index == BUFFER_SIZE - 1) {
                TWDR = sensor_data_buffer[sensor_index];
                // If it's the last byte, prepare to send NACK
                TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
                }
            else {
                TWDR = sensor_data_buffer[sensor_index];
                // If it's not the last byte, send ACK
                TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
        break;

        //Case (0xB8) TWI slave transfer data
        case TW_ST_DATA_ACK:
            if (sensor_index == BUFFER_SIZE - 1) {
                TWDR = sensor_data_buffer[sensor_index];
                // If it's the last byte, prepare to send NACK
                TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
            else {
                TWDR = sensor_data_buffer[sensor_index];
                // If it's not the last byte, send ACK
                TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
            }
        //case (0xC0)
        case TW_ST_DATA_NACK:
```

```
            // Release the TWI peripheral to be ready for a new start
condition
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);

            break;
        //case 0xC8
        case TW_ST_LAST_DATA:

            TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT) | (1 << TWIE);
            break;

        //case 0x60 TWI slave receiver slave_address
        case TW_SR_SLA_ACK:
            // Set TWI status to receive mode with acknowledgment (ACK)
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
            break;

        //Case 0x68
        case TW_SR_ARB_LOST_SLA_ACK:
             // Release the TWI peripheral to be ready for a new start
condition
             TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 <<
TWIE);
        break;
        //case 0x70
        case TW_SR_GCALL_ACK:

        break;
        //case 0x78
        case TW_SR_ARB_LOST_GCALL_ACK:

        break;

        //case 0x80
        case TW_SR_DATA_ACK:
        // Read received data and send ACK for the next byte
        received_data_buffer[received_index] = TWDR;

        // Increment the index for the next received byte
        received_index++;

        if (received_index < (BUFFER_SIZE - 1)) {
            // If it's not the last byte, send ACK
            TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
            }
        else {
            // If it's the last byte, prepare to send NACK
            TWCR = (1 << TWINT) | (0 << TWEA) | (1 << TWEN) | (1 << TWIE);
        }
        break;

        //case 0x88
        case TW_SR_DATA_NACK:
        // Receive data
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //case 0x90
        case TW_SR_GCALL_DATA_ACK:
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;

        //case 0x98
```

```
        case TW_SR_GCALL_DATA_NACK:
        receive_Data();
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //case 0xF8
        case TW_NO_INFO:

        break;
        //case 0x00
        case TW_BUS_ERROR:
        // Release the internal hardware and clear TWSTO
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);
        break;
        //In the case where a STOP condition or repeated START condition
has been received while still addressed as a slave
        case TW_SR_STOP:
        // Set TWI status to ready for a new start condition
        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) | (1 << TWIE);

        break;

        default:
            init_Sensor_Slave();

        break;
    }

    // Clear the TWI interrupt flag
    TWCR |= (1 << TWINT);
}

int main () {

    init_Sensor_Slave();
    while(1){
        //IF no conversion is active check.

    }
}
```

2 Sensormodul kod

```
#include "init.h"#include "sensor_slave.h"void init(){ wdt_disable();
   initilize_ports(); init_ADC(); //interupt_init();
   init_Sensor_Slave();  //sensor_index=0;  //received_index=0;
   timer_setup();sei();  lastADCReflex1 = STATE_FIRST_RUN;lastADCReflex2
= STATE_FIRST_RUN;}//CTC timer for wheel measurment (16-bit
timer/counter)void timer_setup(){// Set timer to CTC mode  TCCR1B |=
(1 << WGM12);  // Enable interrupt on compare match  TIMSK1 |= (1 <<
OCIE1A);  // Set compare match value for 1 kHz interval   OCR1A = 15999;
// (16 MHz / (1 kHz * 1 prescaler)) - 1  // Set prescaler to 1 and
start the timer  TCCR1B |= (1 << CS10);  TCCR1B &= ~((1 << CS12) | (1
<< CS11));}void initilize_ports(){  // Initilize the input pins to IR
sensors and reflex sensorsDDRA &= ~(1 << IR1_30_INPUT_PIN | 1 <<
IR2_30_INPUT_PIN |1 << IR_80_INPUT_PIN);//-----------------------------
--------------------------------- //Initilize digital pins for reflex
sensor DDRA &= ~(1 << REFLEX1_INPUT_PIN) | (1 << REFLEX2_INPUT_PIN);//
Enable pull-up resistor for above PORTA |= (1 << REFLEX1_INPUT_PIN) |
(1 << REFLEX2_INPUT_PIN);  //--------------------------------------------
------------------ // Add Unused pins //IF more pins need to be
activated}// Initializing for ADCvoid init_ADC(){ // Set 5V internal
ref  ADMUX &= ~(1<< REFS1);  ADMUX |= (1<< REFS0);// Set Result left
adjusted -- changed to right ADMUX &= ~(1 << ADLAR); // Activate ADC
   ADCSRA |= (1 << ADEN);  //Disable auto-trigger  ADCSRA &=
~(1<<ADATE);   //Enable interupt from ADC,  ADCSRA |= (1 << ADIE);
   //Set ADC frequency division factor ADCSRA |= (1 << ADPS2) | (1 <<
ADPS1);// ADSC -set to one for start conversion, ADSC will read as
high as long as conversion is active}/*
// Interupt init for odometer pins
void interupt_init(){
   //------------------------------------------------------------
   //Testing interupt on digital pins
   PCICR  |= (1 << PCIE0 );
   PCMSK0 |= (1 << PCINT6) | (1 << PCINT7);


}
*/
```

```
#ifndef INIT_H#define INIT_H#include "main.h"#define IR1_30_INPUT_PIN
PA0  // Define PA0 as input pin for IR-sensor 1 30cm#define
IR1_30_OUTPUT_PIN PA1 // Define PA1 as output pin for IR-sensor 1
30cm#define IR2_30_INPUT_PIN PA2 // Define PA2 as input pin for IR-
sensor 2 30cm#define IR2_30_OUTPUT_PIN PA3   // Define PA3 as output
pin for IR-sensor 2 30cm#define IR_80_INPUT_PIN PA4  // Define PA4 as
input pin  for IR-sensor 80cm#define IR_80_OUTPUT_PIN PA5 // Define
PA5 as output pin  for IR-sensor 80cm#define REFLEX1_INPUT_PIN PA6 //
Define PA6 as input pin  for reflexsensor 1#define REFLEX2_INPUT_PIN
PA7 // Define PA7 as input pin  for reflexsensor 2//#define
REFLEX1_INPUT_PIN_DIGI PD2 // Define PA6 as input pin  for
reflexsensor 1//#define REFLEX2_INPUT_PIN_DIGI PD3 // Define PA7 as
input pin  for reflexsensor 2//Initializervoid init(void);void
initilize_ports(void);void init_ADC(void);//void
interupt_init(void);void timer_setup(void);  #endif
```

```
/*
 * sensormodul_atmel.c
 *
 * Created: 2023-10-30 14:38:10
 * Author : thewe344
 */
```

