```c
/*
 * debug.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * debug.c
 *
 * Created: 11/1/2016 3:59:37 PM
 *  Author: patsl736
 */

#include "config.h"
#include "debug.h"

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <assert.h>
#include <stdbool.h>

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

int USARTPutChar(char data, FILE *stream);

static FILE uart_stdout = FDEV_SETUP_STREAM(USARTPutChar, NULL, _FDEV_SETUP_WRITE);

bool uart_initialized = false;
char data_to_write[1024];
unsigned char write_data_index = 0;
unsigned char read_data_index = 0;
unsigned int data_available = 0;

void initialize_uart() {
  // Set baud rate
  UBRR0L = BAUD_PRESCALE;
  UBRR0H = (BAUD_PRESCALE>>8);

  // Enable RX and TX for USART
  UCSR0B=(1<<RXEN0)|(1<<TXEN0);

  // Redirect stdout to UART
  stdout = &uart_stdout;

  // Keep track of state to detect errors early
  uart_initialized = true;
}

void USARTWriteChar(char data)
{
  data_to_write[write_data_index] = data;
  ++write_data_index;
  ++data_available;
  UCSR0B |= (1<<UDRIE0);
}

int USARTPutChar(char data, FILE *stream)
{
  // Fail hard if UART is not initialized
  assert(uart_initialized);

  // Include carriage return to start at beginning of line
  if (data == '\n')
  {
    USARTWriteChar('\r');
  }

  USARTWriteChar(data);
  return 0;
}

ISR(USART0_UDRE_vect) {
  if (data_available > 0) {
    char data = data_to_write[read_data_index];
    ++read_data_index;
    --data_available;

    UDR0=data;
    if (data_available == 0) {
      read_data_index = 0;
      write_data_index = 0;
      UCSR0B &= ~(1<<UDRIE0);
    }
  } else {
    USARTWriteChar('-');
  }
}
```

```c
/*
 * event.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * event.c
 *
 * Created: 11/7/2016 1:59:17 PM
 *  Author: antda685
 */

#include "event.h";
void_func sensor_data_request;
sensor_data_func sensor_data_returned;

motor_speed_func motor_speed_received;
left_motor_speed_func left_motor_speed_received;
right_motor_speed_func right_motor_speed_received;

//-----------------SENSORENHET-------------------------
void listen_for_sensor_data_request(void_func vf) {
  sensor_data_request = vf;
}

void listen_for_sensor_data_returned(sensor_data_func sdf) {
  sensor_data_returned = sdf;
}


void notify_sensor_data_request() {
  sensor_data_request();
}

void notify_sensor_data_returned(struct sensor_data* sd) {
  sensor_data_returned(sd);
}


//-----------------STYRENHET-------------------------
void listen_for_motor_speed_received(motor_speed_func msf) {
  motor_speed_received = msf;
}

void listen_for_left_motor_speed_received(left_motor_speed_func lmsf) {
  left_motor_speed_received = lmsf;
}

void listen_for_right_motor_speed_received(right_motor_speed_func rmsf) {
  right_motor_speed_received = rmsf;
}


void notify_motor_speed_received(struct motor_speed* ms) {
  motor_speed_received(ms);
}

void notify_left_motor_speed_received(unsigned char speed) {
  left_motor_speed_received(speed);
}

```

```
void notify_right_motor_speed_received(unsigned char speed) {
  right_motor_speed_received(speed);
}
```

```c
/*
 * i2cslave.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * i2cslave.c
 *
 * Created: 11/2/2016 3:08:08 PM
 *  Author: antda685
 */
#include "debug.h"
#include "i2cslave.h"
#include "queue.h"
#include "packet.h"

#include <string.h>
#include <avr/io.h>
#include <compat/twi.h>
#include <avr/interrupt.h>


#define MAX_DATA_SIZE 255
#define DATA_CMD_LEN 0
#define DATA_CMD_BYTE 1

unsigned char data_size;
unsigned char data_index;
unsigned char data_buffer[MAX_DATA_SIZE];

unsigned char available_data_size;
unsigned char available_data_index;
unsigned char available_data_buffer[MAX_DATA_SIZE];

unsigned int dts_index;
struct queue* data_to_send;
struct queue* data_recieved;

#define I2C_STATE_UNINIT 0
#define I2C_STATE_WAITING_FOR_ADDR 1
#define I2C_STATE_WAITING_FOR_DATA 2
#define I2C_STATE_READING_DATA 3
unsigned char addr;
unsigned char data;
ISR(TWI_vect) {
  static unsigned char i2c_state = I2C_STATE_UNINIT;
  unsigned char twi_status;
  cli();
  twi_status = TWSR & 0xF8;
  switch (twi_status) {
  case (TW_SR_SLA_ACK) : //SLA+R received, ACK returned
    i2c_state = I2C_STATE_WAITING_FOR_ADDR;
    TWCR |= (1<<TWINT); //Reset TWINT flag
    break;

  case (TW_SR_DATA_ACK) : //Data received, ACK returned
    if(i2c_state == I2C_STATE_WAITING_FOR_ADDR) {
      addr = TWDR; //Saving address
      i2c_state = I2C_STATE_WAITING_FOR_DATA;
    } else {
      switch (addr) {
        case DATA_CMD_LEN:
          data_size = TWDR;
          data_index = 0;
          break;
        case DATA_CMD_BYTE:
          data_buffer[data_index] = TWDR;
          ++data_index;
          break;
      }

      i2c_state = I2C_STATE_READING_DATA;
    }
    TWCR |= (1<<TWINT); //Reset TWINT flag
    break;

  case (TW_SR_STOP) : //STOP or START condition received while selected
    if(i2c_state == I2C_STATE_READING_DATA) {
      //Eventually save data somewhere
      i2c_state = I2C_STATE_WAITING_FOR_ADDR;
      if(data_index == data_size) {
        data_index = 0;
        struct packet* p = malloc(sizeof(struct packet));
        p->size = data_size;
        memcpy(p->data, data_buffer, data_size);
        queue_push(data_recieved, p);
      }
    }
    TWCR |= (1<<TWINT); //Reset TWINT
    break;

  case (TW_ST_DATA_ACK) : //Data transmitted, ACK received
  case (TW_ST_SLA_ACK) : //SLA+R received, ACK returned
    if(i2c_state == I2C_STATE_WAITING_FOR_DATA) {
      switch (addr) {
        case DATA_CMD_LEN:
          if (queue_empty(data_to_send)) {
            TWDR = 0;
          } else {
            dts_index = 0;
            struct packet* p = queue_front(data_to_send);
            TWDR = p->size;
          }
          break;
        case DATA_CMD_BYTE:
          if (queue_empty(data_to_send)) {
            TWDR = 0;
          }
          else {
            struct packet* p = queue_front(data_to_send);
            TWDR = p->data[dts_index];
            ++dts_index;
            if(dts_index == p->size) {
              free(p);
              queue_pop(data_to_send);
            }
          }
          break;
      }

      i2c_state = I2C_STATE_WAITING_FOR_ADDR;
    }
    TWCR |= (1<<TWINT); //Reset TWINT
    break;
  case (TW_ST_DATA_NACK) : //Data received, NACK returned
  case (TW_ST_LAST_DATA) : //last data byte transmitted, ACK received
  case (TW_BUS_ERROR) : //Illegal start or stop condition
  default:
    TWCR |= (1 << TWINT); //Reset TWINT
    i2c_state = I2C_STATE_WAITING_FOR_ADDR;
  }
  sei();
}

void send_data(struct packet* p) {
  queue_push(data_to_send, p);
}

struct packet* get_received_data() {
  if(queue_empty(data_recieved)) {
    return NULL;
  } else {
    struct packet* p = queue_front(data_recieved);
    queue_pop(data_recieved);
    return p;
  }
}

void initialize_i2c(unsigned char address)
{
  data_to_send = queue_create();
  data_recieved = queue_create();
  //Initializing i2cslave
  TWAR = (address<<1) & 0xFE; //Sets slavei2caddress and ignore general
  TWDR = 0x00; //Initial data is set to 0

  //Starts listening on i2c