```
#include "main.h"#include "init.h"//#include "odometer.h"#include
"sensor_slave.h"ISR(TIMER1_COMPA_vect);//ISR(PCINT0_vect);volatile
uint8_t pinTracker = 0;void packData(volatile uint16_t
leftWheelDistance, volatile uint16_t rightWheelDistance){ //volatile
uint8_t ircntr, volatile float adcData, uint8_t highByteLeftDist =
(uint8_t)(leftWheelDistance >> 8); // Split data up to two bytes, high
part uint8_t lowByteLeftDist = (uint8_t)(leftWheelDistance  & 0xFF); //
Split data up to two bytes, low part  uint8_t highByteRightDist =
(uint8_t)(rightWheelDistance >> 8); // Split data up to two bytes,
high part uint8_t lowByteRightDist = (uint8_t)(rightWheelDistance  &
0xFF); // Split data up to two bytes, low part
  sensor_data_buffer[0] = highByteLeftDist;  sensor_data_buffer[1] =
lowByteLeftDist; sensor_data_buffer[2] = highByteRightDist;
  sensor_data_buffer[3] = lowByteRightDist;}// Need to implement
when/how data is sent to RPI:n, that correct data is sent, and cleared
after so correctly updatedint main () {  MCUSR_data |= MCUSR;  DDRB =
0XFF;  pinTracker = 0;  init();leftWheelDistance = 0;
  rightWheelDistance = 0; reflex1Counter = 0;   reflex2Counter = 0;
  while(1){   if (pinTracker == 0){
  adcPortEnableAndConvert(REFLEX1_INPUT_PIN); //Check right odo
  }else if (pinTracker == 1)
  adcPortEnableAndConvert(REFLEX2_INPUT_PIN);//check left odo'
  packData(leftWheelDistance, rightWheelDistance);  }
  }ISR(BADISR_vect) { while(1); }//This reads selected IR sensor from
the pinsvoid adcPortEnableAndConvert(volatile uint8_t input_pin){
  //Set input channel for which IR sensor to ADMUX  ADMUX |= (ADMUX &
0xF8) | (input_pin & 0x07); //First expression stores values set on
the high bits, second value select channel where result will come //
Start ADC conversion ADCSRA |= (1 << ADSC);}//To read input from ADC,
IR sensors or reflex sensorsISR(ADC_vect) {  volatile uint16_t
adcValue = 0; //Read ADC result  adcValue = ADC;  //adcValue |= (ADCH
<< 8); ADCSRA &= ~(1 << ADSC); if (pinTracker == 0){   if ((adcValue >
500) && ((lastADCReflex1 == STATE_LOW ) || (lastADCReflex1 ==
STATE_FIRST_RUN))){       reflex1Counter = reflex1Counter + 1;
  lastADCReflex1 = STATE_HIGH;      }else if ((adcValue < 307) &&
((lastADCReflex1 == STATE_HIGH)|| (lastADCReflex1 ==
STATE_FIRST_RUN))){       reflex1Counter = reflex1Counter +1;
  lastADCReflex1 = STATE_LOW;    }  }  if(pinTracker == 1){    if
((adcValue > 500) && ((lastADCReflex2 == STATE_LOW ) ||
(lastADCReflex2 == STATE_FIRST_RUN))){       reflex2Counter =
reflex2Counter+ 1;      lastADCReflex2 = STATE_HIGH;      }else if
((adcValue < 307) && ((lastADCReflex2 == STATE_HIGH)|| (lastADCReflex2
== STATE_FIRST_RUN))){       reflex2Counter =reflex2Counter + 1;
  lastADCReflex2 = STATE_LOW;    }  }  if (pinTracker == 1){
  pinTracker = 0;    }else{    pinTracker = pinTracker + 1; }  }void
reset_distance_buff(){  reflex1Counter = 0;   reflex2Counter =
0;}ISR(TIMER1_COMPA_vect){uint16_t numberOfMarks = 10; // Number of
marks on wheeluint16_t wheelDiameter = 62; // Diameter of wheel in mm
  //uint16_t timeInterval = ; // Time interval to measure wheel float
PI = 3.141; uint16_t leftRotations = reflex1Counter /
(numberOfMarks*2); uint16_t rightRotations = reflex2Counter /
(numberOfMarks*2); //float rpmLeft = leftRotations *
(60/timeInterval); //float rpmRight = rightRotations *
(60/timeInterval); uint16_t leftWheelDistanceF = leftRotations * PI *
wheelDiameter;uint16_t rightWheelDistanceF = rightRotations * PI *
wheelDiameter;//leftWheelSpeed = (uint8_t)rpmLeft;  //rightWheelSpeed
= (uint8_t)rpmRight;
  leftWheelDistance = leftWheelDistanceF + leftWheelDistance;
  rightWheelDistance = rightWheelDistanceF + rightWheelDistance;
  //Reset wheel couter //reflex1Counter = 0; //reflex2Counter = 0;
  //Reset timer TCNT1 = 0;}/*
```

```
ISR(PCINT0_vect){

  uint8_t changedPins = PINA ^ lastPinState;
  lastPinState = PINA;

  if (changedPins & (1 << REFLEX1_INPUT_PIN)){
    adcPortEnableAndConvert(REFLEX1_INPUT_PIN);
  }

  if (changedPins & (1 << REFLEX2_INPUT_PIN)){
    adcPortEnableAndConvert(REFLEX2_INPUT_PIN);
  }
}



*//*
//Interupt handler for PD2 aka reflex sensor 1
ISR(PCINT6){

  reflex1Counter += 1;

}

//Interupt handler for PD3 aka reflex sensor 2
ISR(PCINT7){

  reflex2Counter += 1;

}
*/
```

```
#ifndef MAIN_H#define MAIN_H#include <avr/wdt.h>#include
<avr/io.h>#include <avr/interrupt.h> #include <stdio.h>#define
STATE_FIRST_RUN 3#define STATE_HIGH 1#define STATE_LOW 0// Debugging
variablevolatile uint8_t MCUSR_data;volatile uint8_t lastPinState;//
adcData will store ir sensor valuevolatile float adcData;volatile
uint8_t lastADCReflex1;volatile uint8_t lastADCReflex2;// Counter for
keeping track of What IR sensor is read fromvolatile uint8_t ircntr;/*
uint8_t numberOfMarks = 10; // Number of marks on wheel
float wheelDiameter = 6.2; // Diameter of wheel in cm
float timeInterval = 1.0; // Time interval to measure wheel
float PI = 3.14159265358979323846 2643;
*/volatile uint16_t reflex1Counter;volatile uint16_t reflex2Counter;//
Left wheel informationvolatile uint8_t leftWheelSpeed; //This is a
slave variable and will be sent to mastervolatile uint16_t
leftWheelDistance; //This is a slave variable and will be sent to
master//Right wheel informationvolatile uint8_t rightWheelSpeed;
//This is a slave variable and will be sent to mastervolatile uint16_t
rightWheelDistance; //This is a slave variable and will be sent to
master//ISR(PCINT6);//ISR(PCINT7);//Send IR Pulse, activate
adcconverter on correct portvoid adcPortEnableAndConvert(volatile
uint8_t);//Handles rouge interuptsISR(BADISR_vect);//Handles interupts
from adc, this will be used to read IR sensor valueISR(ADC_vect);void
reset_distance_buff(void);// Function to pack data in to 32bitvoid
packData(volatile uint16_t, volatile uint16_t);//, volatile uint8_t ,
volatile uint8_t#endif
```

2.1 LIDAR kod

```
from rplidar import RPLidar
from app.rp_master import read_from_sensor


#lidar = RPLidar('/dev/ttyUSB0')
#info = lidar.get_info()
#print(info)
#health = lidar.get_health()
#print(health)

########################################################################
# Small functions to handle on/off/disconnect lidar
def lidarDisconnect(lidar):
    lidar.stop()
    lidar.stop_motor()
    lidar.disconnect()

def stopLidar(lidar):
    lidar.stop()
    lidar.stop_motor()


def lidarStart(lidar):
    lidar.start_motor()
    return "Starting motor"
########################################################################

def get_and_calc_odometer(controlVariable):
    wheelBase = 100 #Distance between center of wheels in mm
    sensorData = read_from_sensor()
    lwdData = (sensorData[1] << 8) | sensorData[2]
    rwdData = (sensorData[3] << 8) | sensorData[0]

    if not controlVariable['leftDir']:
        lwdData = 0 - lwdData
    if not controlVariable['rightDir']:
        rwdData = 0- rwdData

    dthetaRadians = (rwdData - lwdData)/wheelBase
    dxy = (lwdData + rwdData)/2

    return dxy, dthetaRadians


########################################################################
#Get Lidar data for slam
def getScansForSlam(lidar, lidarDict):
    distance_mm = []
    angles_deg = []
    for i, scan in enumerate(lidar.iter_scans()):
        for quality, angle, distance in scan:
            if quality > 0 and i != 0:
                angles_deg.append(angle)
                distance_mm.append(distance)
                roundAngle = round(angle)
                lidarDict[roundAngle] = (distance/1000)
                #print(f'Angle : {angle} | Distance {distance}')
        if i >0:
            lidar.stop()
            break
    return distance_mm, angles_deg

########################################################################
#Lidar process, will run on one core. This is for SLAM and auto

def lidar_process(event, queue, autoVariable, lidarDict, controlVariable):
```

```
    lidar = RPLidar('/dev/ttyUSB0')
    info = lidar.get_info()
    print(info)
    health = lidar.get_health()
    print(health)
    while not event.is_set():
        while(autoVariable.value):
            distances, angles = getScansForSlam(lidar, lidarDict)
            dxy, dthetaRadians = get_and_calc_odometer(controlVariable)
            lidarDict['dxy'] = dxy
            lidarDict['dthetaRadians'] = dthetaRadians
            queue.put((distances, angles, dxy, dthetaRadians))
        if (not autoVariable.value):
            stopLidar(lidar)

    #Disconnect and stop lidar
    lidarDisconnect(lidar)

# Alternative function with iterator, not tested

""" def lidar_process(event, queue, autoVariable, lidarDict,
controlVariable):
    lidar = RPLidar('/dev/ttyUSB0')
    lidar.start_motor()
    iterator = lidar.iter_scans()

    while not event.is_set():
        if autoVariable.value:
            try:
                items = next(iterator)  # Fetch the next scan
                distances = [item[2] for item in items if item[0] > 0]  #
Quality filtering
                angles = [item[1] for item in items if item[0] > 0]
                dxy, dthetaRadians =
get_and_calc_odometer(controlVariable)
                roundAngle = round(angles)
                lidarDict[roundAngle] = (distance)
                lidarDict['dxy'] = dxy
                lidarDict['dthetaRadians'] = dthetaRadians
                queue.put((distances, angles, dxy, dthetaRadians))
            except StopIteration:
                break  # Stop if no more scans

        else:
            time.sleep(0.1)  # Adjust the sleep time as needed

    lidar.stop()
    lidar.stop_motor()
    lidar.disconnect() """

########################################################################

def scanDistanceAngleManual(lidarDict, lidar):
    for i, scan in enumerate(lidar.iter_scans()):
        print('%d: Got %d measurments' % (i, len(scan)))
        for quality, angle, distance in scan:
            roundAngle = round(angle)
            lidarDict[roundAngle] = (distance/1000)
            #print(f'Angle: {roundAngle} | Distance:
{lidarDict[roundAngle]}')
        if i > 5:
            lidar.stop()
            lidar.stop_motor()
            break
    for key, value in lidarDict.items():
        print(f"Angle: {key} | Distance: {value}")
```

```
########################################################################

def scanDistanceAngleAuto(lidarDict):
    for i, scan in enumerate(lidar.iter_scans()):
        print('%d: Got %d measurments' % (i, len(scan)))
        for quality, angle, distance in scan:

            roundAngle = round(angle)
            lidarDict[roundAngle] = (distance/1000)
            #print(f'Angle: {roundAngle} | Distance:
{lidarDict[roundAngle]}')
        if i > 5:
            break
    for key, value in lidarDict.items():
        print(f"Angle: {key} | Distance: {value}")

########################################################################
```

3 Styrmodul kod

3.1 Autonom styrning kod

3.1.1 RunAuto.py

```
from app.rp_master import write_to_styr

import time
import heapq
import math

import numpy as np

from datetime import datetime

f= open('debbug_aauto.txt', 'w')

def heuristic(a, b):
    # Using Manhattan distance as heuristic
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

########################################################################

def get_neighbors(grid, node):
    neighbors = []
    print(f'In start of get_neighbors for node: {node}')
    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  # Adjacent
neighbors
        x, y = node[0] + dx, node[1] + dy
        if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] ==
0:
            neighbors.append((x, y))
    print(f'Neighbors found: {neighbors}')
    return neighbors

########################################################################

def a_star(grid, start, goal):
    print(f"{datetime.now().strftime('%H:%M:%S ')} Starting A* from
{start} to {goal}\n")
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}

    while open_set:
        current = heapq.heappop(open_set)[1]
        print(f"{datetime.now().strftime('%H:%M:%S ')} Current node in A*:
{current}\n")

        if current == goal:
            print(f'current == goal: {current == goal}')
            return reconstruct_path(came_from, current)

        for neighbor in get_neighbors(grid, current):
            print(f'Checking neighbor: {neighbor}')
            tentative_g_score = g_score[current] + 1  # Assuming each move
has a cost of 1
            if neighbor not in g_score or tentative_g_score <
g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score +
heuristic(neighbor, goal)
                print(f"tentative_g_score: {tentative_g_score}\n")
                print(f"Updated scores: g_score[{neighbor}] =
{g_score[neighbor]}, f_score[{neighbor}] = {f_score[neighbor]}")

                if neighbor not in [i[1] for i in open_set]:
```

```
                    heapq.heappush(open_set, (f_score[neighbor], 
neighbor))
                    print(f"Added {neighbor} to open_set")
        print(f"open_set: {[i[1] for i in open_set]}")
    print("No path found to the goal.")
    return None

########################################################################

def reconstruct_path(came_from, current):
    path = []
    while current in came_from:
        path.append(current)
        current = came_from[current]
    path.reverse()
    return path

########################################################################

def translate_to_wheel_commands(turnAngleRadians, distance):
    max_cell_distance = 10
    max_speed = 255
    # Calculate speed proportional to the distance to travel
    speed = int((distance / max_cell_distance) * max_speed)
    speed = max(min(speed, max_speed), 64)  # Ensure speed is within 0 to 
max_speed

    # Adjust speed for turning
    if abs(turnAngleRadians) > math.pi / 4:  # Turning more than 45 
degrees
        turnSpeed = speed // 2
    else:
        turnSpeed = speed

    f.write(f"turnAngleRadians: {turnAngleRadians}, speed: {speed}, 
turnSpeed: {turnSpeed}\n")
    print(f"turnAngleRadians: {turnAngleRadians}, speed: {speed}, 
turnSpeed: {turnSpeed}\n")
    # Determine wheel speeds and directions based on direction and turn 
angle
         # Forward 
Back
    if (-(math.pi / 4) <= turnAngleRadians <= (math.pi / 4)) and 
((3*math.pi / 4) <= turnAngleRadians <= -(3*math.pi / 4)):
        left_speed = right_speed = speed
        if ((3*math.pi / 4) <= turnAngleRadians <= -(3*math.pi / 4)): # If 
backwards
            left_direction = right_direction = 0
        else: #Else forward
            left_direction = right_direction = 1

        if turnAngleRadians > 0:  # Turning right
            left_speed = turnSpeed
        elif turnAngleRadians < 0:  # Turning left
            right_speed = turnSpeed

        f.write(f"Forward/Backwards command: left_speed: {left_speed} | 
right_speed: {right_speed} | left_direction: {left_direction} | 
right_direction: {right_direction}\n")
        print(f"turnAngleRadians: {turnAngleRadians}, speed: {speed}, 
turnSpeed: {turnSpeed}\n")
         # Right 
Left
    elif (-(3*math.pi / 4) < turnAngleRadians < -(math.pi / 4) and 
(math.pi / 4) < turnAngleRadians < (3*math.pi / 4)):
        if (math.pi / 4) < turnAngleRadians < (3*math.pi / 4):
```

```
            left_speed, right_speed = turnSpeed , speed
            left_direction, right_direction = 0, 1
        else:
            left_speed, right_speed = speed, turnSpeed
            left_direction, right_direction = 1, 0

        if turnAngleRadians != 0:
            # Adjust for diagonal or curved paths
            left_speed = right_speed = turnSpeed
        f.write(f"Right/Left command: left_speed: {left_speed} |
right_speed: {right_speed} | left_direction: {left_direction} |
right_direction: {right_direction}\n")
        print(f"Right/Left : {turnAngleRadians}, speed: {speed},
turnSpeed: {turnSpeed}\n")

    else:  # Stop or undefined direction
        f.write(f"{datetime.now().strftime('%H:%M:%S ')} Stop or undefined
direction\n")
        print(f"{datetime.now().strftime('%H:%M:%S ')} Stop or undefined
direction\n")
        return 0, 0, 0, 0

    return left_speed, right_speed, left_direction, right_direction

########################################################################

def get_direction(current, next):
    delta_x = next[0] - current[0]
    delta_y = next[1] - current[1]

    if delta_x > 0 and delta_y == 0:
        return 'east'
    elif delta_x < 0 and delta_y == 0:
        return 'west'
    elif delta_x == 0 and delta_y > 0:
        return 'north'
    elif delta_x == 0 and delta_y < 0:
        return 'south'
    elif delta_x > 0 and delta_y > 0:
        return 'northeast'
    elif delta_x > 0 and delta_y < 0:
        return 'southeast'
    elif delta_x < 0 and delta_y > 0:
        return 'northwest'
    elif delta_x < 0 and delta_y < 0:
        return 'southwest'
    else:
        return 'stop'

########################################################################
#updates current orientation
def update_orientation_in_radians(currentOrientation, turnAngleRadians):
    # Update the orientation by adding the turn angle
    # Ensure that the orientation stays within the range [0, 2*pi)
    newOrientation = (currentOrientation + turnAngleRadians) % (2 *
math.pi)
    return newOrientation

########################################################################

def calculate_turn_angle(currentOrientation, targetOrientation):
    # Calculate the minimal turn angle from current_orientation to
target_orientation
    turnAngle = targetOrientation - currentOrientation
```