  //Reset TW-Interrupt, Enable TW-ACK, TW-Enabled, TW-Interrupt Enable
  TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN) | (1<<TWIE);
  sei();
}
```

```c
/*
 * indexed_packet.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * packet_reader.c
 *
 * Created: 11/7/2016 1:20:59 PM
 *  Author: antda685
 */
#include "indexed_packet.h"


unsigned char read_byte(struct indexed_packet* p) {

  unsigned char byte = p->p->data[p->index];
  ++p->index;
  return byte;
}

void write_byte(struct indexed_packet* p, unsigned char byte) {
  p->p->data[p->index] = byte;
  ++p->index;
  ++p->p->size;
}
```

```c
/*
 * main.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * main.c
 *
 * Created: 11/7/2016 11:43:49 AM
 *  Author: antda685
 */

#include "main.h"
#include "packet.h"
int run_program(loopHandler handler)
{
  // TODO: Init

  for (;;)
  {
    struct packet* p;
    while (p = get_received_data()) {
      parse_and_execute(p);
    }

    handler();
  }
}
```

```c
/*
 * outbound.c
 *
 * Version 1.0
 * Senast modifierad 2016-12-12
 *
 * Matilda Dahlström
 * Patrik Sletmo
 */

/*
 * outbound.c
 *
 * Created: 11/7/2016 2:13:05 PM
 *  Author: antda685
 */
#include "outbound.h"
#include "indexed_packet.h"
#include "i2cslave.h"

void initalize_packet(struct indexed_packet* ip, unsigned char packet_id) {
  struct packet* p = malloc(sizeof(struct packet));
  ip->p = p;
  ip->index = 0;
  p->size=0;
  write_byte(ip, packet_id);
}

void request_sensor_data() {
  struct indexed_packet ip;
  initalize_packet(&ip, CMD_REQUEST_SENSOR_DATA);
  send_data(ip.p);
}

void return_sensor_data(struct sensor_data* sd) {
  struct indexed_packet ip;
  initalize_packet(&ip, CMD_RETURN_SENSOR_DATA);
  write_byte(&ip, sd->ir_left_mm >> 8);
  write_byte(&ip, sd->ir_left_mm & 0xFF);
  write_byte(&ip, sd->ir_right_mm >> 8);
  write_byte(&ip, sd->ir_right_mm & 0xFF);
  write_byte(&ip, sd->ir_right_back_mm >> 8);
  write_byte(&ip, sd->ir_right_back_mm & 0xFF);
  write_byte(&ip, sd->ir_left_back_mm >> 8);
  write_byte(&ip, sd->ir_left_back_mm & 0xFF);
  send_data(ip.p);
}
```

```c
/*
 * packet_parser.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-24
 *
 * Matilda Dahlström
 * Patrik Sletmo
 */

/*
 * packet_parser.c
 *
 * Created: 11/7/2016 1:32:55 PM
 *  Author: antda685
 */

#include "indexed_packet.h"
#include "protocol.h"

void parse_sensor_data_packet(struct indexed_packet* ip) {
  struct sensor_data* sd = malloc(sizeof(struct sensor_data));
  sd->ir_left_mm = (read_byte(ip) << 8) & read_byte(ip);
  sd->ir_right_mm = (read_byte(ip) << 8) & read_byte(ip);
  sd->ir_right_back_mm = (read_byte(ip) << 8) & read_byte(ip);
  sd->ir_left_back_mm = (read_byte(ip) << 8) & read_byte(ip);

  notify_sensor_data_returned(sd);
  free(sd);
}

void parse_motor_data_packet(struct indexed_packet* ip) {
  struct motor_speed* ms = malloc((sizeof(struct motor_speed)));
  ms->left_speed = read_byte(ip);
  ms->right_speed = read_byte(ip);

  notify_motor_speed_received(ms);
  free(ms);
}

void parse_and_execute(struct packet* p) {
  struct indexed_packet ip;
  ip.p = p;
  ip.index = 0;
  unsigned char cmd_id = read_byte(&ip);

  switch (cmd_id) {
    case (CMD_REQUEST_SENSOR_DATA):
      notify_sensor_data_request();
      break;
    case (CMD_RETURN_SENSOR_DATA):
      parse_sensor_data_packet(&ip);
      break;
    case (CMD_PING):
      break;
    case (CMD_PONG):
      break;
    case (CMD_SET_MOTOR_SPEED):
      parse_motor_data_packet(&ip);
      break;
    case (CMD_SET_LEFT_MOTOR_SPEED):
      notify_left_motor_speed_received(read_byte(&ip));