```
    # Adjust to find the shortest turning path (clockwise or
counterclockwise)
    turnAngle = (turnAngle + math.pi) % (2 * math.pi) - math.pi

    return turnAngle

########################################################################
#This returns in degrees
def get_target_orientation(current, nextPosition):
    # calculates the direction the robot needs to face to head towards its
next target
    delta_x = nextPosition[0] - current[0]
    delta_y = nextPosition[1] - current[1]
    return math.atan2(delta_y, delta_x)

########################################################################
#This returns in radians and normalized
def get_direction_in_radians(current, nextPosition):
    delta_x = nextPosition[0] - current[0]
    delta_y = nextPosition[1] - current[1]

    # Calculate the angle in radians
    angle = math.atan2(delta_y, delta_x)

    # Normalize the angle between 0 and 2*pi
    angle = angle % (2 * math.pi)

    return angle

########################################################################


def navigate_path(path, currentOrientation):
    for i in range(len(path) - 1):

        currentPosition = path[i] ## Maybe xPos and yPos
        nextPosition = path[i + 1]
        f.write(f"{datetime.now().strftime('%H:%M:%S ')} Navigate path,
current position: {currentPosition}, next position: {nextPosition}\n")

        targetOrientation = get_direction_in_radians(currentPosition,
nextPosition)
        f.write(f"{datetime.now().strftime('%H:%M:%S ')}
targetOrientation: {targetOrientation}\n")

        turnAngleRadians = calculate_turn_angle(currentOrientation,
targetOrientation)
        f.write(f"{datetime.now().strftime('%H:%M:%S ')} turnAngleRadians:
{turnAngleRadians}\n")

        # Calculate direction to move
        #direction = get_target_orientation(currentPosition, nextPosition,
currentOrientation)
        distance = math.sqrt((nextPosition[0] - currentPosition[0]) ** 2 +
(nextPosition[1] - currentPosition[1]) ** 2)
        f.write(f"{datetime.now().strftime('%H:%M:%S ')} distance:
{distance}\n")

        # Translate direction to wheel speeds and directions
        leftSpeed, rightSpeed, leftDirection, rightDirection =
translate_to_wheel_commands(turnAngleRadians, distance)
        f.write(f"{datetime.now().strftime('%H:%M:%S ')} Commands sent:
Left Speed: {leftSpeed}, Right Speed: {rightSpeed}, Left Direction:
{leftDirection}, Right Direction: {rightDirection}\n")
        # Send commands to robot
```

```
        write_to_styr([leftSpeed, rightSpeed, leftDirection,
rightDirection])

        # Update current orientation based on movement
        currentOrientation =
update_orientation_in_radians(currentOrientation, turnAngleRadians)

########################################################################


def identify_frontiers(grid, explored):
    frontiers = []
    rows = len(grid)
    cols = len(grid[0]) if rows > 0 else 0

    for x in range(rows):
        for y in range(cols):
            if grid[x][y] == 1 and not explored[x][y] and
is_adjacent_to_explored(grid, x, y, explored):
                frontiers.append((y, x))# This is mega whack but get
correct results
                #f.write(f"Identified frontiers from identifiy:
{frontiers} | {(x, y)}\n")


    return frontiers

########################################################################

def is_adjacent_to_explored(grid, x, y, explored):

    for dx, dy  in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        nx, ny = x + dx, y + dy
        if (0 <= nx < len(grid)) and (0 <= ny < len(grid[0])):
            if grid[nx][ny] == 1: # and explored[nx][ny]:
                print(f"is_adjacent_to_explored: {(nx,ny)} \n")
                print('here')
                return True
    return False

########################################################################
# Maybe change to longer distance
def select_frontier(frontiers, current_position):
    # Select the nearest frontier to the current position
    furthest_frontier = None
    max_distance = float('-inf')
    #f.write(f"{datetime.now().strftime('%H:%M:%S ')} frontiers in
select_frontier: {frontiers}\n")
    for frontier in frontiers:
        distance = calculate_distance(current_position, frontier)
        if distance > max_distance:
            furthest_frontier = frontier
            max_distance = distance
          # f.write(f"{datetime.now().strftime('%H:%M:%S ')} Selected
frontier: {nearest_frontier}\n")

    return furthest_frontier

########################################################################

def calculate_distance(point1, point2):
    # Use Euclidean distance or Manhattan distance as needed
    f.write(f" point1: {point1} |point2: {point2}\n")

    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] -
point2[1])**2)
```

```
########################################################################

def return_to_start(grid, current_position, start_position):
    # Use A* algorithm to find the shortest path back to the start
position
    return_path = a_star(grid, current_position, start_position)
    return return_path

########################################################################

def update_grid_from_slam(grid, slam_data):
    # Assuming slam_data contains information about obstacles and free
spaces
    # Update the grid accordingly
    for data_point in slam_data:
        x, y, is_obstacle = data_point
        if is_valid_coordinate(grid, x, y):
            if is_obstacle:
                grid[x][y] = 0  # Mark as obstacle
            else:
                grid[x][y] = 1  # Mark as free space

def is_valid_coordinate(grid, x, y):
    return 0 <= x < len(grid) and 0 <= y < len(grid[0])

########################################################################
def convert_to_grid(gridQueue):

    threshold = 127 #or 127 without that scaling boi 0.5
    gridMap = gridQueue.get()
    gridMap  = np.array(gridMap).reshape(100,100)
    gridMap  = np.transpose(gridMap)[::-1]
    #normalizedGrid = 1 - (gridMap /255)

    binaryGrid = np.zeros_like(gridMap, dtype=int)
    gridSize = gridMap.shape[0]

    for y in range(gridSize):
        for x in range(gridSize):
            binaryGrid[y,x] = 1 if gridMap[y,x] < threshold else 0

    return binaryGrid

########################################################################

def run_exploration(grid, startPosition, xythetaQueue, gridQueue):

    explored = np.zeros((100,100),dtype=int)
    currentPosition = startPosition
    home = startPosition

    while True:
        grid = convert_to_grid(gridQueue)
        xPos, yPos, currentOrientation = xythetaQueue.get()  # Get latest
SLAM data
        frontiers = identify_frontiers(grid, explored)
        #for rows in (grid):
         #   f.write(f'{rows}\n\n')
        if not frontiers:
            break
        targetFrontier = select_frontier(frontiers,(xPos, yPos))
        pathToFrontier = a_star(grid, currentPosition, targetFrontier)
        #explore_frontier(grid, pathToFrontier[-1])
        print(targetFrontier)
        print(f"pathToFrontier: {pathToFrontier}\n")
```

```
        print(f"pathToFrontier: {pathToFrontier}\n")
        navigate_path(pathToFrontier, currentOrientation) #unsure aboutr -
1
        currentPosition = pathToFrontier # dubbel check if actually there
        # Update explored set
        explored.update(pathToFrontier)


    grid = convert_to_grid(gridQueue)
    xPos, yPos, currentOrientation = xythetaQueue.get()
    pathHome = a_star(grid, currentPosition, home)
    navigate_path(pathHome[-1], currentOrientation)
    return 0
    # Navigate back to start if needed
    # ...



def runAuto(event, autoVariable, xythetaQueue, gridQueue):
    while not event.is_set():
        if autoVariable.value:
            xPos, yPos, theta = xythetaQueue.get()
            grid = gridQueue.get()
            startPosition = (int(xPos), int(yPos))
            f.write(f"{datetime.now().strftime('%H:%M:%S ')} Auto run
started. Initial Position: {startPosition}\n")
            run_exploration(grid, startPosition, xythetaQueue, gridQueue)
            f.write("{datetime.now().strftime('%H:%M:%S ')} Auto run
ended.\n")
            f.close()
            autoVariable.value = False

    return 0
```

3.1.2 BreezySLAM exempelkod

```
from breezyslam.algorithms import RMHC_SLAM
from breezyslam.sensors import RPLidarA1 as LaserModel
from rplidar import RPLidar as Lidar
from roboviz import MapVisualizer

import numpy as np
import plotly.graph_objects as go
import math
lastMapState = None

########################################################################
# To sort out old elements from map (Not in use right now)
def calculate_Map(newMapState):
    global lastMapState
    if lastMapState is None:
        lastMapState = newMapState
        return {str((x, y)): newMapState[y, x]
                for y in range(newMapState.shape[0])
                for x in range(newMapState.shape[1])}

    diff = newMapState != lastMapState
    y_indices, x_indices = np.where(diff)
    newData = {str((x, y)): newMapState[x, y]for y, x in zip(y_indices,
x_indices)}
    lastMap = newMapState
    return newData


########################################################################
# Generate Figure of the map variables, using plotly

def create_Slam_Map(slamMap, xPos, yPos):
    fig = go.Figure()
    xPos = [xPos]
    yPos = [yPos]

    colorScale = [[1, 'black'], [0, 'white']]

    fig.add_trace(go.Heatmap(z=slamMap, colorscale='greys' ,
showscale=False))
    fig.add_trace(go.Scatter(x=xPos, y=yPos, mode = 'markers+lines',
name='Robot'))
    fig.update_layout(xaxis_title = 'X', yaxis_title='Y') # title='SLAM
VIS',
    return fig

########################################################################
# Function to conver slam data to plotly data

def mapbytes_to_plotly(mapbyte, mapSize):
    mapGrid = np.array(mapbyte).reshape(mapSize,mapSize)
    normalizedGrid = 1 - (mapGrid/255)
    normalizedGrid = np.transpose(normalizedGrid)[::-1]
    return normalizedGrid

########################################################################
# Main process to handle slam algorithm
def slam_process(event, queue, autoVariable, xythetaQueue, figQueue,
gridQueue):
    MAP_SIZE_PIXELS          = 100
    MAP_SIZE_METERS          = 10
    holeWidthMm              = 400

    # Ideally we could use all 250 or so samples that the RPLidar delivers
in one
```

```
    # scan, but on slower computers you'll get an empty map and unchanging
position
    # at that rate.
    MIN_SAMPLES    = 100

    # Create an RMHC SLAM object with a laser model and optional robot
model
    slam = RMHC_SLAM(LaserModel(), MAP_SIZE_PIXELS, MAP_SIZE_METERS,
hole_width_mm =holeWidthMm)

    # Set up a SLAM display
    #viz = MapVisualizer(MAP_SIZE_PIXELS, MAP_SIZE_METERS, 'SLAM')

    # Initialize an empty trajectory
    trajectory = []

    # Initialize empty map
    mapbytes = bytearray(MAP_SIZE_PIXELS * MAP_SIZE_PIXELS)

    # We will use these to store previous scan in case current scan is
inadequate
    previous_distances = None
    previous_angles    = None
    dxy = 0
    dtheta = 0

    while not event.is_set():
        while(autoVariable.value):

            distances, angles, dxy, dtheta = queue.get()
            print(len(distances))
            # Update SLAM with current Lidar scan and scan angles if
adequate

            if len(distances) > MIN_SAMPLES:
                slam.update(distances,  scan_angles_degrees=angles)
#pose_change=(0,0,0),
                #slam.update(distances, scan_angles_degrees=angles)
                previous_distances = distances.copy()
                previous_angles    = angles.copy()

            # If not adequate, use previous
            elif previous_distances is not None:
                #slam.update(previous_distances,
scan_angles_degrees=previous_angles)
                slam.update(previous_distances,
scan_angles_degrees=previous_angles) #pose_change=(0,0,0),

            # Get current robot position
            xPos, yPos, theta = slam.getpos()
            xPos = xPos/100
            yPos = yPos/100
            theta = math.radians(theta)

            print(f'X : {xPos} | Y : {yPos} | Theta : {theta}')
            xythetaQueue.put((xPos, yPos, theta))
            # Get current map bytes as grayscale
            slam.getmap(mapbytes)
            gridQueue.put(mapbytes)
            #Convert to numpy array
            numpyMapBytes= mapbytes_to_plotly(mapbytes, MAP_SIZE_PIXELS)

            #check if similarites for old and new scan
            #newData = calculate_Map(numpyMapBytes)

            newData = create_Slam_Map(numpyMapBytes, xPos, yPos)
```

```
            figQueue.put(newData)


            #create figure to be sent to web
            #fig = create_Slam_Map(numpyMapBytes , xPos, yPos)

            #figQueue.put(fig)


            # Display map and robot pose, exiting gracefully if user
closes it
            #if not viz.display(x/1000., y/1000., theta, mapbytes):
                #exit(0)
```

3.1.3 Frontier Exploration Algorithm exempelkod

```
import rospy
from nav_msgs.msg import OccupancyGrid
from geometry_msgs.msg import Pose, Point, Twist
from tf.transformations import euler_from_quaternion

class FrontierExploration:
    def __init__(self):
        rospy.init_node('frontier_exploration')
        self.map_sub = rospy.Subscriber('/map', OccupancyGrid,
self.map_callback)
        self.pose_sub = rospy.Subscriber('/pose', Pose,
self.pose_callback)
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist,
queue_size=10)
        self.map_data = None
        self.robot_pose = None

    def map_callback(self, msg):
        self.map_data = msg

    def pose_callback(self, msg):
        self.robot_pose = msg

    def explore(self):
        rate = rospy.Rate(10)  # 10 Hz
        while not rospy.is_shutdown():
            if self.map_data is not None and self.robot_pose is not None:
                frontiers = self.detect_frontiers()
                if frontiers:
                    target_pose = self.choose_target(frontiers)
                    self.move_to_target(target_pose)
                else:
                    rospy.loginfo("Exploration complete!")
                    break

            rate.sleep()

    def detect_frontiers(self):
        # Implement frontier detection logic
        # Analyze the map and identify unexplored frontiers
        # Return a list of frontiers

    def choose_target(self, frontiers):
        # Implement logic to choose the next frontier as the target
        # Return the target pose (Pose message)

    def move_to_target(self, target_pose):
        # Implement logic to move the robot to the target pose
        # Use the cmd_vel publisher to send velocity commands

if __name__ == '__main__':
    explorer = FrontierExploration()
    explorer.explore()
```

3.1.4 A* (A-star) exempelkod

```
class Node():
    """A node class for A* Pathfinding"""

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position


def astar(maze, start, end):
    """Returns a list of tuples as a path from the given start to the
given end in the given maze"""

    # Create start and end node
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    # Initialize both open and closed list
    open_list = []
    closed_list = []

    # Add the start node
    open_list.append(start_node)

    # Loop until you find the end
    while len(open_list) > 0:

        # Get the current node
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        # Pop current off open list, add to closed list
        open_list.pop(current_index)
        closed_list.append(current_node)

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1] # Return reversed path

        # Generate children
        children = []
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1),
(-1, 1), (1, -1), (1, 1)]: # Adjacent squares

            # Get node position
            node_position = (current_node.position[0] + new_position[0],
current_node.position[1] + new_position[1])

            # Make sure within range
```

```
            if node_position[0] > (len(maze) - 1) or node_position[0] < 0
or node_position[1] > (len(maze[len(maze)-1]) -1) or node_position[1] < 0:
                continue

            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            # Create new node
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
        for child in children:

            # Child is on the closed list
            for closed_child in closed_list:
                if child == closed_child:
                    continue

            # Create the f, g, and h values
            child.g = current_node.g + 1
            child.h = ((child.position[0] - end_node.position[0]) ** 2) +
((child.position[1] - end_node.position[1]) ** 2)
            child.f = child.g + child.h

            # Child is already in the open list
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue

            # Add the child to the open list
            open_list.append(child)


def main():

    maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    start = (0, 0)
    end = (7, 6)

    path = astar(maze, start, end)
    print(path)


if __name__ == '__main__':
    main()
```

4 Gränsnitt kod

4.1 Frontend kod

4.1.1 HTML

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

4.1.1.1 index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Robot Control Interface</title>
</head>
<link rel="shortcut icon" href="/static/favicon.ico">
<link rel="stylesheet" href="/static/css/newstyle.css">
<body>
<!--new-->

<div id="map-container">
    <div id="map">MAP</div>
</div>

<div class="flex-container">
  <div id="log-container" class="log-container">
    <div id="commandLog" class="command-log"></div>
  </div>

  <div class="control-container">
    <div id = "lwsSpeedLog" class="info-box"></div>
    <br>
    <div id = "rwsSpeedLog" class="info-box"></div>
    <br>
  <!--<div id = "dxyLog" class="info-box"></div>
    <br>
    <div id = "firSensorLog" class="info-box"></div>
    <br>
    <div id = "rirSensorLog" class="info-box"></div>-->
  </div>

  <div class="command-container">
    <button class="button" onclick="sendCommand('auto')">Auto</button>
    <button class="button" onclick="sendCommand('manual')"
>Manual</button>
    <button class="button" onclick="startScan()">Start Scan</button>
    <br>
  </div>
  <div class="control-container">
    <button class="button_dir" onmousedown="handleButtonEvent(event,
'start', 'forward')" onmouseup="handleButtonEvent(event, 'end',
'forward')" ontouchstart="handleButtonEvent(event, 'start', 'forward')"
ontouchend="handleButtonEvent(event, 'end', 'forward')">&#x2191</button>
    <div class="flex_button_dir">
      <button class="button_dir" onmousedown="handleButtonEvent(event,
'start', 'left')" onmouseup="handleButtonEvent(event, 'end', 'left')"
ontouchstart="handleButtonEvent(event, 'start', 'left')"
ontouchend="handleButtonEvent(event, 'end', 'left')">&#x2190</button>
      <button class="button_dir" onmousedown="handleButtonEvent(event,
'start', 'back')" onmouseup="handleButtonEvent(event, 'end', 'back')"
ontouchstart="handleButtonEvent(event, 'start', 'back')"
ontouchend="handleButtonEvent(event, 'end', 'back')">&#x2193</button>
      <button class="button_dir"  onmousedown="handleButtonEvent(event,
'start', 'right')" onmouseup="handleButtonEvent(event, 'end', 'right')"
ontouchstart="handleButtonEvent(event, 'start', 'right')"
ontouchend="handleButtonEvent(event, 'end', 'right')">&#x2192</button>
    </div>
  </div>
</div>

<script src="/static/js/robot.js"></script>
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

</body>
```

```
</html>
```

4.1.2 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.