      break;
    case (CMD_SET_RIGHT_MOTOR_SPEED):
      notify_right_motor_speed_received(read_byte(&ip));
      break;
    default:
      break;
  }

  free(p);
}
```

```c
/*
 * queue.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * queue.c
 *
 * Created: 11/4/2016 11:30:03 AM
 *  Author: antda685
 */

#include "queue.h"
#include <assert.h>

struct queue* queue_create() {
  struct queue* q = malloc(sizeof(struct queue));
  q->next = NULL;
  q->data = NULL;

  return q;
}

void queue_free(struct queue* q) {
  struct queue* current = q;
  while (current != NULL) {
    struct queue* next = current->next;
    free(current);
    current = next;
  }
}

void* queue_front(struct queue* q) {
  //assert(q->data == NULL);

  struct queue* front = q->next;
  if (front != NULL) {
    return front->data;
  } else {
    return NULL;
  }
}

void queue_pop(struct queue* q) {
  //assert(q->data == NULL);
  //assert(!queue_empty(q));

  struct queue* old = q->next;
  q->next = q->next->next;

  if (old != NULL) {
    old->next = NULL;
    queue_free(old);
  }
}

void queue_push(struct queue* q, void* data) {
  //assert(q->data == NULL);

  struct queue* tail = q;
  while (tail->next != NULL) {
    tail = tail->next;
  }

  struct queue* next = queue_create();
  next->data = data;
  tail->next = next;

}

bool queue_empty(struct queue* q) {
  //assert(q->data == NULL);

  return q->next == NULL;
}
```

```c
/*
 * common.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * common.c
 *
 * Created: 11/1/2016 3:58:23 PM
 *  Author: patsl736
 */


#include <avr/io.h>
#include "debug.h"

void initialize_PWM();
void set_robot_speed(int speed);

int main(void)
{

  initialize_uart();
  initialize_PWM();
  set_robot_speed(2);

  int counter = 0;
  while (1) {


    counter = TCNT1;
    if(counter < 0) {
      printf("counter negative");
    }
    else{
      printf("Counter value: %d\n", TCNT1);
    }
  }
}

void set_robot_speed(int speed) {
  OCR1A = speed;  // Set speed left wheels
  OCR1B = speed;  // Set speed right wheels

}

void initialize_PWM() {

  DDRA = 0xFF;  // Set Data Direction on PortA
  DDRD = 0xFF;  // Set Data Direction on PortD
  TCNT1 = 0;    // Reset Timer1 counter

  TCCR1A = 0;   // Clear Timer1 settings
  TCCR1B = 0;   // Clear Timer1 settings

  PORTA = 0xF1; // Set direction of wheels (Pin 40 for left wheels, pin 38 for right wheels)

  TCCR1B|= (1<<WGM12)|(1<<CS12)|(1<<CS10)|(1<<WGM13); // Prescaler 1024, Timer1 settings
  TCCR1A|= (1<<COM1A1)|(1<<WGM11)|(1<<COM1B1);    // Timer1 settings
  ICR1 = 16;                          // Set TOP

}
```

```c
/*
 * debug.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * debug.c
 *
 * Created: 11/1/2016 3:59:37 PM
 *  Author: patsl736
 */

#include "config.h"
#include "debug.h"

#include <avr/io.h>
#include <inttypes.h>
#include <assert.h>
#include <stdbool.h>

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void USARTWriteChar(char data);
int USARTPutChar(char data, FILE *stream);

static FILE uart_stdout = FDEV_SETUP_STREAM(USARTPutChar, NULL,
    _FDEV_SETUP_WRITE);

bool uart_initialized = false;


void initialize_uart() {
  // Set baud rate
  UBRR0L = BAUD_PRESCALE;
  UBRR0H = (BAUD_PRESCALE>>8);

  // Enable RX and TX for USART
  UCSR0B=(1<<RXEN0)|(1<<TXEN0);

  // Redirect stdout to UART
  stdout = &uart_stdout;

  // Keep track of state to detect errors early
  uart_initialized = true;
}

void USARTWriteChar(char data)
{
  while(!(UCSR0A & (1<<UDRE0)))
  {
    // Busy wait until we can send data
  }

  UDR0=data;