4.1.2.1 newstyle.css

```
body {
    text-align: center;
    font-family: Arial, sans-serif;
    display: grid;
    grid-template-columns: 3fr 1fr;
    gap: 20px;
    padding: 20px;
}

#map-container {
    grid-column: 1;
    width: 100%; /* Full width of its grid area */
    height: calc(100vh - 40px); /* Full height of the viewport minus
padding */
    border: 2px solid #000;
    background-color: #f3f3f3;
    display: flex;
    justify-content: center;
    align-items: center;
}

#map {
    width: 100%;
    height: 100%;
    background-color: #fff; /* Assuming you want a white background for
the map */
}

.log-command-container {
    grid-column: 2;
    display: flex;
    flex-direction: column;
    height: calc(100vh - 40px); /* Full height of the viewport minus
padding */
    justify-content: space-between; /* Align children to top and bottom */
}

.command-log {
    flex-grow: 1;
    overflow-y: auto;
    max-height: 200px;
    background: black;
    color: lime;
    padding: 10px;
    white-space: pre-wrap;
    font-family: 'Courier New', monospace;
    margin-bottom: 20px;
    border:1px solid #000;
}

.control-container {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.command-container {
    display: flex;
    flex-direction: space-around;
    align-items: center;
}

.button, .button_dir {
    width: 100px;
    height: 50px;
    margin: 5px;
```

```
    font-size: 18px;
}

.info-box {
    width: 100%;
    height: 50px;
    border: 1px solid #000;
    margin: 5px;
    padding: 10px;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

4.1.3 JavaScript kod

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

4.1.3.1 Robot.js

```
// robot.js

////////////////////////////////////////////////////////////////////////////
/////////////////
// This part stores values used in various functions

// Keeps track of mode selected to avoid sending unnecessary data
var autoManual = 'manual';
// Object to track the state of control buttons
const buttonsPressed = {
  forward: false,
  back: false,
  left: false,
  right: false
};

//Logs what keys are pressed
const keysPressed = {
  ArrowUp: false,
  ArrowDown: false,
  ArrowLeft: false,
  ArrowRight: false
};


////////////////////////////////////////////////////////////////////////////
/////////////////
// The following functions handles the directions and bindings

//Maps keypress directions to correct buttonPressed
function mapKeyToDirection(key) {
  switch (key) {
      case 'ArrowUp': return 'forward';
      case 'ArrowDown': return 'back';
      case 'ArrowLeft': return 'left';
      case 'ArrowRight': return 'right';
      default: return null;
  }
}

// Function to determine the robot's direction based on buttons pressed
function determineDirection() {
  // Combine button states to determine the direction
  if (buttonsPressed.forward && buttonsPressed.left) {
    return 'forward-left';
  } else if (buttonsPressed.forward && buttonsPressed.right) {
    return 'forward-right';
  } else if (buttonsPressed.back && buttonsPressed.left) {
    return 'back-left';
  } else if (buttonsPressed.back && buttonsPressed.right) {
    return 'back-right';
  } else if (buttonsPressed.forward) {
    return 'forward';
  } else if (buttonsPressed.back) {
    return 'back';
  } else if (buttonsPressed.left) {
    return 'left';
  } else if (buttonsPressed.right) {
    return 'right';
  } else {
    return null;
  }
}

////////////////////////////////////////////////////////////////////////////
/////////////////
```

```
// Function to stop the robot
async function stopRobot() {
  try {
    const response = await fetch(`/stopDirection/${"stop"}`, { method:
'POST' });
    const data = await response.json();
    logCommand('Response: ' + JSON.stringify(data)); // Log the response
from the server in the web log
  } catch (error) {
    logCommand('Error: ' + error.message); // Log errors in the web log
  }
}

/////////////////////////////////////////////////////////////////////////
/////////////////

// Function to move the robot based on the current state
async function moveRobot() {
  const direction = determineDirection();
  if (autoManual === 'manual'){
    if (direction) {
      console.log('Move:', direction);
      logCommand('Move: ' + direction);
      try {
        const response = await fetch(`/direction/${direction}`, { method:
'POST' });
        const data = await response.json();
        logCommand('Response: ' + JSON.stringify(data)); // Log the
response from the server in the web log
      } catch (error) {
          logCommand('Error: ' + error.message); // Log errors in the web
log
      }
    } else {
      // Stop the robot if no direction is determined (i.e., no buttons
are pressed)
      stopRobot();
      logCommand('Move: ' + direction);
    }
  }else{
    logCommand('Toggle manual mode to control robot');
  }
}

/////////////////////////////////////////////////////////////////////////
/////////////////

// Update the movement state for both button and keyboard inputs
function updateMovementState(inputType, action, direction) {

  if (inputType === 'button') {
    // Handle button press
    buttonsPressed[direction] = action === 'start';
  } else if (inputType === 'keyboard') {
    // Handle keyboard press
      buttonsPressed[mapKeyToDirection(direction)] =
keysPressed[direction];
    }


  if (buttonsPressed.forward || buttonsPressed.back || buttonsPressed.left
|| buttonsPressed.right) {
    moveRobot();
  } else {
    stopRobot();
```

```
  }
}

//////////////////////////////////////////////////////////////////////////
/////////////////
// The following functions handles the input from the website,
keypressed/relaesed, touch or button

function handleButtonEvent(event, action, direction) {
  console.log('hello')
  event.preventDefault(); // Prevent default behavior like scrolling
  updateMovementState('button', action, direction);
}

// Event listener for keydown
document.addEventListener('keydown', (e) => {
  if (e.key in keysPressed && !keysPressed[e.key]) {
    keysPressed[e.key] = true;
    updateMovementState('keyboard', null, e.key);
  }
});

//Event listener for keyup
document.addEventListener('keyup', (e) => {
  if (e.key in keysPressed) {
    keysPressed[e.key] = false;
    updateMovementState('keyboard', null, e.key);
  }
});

//////////////////////////////////////////////////////////////////////////
/////////////////
//function for changhing between auto/maualy

async function sendCommand(command) {
  console.log('sending command')
  let inUse = command === autoManual;
  let toggleInterval = (command === 'auto');
  if (!inUse) {
    autoManual = command;
    if (autoManual === 'auto'){
      startSSE();
    }else if (autoManual === 'manual'){
      stopSSE();
    }
    logCommand('Toggle: ' + command);
    try {
      const response = await fetch(`/command/${command}`, { method: 'POST'
});
      const data = await response.json();
      logCommand('Response: ' + JSON.stringify(data)); // Log the response
from the server in the web log
    } catch (error) {
      logCommand('Error: ' + error.message); // Log errors in the web log
    }
  }else{
    logCommand('Toggle: ' + command + ' already set!');
  }
}

//////////////////////////////////////////////////////////////////////////
/////////////////

// Function that handles logging event on website
function logCommand(message) {
  const commandLog = document.getElementById('commandLog');
```

```
  // Create a new log entry
  const logEntry = document.createElement('div');
  logEntry.textContent = message;
  // Append the new log entry to the command log
  commandLog.appendChild(logEntry);
  // Scroll to the bottom of the log to make the latest entry visible
  commandLog.scrollTop = commandLog.scrollHeight;
}

/////////////////////////////////////////////////////////////////////////
////////////////
// Testing websocket
/*
var ws = new WebSocket("ws://localhost:8000/ws");
ws.onmessage = function(event) {
  var messages = document.getElementById('messages')
  var message = document.createElement('li')
  var content = document.createTextNode(event.data)
  message.appendChild(content)
  messages.appendChild(message)
}

function sendMessage(event) {
  var input = document.getElementById("messageText")
  ws.send(input.value)
  input.value = ''
  event.preventDefault()
}

*/
/////////////////////////////////////////////////////////////////////////
////////////////
// Testing server event
// Left wheel distance

const lwsEvtSource = new EventSource("/lws-stream");

lwsEvtSource.onmessage = function(event) {
  const lwsSpeed = JSON.parse(event.data);

  // Update html
  const lwsSpeedLog = document.getElementById('lwsSpeedLog');
  lwsSpeedLog.textContent = lwsSpeed;
};

lwsEvtSource.onerror = function(event) {
  console.error("LWS EventSource failed.");
  logCommand("LWS EventSource failed.");
  const lwsSpeedLog = document.getElementById('lwsSpeedLog');
  lwsSpeedLog.textContent = "Disconnected";
  // updateLeftSpeedInfoBox("Disconnected");

  // Optionally, attempt to reconnect after a delay
  setTimeout(() => {
    lwsEvtSource.close();
    lwsEvtSource = new EventSource("/lws-stream");
    // Re-add the onmessage handler
    lwsEvtSource.onmessage = function(event) {
    const lwsSpeed = JSON.parse(event.data);
    lwsSpeedLog.textContent = lwsSpeed;
  };
  }, 5000); // Attempt to reconnect after 5 seconds
};

/////////////////////////////////////////////////////////////////////////
////////////////
```

```
// right wheel distance
const rwsEvtSource = new EventSource("/rws-stream");

rwsEvtSource.onmessage = function(event) {
  const rwsSpeed = JSON.parse(event.data);

  // Update html
  const rwsSpeedLog = document.getElementById('rwsSpeedLog');
  rwsSpeedLog.textContent = rwsSpeed;
};

rwsEvtSource.onerror = function(event) {
  console.error("RWS EventSource failed.");
  logCommand("RWS EventSource failed.");
  const rwsSpeedLog = document.getElementById('rwsSpeedLog');
  rwsSpeedLog.textContent = "Disconnected";
  // updateLeftSpeedInfoBox("Disconnected");

  // Optionally, attempt to reconnect after a delay
  setTimeout(() => {
    rwsEvtSource.close();
    rwsEvtSource = new EventSource("/rws-stream");
    // Re-add the onmessage handler
    rwsEvtSource.onmessage = function(event) {
    const rwsSpeed = JSON.parse(event.data);
    rwsSpeedLog.textContent = rwsSpeed;
  };
  }, 5000); // Attempt to reconnect after 5 seconds
};

//////////////////////////////////////////////////////////////////////////
/////////////////
// dxy distance of wheels
/*const dxyEvtSource = new EventSource("/dxy-stream");

dxyEvtSource.onmessage = function(event) {
  const dxy = JSON.parse(event.data);

  // Update html
  const dxyLog = document.getElementById('dxyLog');
  dxyLog.textContent = dxy;
};

dxyEvtSource.onerror = function(event) {
  console.error("dxy EventSource failed.");
  logCommand("dxy EventSource failed.");
  const dxyLog = document.getElementById('dxyLog');
  dxyLog.textContent = "Disconnected";
  // updateLeftSpeedInfoBox("Disconnected");

  // Optionally, attempt to reconnect after a delay
  setTimeout(() => {
    dxyEvtSource.close();
    dxyEvtSource = new EventSource("/dxy-stream");
    // Re-add the onmessage handler
    dxyEvtSource.onmessage = function(event) {
    const dxy = JSON.parse(event.data);
    dxyLog.textContent = dxy;
  };
  }, 5000); // Attempt to reconnect after 5 seconds
};*/

//////////////////////////////////////////////////////////////////////////
/////////////////

//Function http request to Lidar scan function
```

```
async function startScan(){
  logCommand('Starting scan with Lidar');
  try {
      const response = await fetch(`/scan`, { method: 'POST' });
      const data = await response.json();
      logCommand('Response: ' + JSON.stringify(data)); // Log the response
from the server in the web log
  } catch (error) {
      logCommand('Error: ' + error.message); // Log errors in the web log
  }
}

/////////////////////////////////////////////////////////////////////////////
/////////////////
//Part down here handles server side of map plotting
/////////////////////////////////////////////////////////////////////////////
/////////////////

//Function thats be called when app starts
function initializeMapAtStart(){
  creatMapVis(currentMapState);
}

/////////////////////////////////////////////////////////////////////////////
/////////////////
//sort in new data and update currentMapState array
function applyNewData(newData){
  console.log(newData)
  for(const [key, value] of Object.entries(newData)){
    const [x,y] = key.split(',').map(Number);
    console.log('updating x: ${x}, y:${y} with value: ${value}')
    if (!isNaN(x) && !isNaN(y) && y >= 0 && y < currentMapState.length &&
x >= 0 && x < currentMapState[y].length){
      currentMapState[y][x] = value;
    }else{
      console.error('Invalid indices or value: x=${x}, y=${y},
value=${value}')
    }

  }
}

/////////////////////////////////////////////////////////////////////////////
/////////////////
//Function to inti map variables when server starts
function creatInitMapState(width, height){
  let mapState = new Array(height);
  for (let y = 0; y < height; y++){
    mapState[y] = new Array(width).fill(0);
  }
  return mapState
}

/////////////////////////////////////////////////////////////////////////////
/////////////////
//Creates the map when server starts
function creatMapVis(initialMapState){
  var plotlyData = convertMapStateToPlotly(initialMapState);
  Plotly.newPlot('map', plotlyData.data, plotlyData.layout)
}


/////////////////////////////////////////////////////////////////////////////
/////////////////
//Updates the map trough out
function updateMapVis(currentMapState){
```

```
    var plotlyData = convertMapStateToPlotly(currentMapState);
    Plotly.react('map', plotlyData.data, plotlyData.layout)
  }

  //////////////////////////////////////////////////////////////////////////
  /////////////////
  //Converts array data to plotable thingy
  function convertMapStateToPlotly(mapState){
    var data = [
      {
        z: mapState,
        type: 'heatmap',
        colorscale: 'Greys',
        showscale: false
      }
    ];

    var layout = {
      //title: 'SLAM map'
      xaxis:{
        autorange: true
      },
      yaxis: {
        autorange: true,
        scaleeanchor: 'x',
        scaleratio: 1
      }
    };
    return {data: data, layout: layout}
  }



  //////////////////////////////////////////////////////////////////////////
  /////////////////
  function startSSE(){
    eventSourceMap = new EventSource('/slam-visualization');

    eventSourceMap.onmessage = function(event){
      var data = JSON.parse(event.data);
      Plotly.newPlot('map', data.data, data.layout);
      //var newData = JSON.parse(event.data);
      //applyNewData(newData);
      //updateMapVis(currentMapState);
    };

    eventSourceMap.onerror = function(event){
      console.log('SSE failed.')
    };
  }


  function stopSSE(){
    if(eventSourceMap){
      eventSourceMap.close();
      eventSourceMap = null
    }
  }

  //////////////////////////////////////////////////////////////////////////
  /////////////////

  var eventSourceMap;
  var currentMapState = creatInitMapState(800, 800);
```

```
//////////////////////////////////////////////////////////////////////////
/////////////////

document.addEventListener('DOMContentLoaded', function(){
  initializeMapAtStart();
});

//////////////////////////////////////////////////////////////////////////
/////////////////


//old
var eventSourceMap = new EventSource('/slam-visualization');

eventSourceMap.onmessage = function(event){
    var data = JSON.parse(event.data);
    Plotly.newPlot('map', data.data, data.layout);
}
```

4.2 Backend kod

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

4.2.1 main.py

```
from app.lidar import scanDistanceAngleManual, lidarStart, stopLidar,
lidar_process, lidarDisconnect
from app.rp_master import read_from_sensor, write_to_styr
#from app.manAutoMode import runAuto
from app.runAuto import runAuto

import RPi.GPIO as GPIO
import app.rpslam
from app.rpslam import slam_process, mapbytes_to_plotly

import multiprocessing
from multiprocessing import Process, Event, Manager, Queue

from rplidar import RPLidar

import plotly.graph_objects as go
from plotly.io import to_json

from fastapi import FastAPI
from fastapi import Request
from fastapi import Response
from fastapi.staticfiles import StaticFiles
from fastapi.responses import StreamingResponse
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates



from typing import Generator
import asyncio
import json
import time



app = FastAPI()

#//////////////////////////////////////////////////////////////////////////
//////////////////
#lidar = RPLidar('/dev/ttyUSB0')

#For multiprocess
#Declare core variables
autoProcess = None
lidarProcess = None
slamProcess = None
#//////////////////////////////////////////////////////////////////////////
//////////////////
# Declare and initilize Queue variables