}


int USARTPutChar(char data, FILE *stream)
{
  // Fail hard if UART is not initialized
  assert(uart_initialized);

  // Include carriage return to start at beginning of line
  if (data == ’\n’)
  {
    USARTWriteChar(’\r’);
  }

  USARTWriteChar(data);
  return 0;
}
```

```
/*
 * debug.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * debug.c
 *
 * Created: 11/1/2016 3:59:37 PM
 *  Author: patsl736
 */

#include "config.h"
#include "debug.h"

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <assert.h>
#include <stdbool.h>

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

int USARTPutChar(char data, FILE *stream);

static FILE uart_stdout = FDEV_SETUP_STREAM(USARTPutChar, NULL,
    _FDEV_SETUP_WRITE);

bool uart_initialized = false;
char data_to_write[1024];
unsigned char write_data_index = 0;
unsigned char read_data_index = 0;
unsigned int data_available = 0;

void initialize_uart() {
  // Set baud rate
  UBRR0L = BAUD_PRESCALE;
  UBRR0H = (BAUD_PRESCALE>>8);

  // Enable RX and TX for USART
  UCSR0B=(1<<RXEN0)|(1<<TXEN0);

  // Redirect stdout to UART
  stdout = &uart_stdout;

  // Keep track of state to detect errors early
  uart_initialized = true;
}

void USARTWriteChar(char data)
{
  data_to_write[write_data_index] = data;
  ++write_data_index;
  ++data_available;
  UCSR0B |= (1<<UDRIE0);
}

int USARTPutChar(char data, FILE *stream)
{
  // Fail hard if UART is not initialized
  assert(uart_initialized);

  // Include carriage return to start at beginning of line
  if (data == ’\n’)
  {
    USARTWriteChar(’\r’);
  }

  USARTWriteChar(data);
  return 0;
}

ISR(USART0_UDRE_vect) {
  if (data_available > 0) {
    char data = data_to_write[read_data_index];
    ++read_data_index;
    --data_available;

    UDR0=data;
    if (data_available == 0) {
      read_data_index = 0;
      write_data_index = 0;
      UCSR0B &= ~(1<<UDRIE0);
    }
  } else {
    USARTWriteChar(’-’);
  }
}
```

```
/*
 * queue.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * queue.c
 *
 * Created: 11/4/2016 11:30:03 AM
 *  Author: antda685
 */

#include "queue.h"
#include <assert.h>

struct queue* queue_create() {
  struct queue* q = malloc(sizeof(struct queue));
  q->next = NULL;
  q->data = NULL;

  return q;
}

void queue_free(struct queue* q) {
  struct queue* current = q;
  while (current != NULL) {
    struct queue* next = current->next;
    free(current);
    current = next;
  }
}

void* queue_front(struct queue* q) {
  //assert(q->data == NULL);

  struct queue* front = q->next;
  if (front != NULL) {
    return front->data;
  } else {
    return NULL;
  }
}

void queue_pop(struct queue* q) {
  //assert(q->data == NULL);
  //assert(!queue_empty(q));

  struct queue* old = q->next;
  q->next = q->next->next;

  if (old != NULL) {
    old->next = NULL;
    queue_free(old);
  }
}

void queue_push(struct queue* q, void* data) {
  //assert(q->data == NULL);

  struct queue* tail = q;
  while (tail->next != NULL) {
    tail = tail->next;
  }

  struct queue* next = queue_create();
  next->data = data;
  tail->next = next;

}

bool queue_empty(struct queue* q) {
  //assert(q->data == NULL);

  return q->next == NULL;
}
```

```
/*
 * i2cslave.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * i2cslave.c
 *
 * Created: 11/2/2016 3:08:08 PM
 *  Author: antda685
 */
#include "common/debug.h"
#include "i2cslave.h"
#include "common/queue.h"
#include "common/packet.h"

#include <string.h>
#include <avr/io.h>
#include <compat/twi.h>
#include <avr/interrupt.h>


void i2c_slave_action(unsigned char read_write_action) {
  //read = 0, write = 1. Derived from buss
  unsigned char test_data = 2;
  unsigned char test_recieve;
  if(read_write_action) {
    DDRD = test_data; //Write data to DDRD
  } else {
    test_recieve = DDRD; //Read data from DDRD
  }
}

#define MAX_DATA_SIZE 255
#define DATA_CMD_LEN 0
#define DATA_CMD_BYTE 1

unsigned char data_size;
unsigned char data_index;
unsigned char data_buffer[MAX_DATA_SIZE];

unsigned char available_data_size;
unsigned char available_data_index;
unsigned char available_data_buffer[MAX_DATA_SIZE];

unsigned int dts_index;
struct queue* data_to_send;
struct queue* data_recieved;

#define I2C_STATE_UNINIT 0
#define I2C_STATE_WAITING_FOR_ADDR 1
#define I2C_STATE_WAITING_FOR_DATA 2
#define I2C_STATE_READING_DATA 3
unsigned char addr;
unsigned char data;
ISR(TWI_vect) {
  static unsigned char i2c_state = I2C_STATE_UNINIT;
  unsigned char twi_status;
  cli();
  twi_status = TWSR & 0xF8;
  switch (twi_status) {
  case (TW_SR_SLA_ACK) : //SLA+R received, ACK returned
    i2c_state = I2C_STATE_WAITING_FOR_ADDR;
    TWCR |= (1<<TWINT); //Reset TWINT flag
    break;

  case (TW_SR_DATA_ACK) : //Data received, ACK returned
    if(i2c_state == I2C_STATE_WAITING_FOR_ADDR) {
      addr = TWDR; //Saving address
      i2c_state = I2C_STATE_WAITING_FOR_DATA;
    } else {
      switch (addr) {
        case DATA_CMD_LEN:
          data_size = TWDR;
          data_index = 0;
          break;
        case DATA_CMD_BYTE:
          data_buffer[data_index] = TWDR;
          ++data_index;
          break;
      }

      i2c_state = I2C_STATE_READING_DATA;
    }
    TWCR |= (1<<TWINT); //Reset TWINT flag
    break;

  case (TW_SR_STOP) : //STOP or START condition received while selected
    if(i2c_state == I2C_STATE_READING_DATA) {
      //Eventually save data somewhere
      i2c_state = I2C_STATE_WAITING_FOR_ADDR;
      if(data_index == data_size) {
        data_index = 0;
        struct packet* p = malloc(sizeof(struct packet));
        p->size = data_size;
        memcpy(p->data, data_buffer, data_size);
        queue_push(data_recieved, p);
      }
    }
    TWCR |= (1<<TWINT); //Reset TWINT
    break;

  case (TW_ST_DATA_ACK) : //Data transmitted, ACK received
  case (TW_ST_SLA_ACK) : //SLA+R received, ACK returned
    if(i2c_state == I2C_STATE_WAITING_FOR_DATA) {
      switch (addr) {
        case DATA_CMD_LEN:
          if (queue_empty(data_to_send)) {
            TWDR = 0;
          } else {
            dts_index = 0;
            struct packet* p = queue_front(data_to_send);
            TWDR = p->size;
          }
          break;
        case DATA_CMD_BYTE:
          if (queue_empty(data_to_send)) {
            TWDR = 0;
          }
          else {
            struct packet* p = queue_front(data_to_send);
            TWDR = p->data[dts_index];
            ++dts_index;
            if(dts_index == p->size) {
              free(p);
```

```
       queue_pop(data_to_send);
      }
     }
     break;
   }

   i2c_state = I2C_STATE_WAITING_FOR_ADDR;
  }
  TWCR |= (1<<TWINT); //Reset TWINT
  break;
 case (TW_ST_DATA_NACK) : //Data received, NACK returned
 case (TW_ST_LAST_DATA) : //last data byte transmitted, ACK received
 case (TW_BUS_ERROR) : //Illegal start or stop condition
 default:
  TWCR |= (1 << TWINT); //Reset TWINT
  i2c_state = I2C_STATE_WAITING_FOR_ADDR;
 }
 sei();
}

void send_data(struct packet* p) {
 queue_push(data_to_send, p);
}

struct packet* get_received_data() {
 if(queue_empty(data_recieved)) {
  return NULL;
 } else {
  struct packet* p = queue_front(data_recieved);