# sensorQueue contains lidar data (angle and distance), odometer data
(average distance and angle)
sensorQueue = Queue()
#xythetaQueue contains (x,y) coordinates of robots current position and
angle
xythetaQueue = Queue()
#figQueue contans figure to be plotted on website
figQueue = Queue()

gridQueue = Queue()

#
//////////////////////////////////////////////////////////////////////////
/////////////////
# Declare manager variable(s)
manager = Manager()
```

```
# autoVariable contains boolean of control state (True/False)
autoVariable = manager.Value('b', False)
# lidarDict (also) contains angle and corresponding distance at each angle
(for eazy access)
lidarDict = manager.dict({angle: None for angle in range(361)})
lidarDict['dxy'] = 0
lidarDict['dthetaRadians'] = 0
# controlVariable contains control variables (left and right wheel speed
and direction for each wheel pair)
controlVariable = manager.dict({
    'leftWheelDistance': 0,
    'rightWheelDistance': 0,
    'leftDir': 0,
    'rightDir':0
})

def switch_Switch(channel):
    autoVariable.value = not autoVariable.value
    print(f"Toggle autoVariable: {autoVariable.value}")


# Set GPIO mode to BCM
GPIO.setmode(GPIO.BCM)
# Set reset pin as output
GPIO.setup(27, GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(27, GPIO.BOTH, callback=switch_Switch, bouncetime=
1000)


#
//////////////////////////////////////////////////////////////////////////
/////////////////
# Event to handle shutdown of app, gracefully close different core
proccesses
shutdown_event = Event()

#/////////////////////////////////////////////////////////////////////////
//////////////////
#Global variables
# sensor data variables (Mostly used for debugging)
lwdData = 0
rwdData = 0

#/////////////////////////////////////////////////////////////////////////
//////////////////
# Dont really know how what the following 5-10 lines does, something about
initiating the web app
# Includes template folder and static folder to fastAPI

templates = Jinja2Templates(directory="templates")
app.mount("/static", StaticFiles(directory="static"), name="static")

#/////////////////////////////////////////////////////////////////////////
//////////////////


#/////////////////////////////////////////////////////////////////////////
//////////////////

@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

#
//////////////////////////////////////////////////////////////////////////
////////////////
```

```
#On startup of server

@app.on_event("startup")
async def on_startup():
    global autoProcess, lidarProcess, slamProcess
    #Check if process is already running, saftey step
    if lidarProcess and lidarProcess.is_alive():
        print("status: Lidar process is already running.")
    else:
        lidarProcess = multiprocessing.Process(target=lidar_process, args=
(shutdown_event, sensorQueue, autoVariable, lidarDict, controlVariable))
        lidarProcess.start()
        print("Status Lidar process started and sensor task")

    if slamProcess and slamProcess.is_alive():
        print("status: SLAM process is already running.")
    else:
        slamProcess = multiprocessing.Process(target=slam_process, args=
(shutdown_event, sensorQueue, autoVariable, xythetaQueue, figQueue,
gridQueue))
        slamProcess.start()
        print("Status SLAM process started and sensor task")

    if autoProcess and autoProcess.is_alive():
        print("status: Auto process is already running.")
        on_shutdown()
    else:
        autoProcess = multiprocessing.Process(target=runAuto,
args=(shutdown_event, autoVariable, xythetaQueue, gridQueue))
        autoProcess.start()
        print("Status Auto process started and sensor task")

    #start server event task
    #asyncio.create_task(periodic_task())

#/////////////////////////////////////////////////////////////////////////
//////////////////
#Shutting down server

@app.on_event("shutdown")
async def on_shutdown():
    global shutdown_event
    shutdown_event.set()
    GPIO.cleanup()
    #lidarDisconnect(lidar)
    # Check if processes are running, if they are, close them
    if autoProcess:
        autoProcess.join()
        autoProcess = None
        print("status Auto process stopped.")

    if slamProcess is not None:
        slamProcess.join()
        slamProcess = None
        print("status SLAM process stopped.")

    if lidarProcess is not None:
        lidarProcess.join()
        lidarProcess = None
        print("status Lidar process stopped.")


#/////////////////////////////////////////////////////////////////////////
//////////////////
# SSE for map
```

```
@app.get('/slam-visualization')
```

```
async def get_slam_vis():
    async def event_map_gen():
        # Run aslong as autoVariable is set (AutoMode)
        while autoVariable.value:
            fig = figQueue.get()
            jsonMapData = to_json(fig, validate = False)

            yield f'data: {jsonMapData}\n\n'
            await asyncio.sleep(2) #call every sencond

    # Check if Auto,if return map to website, else do something?
    if autoVariable.value:
        return StreamingResponse(event_map_gen(), media_type="text/event-
stream")
    else:
        return Response(status_code=204)

#
///////////////////////////////////////////////////////////////////////////
/////////////////
# Updates sensor data
async def periodic_task():
    global lwdData, rwdData
    while True:
        sensorData = read_from_sensor()

        lwdData = (sensorData[1] << 8) | sensorData[2]
        rwdData = (sensorData[3] << 8) | sensorData[0]

        await asyncio.sleep(1)  # Wait for 0.2 second


#//////////////////////////////////////////////////////////////////////////
//////////////////
# Server event

# Left wheel speed
@app.get("/lws-stream")
async def lws_stream(request: Request):
    async def event_stream():
        while True:
            #lwsData = 1  # Replace this with the actual function call to
get left wheel speed
            yield f"data: {json.dumps(lwdData)}\n\n"
            await asyncio.sleep(1)

    return StreamingResponse(event_stream(), media_type="text/event-
stream")

#//////////////////////////////////////////////////////////////////////////
//////////////////

# Right wheel speed
@app.get("/rws-stream")
async def rws_stream(request: Request):
    async def event_stream():
        while True:
            #rwsData = 1  # Replace this with the actual function call to
get left wheel speed
            yield f"data: {json.dumps(rwdData)}\n\n"
            await asyncio.sleep(1)

    return StreamingResponse(event_stream(), media_type="text/event-
stream")
```

```
#//////////////////////////////////////////////////////////////////////////
//////////////////
# dxy distance stream
#@app.get("/dxy-stream")
#async def dxy_stream(request: Request):
 #   async def event_stream():
 #       while True:
   #         #rwsData = 1  # Replace this with the actual function call to
get left wheel speed
 #           yield f"data: {json.dumps(lidarDict['dxy'])}\n\n"
    #        await asyncio.sleep(1)

   # return StreamingResponse(event_stream(), media_type="text/event-
stream")

#//////////////////////////////////////////////////////////////////////////
//////////////////

# This handles the API request or what it now is to toggle between
manual/auto mode
@app.post("/command/{action}")
async def command(action: str):
    autoVariable.value = (action == 'auto')
    #autoVariable.put((action == 'auto'))
    return {"status": "Command received: " + action}

#//////////////////////////////////////////////////////////////////////////
//////////////////
# Handles http/api request to stop motors on robot

@app.post("/stopDirection/{stop}")
async def stopDirection(stop: str):
    if stop == "stop":
        write_to_styr([0,0,0,0])
        return {"status": "Command received: " + stop}

#//////////////////////////////////////////////////////////////////////////
//////////////////

# Handles direction http/api request to robot
@app.post("/direction/{direction}")
async def sendDirection(direction: str):
    if direction == 'forward':
        write_to_styr([255,255,1,1])
        return {"status": "Direction received: " + direction}
    elif direction == 'back':
        write_to_styr([128,128,0,0])
        return {"statuss": "Direction received: " + direction}
    elif direction == 'right':
        write_to_styr([254,254,1,0])
        return {"status": "Direction received: " + direction}
    elif direction == 'left':
        write_to_styr([254,254,0,1])
        return {"status": "Direction received: " + direction}
    elif direction == 'forward-right':
        write_to_styr([128,255,1,1])
        return {"status": "Direction received: " + direction}
    elif direction == 'forward-left':
        write_to_styr([255,128,1,1])
        return {"status": "Direction received: " + direction}
    elif direction == 'back-left':
        write_to_styr([255,128,0,0])
        return {"status": "Direction received: " + direction}
    elif direction == 'back-right':
        write_to_styr([128,255,0,0])
        return {"status": "Direction received: " + direction}
```

```
#/////////////////////////////////////////////////////////////////////////
//////////////////

# Handles scan command from web
@app.post("/scan")
async def scan():
    scanDistanceAngleManual(lidarDict)
    return {"status": "Starting scan" }

#/////////////////////////////////////////////////////////////////////////
//////////////////
```