  queue_pop(data_recieved);
  return p;
 }
}
int main(void)
{
 data_to_send = queue_create();
 data_recieved = queue_create();
 initialize_uart();
 printf("Boooooooted\n");
 //Initializing i2cslave
 TWAR = (SLAVE_ADDRESS<<1) & 0xFE; //Sets slavei2caddress and ignore general
 TWDR = 0x00; //Initial data is set to 0

 //Starts listening on i2c
 //Reset TW-Interrupt, Enable TW-ACK, TW-Enabled, TW-Interrupt Enable
 TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN) | (1<<TWIE);
 sei();
 for(;;) {

 }
 return 1;
}
```

```c
/*
 * sensorenhet.c
 *
 * Version 1.0
 * Senast modifierad 2016-12-12
 *
 * Matilda Dahlström
 * Patrik Sletmo
 */

/*
 * sensorenhet.c
 *
 * Created: 11/7/2016 11:45:31 AM
 *  Author: antda685
 */

#include "common/main.h"
#include "common/protocol.h"
#include <avr/io.h>
#include "common/debug.h"
#include <stdbool.h>

void adc_init(void);
void adc_start(uint8_t channel);
bool adc_ready();
int to_mm(int n);
uint16_t adc_synch(uint8_t channel);
static struct sensor_data sd;
unsigned channel = MUX0;

void handle_data_request()
{
  return_sensor_data(&sd);
}

void handle_loop()
{
  if (adc_ready()) {
    if (channel == MUX0) {
      sd.ir_right_mm = to_mm(ADCW);
      channel = MUX1;
    }
    else if (channel == MUX1) {
      sd.ir_left_mm = to_mm(ADCW);
      channel = MUX2;
    }
    else if (channel == MUX2) {
      sd.ir_right_back_mm = to_mm(ADCW);
      channel = MUX3;
    }
    else if (channel == MUX3) {
      sd.ir_left_back_mm = to_cm_large(ADCW);
      channel = MUX0;
    }
    adc_start(channel);
  }
}


int main(void)
{
  // TODO: Register handlers
  initialize_uart();
  initialize_i2c(0x30);
  adc_init();

  printf("BOOT\n");

  listen_for_sensor_data_request(&handle_data_request);

  return run_program(&handle_loop);
}


void adc_init(void) {
  ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));  //FEL MATTE 16Mhz/128 = 125Khz
    the ADC reference clock
  ADMUX |= ((1<<REFS0)|(1<<REFS1)); //2.56 internal voltage as reference
  //ADMUX |= (1<<REFS0);              //Voltage reference, koppla 3.3v till AREF
  ADCSRA |= (1<<ADEN);              //Turn on ADC
  ADCSRA |= (1<<ADSC);              //Do an initial conversion because this one
    is the slowest and to ensure that everything is up and running
}

void adc_start(uint8_t channel){
  ADMUX &= 0xE0;             //Clear the older channel that was read
  ADMUX |= channel;          //Defines the new ADC channel to be read
  ADCSRA |= (1<<ADSC);       //Starts a new conversion
}

bool adc_ready(){
  return !(ADCSRA & (1<<ADSC));
}

uint16_t adc_synch(uint8_t channel){
  adc_start(channel);
  while(!adc_ready()) {};             //Wait until the conversion is done
  return ADCW;                     //Returns the ADC value of the chosen channel
}

int to_mm(int n) {
  const int min = 160;
  if (n > min + 840 || min > n) return -1;
  int data[840] = {206, 206, 205, 205, 204, 204, 203, 203, 203, 202, 202, 201,
    201, 200, 200, 200, 199, 199, 198, 198, 197, 197, 197, 196, 196, 195, 195,
    194, 194, 194, 193, 193, 192, 192, 192, 191, 191, 190, 190, 190, 189, 189,
    188, 188, 187, 187, 187, 186, 186, 185, 185, 185, 184, 184, 183, 183, 183,
    182, 182, 182, 181, 181, 180, 180, 180, 179, 179, 178, 178, 178, 177, 177,
    177, 176, 176, 175, 175, 175, 174, 174, 173, 173, 173, 172, 172, 172, 171,
    171, 171, 170, 170, 169, 169, 169, 168, 168, 168, 167, 167, 167, 166, 166,
    165, 165, 165, 164, 164, 164, 163, 163, 163, 162, 162, 162, 161, 161, 161,
    160, 160, 160, 159, 159, 158, 158, 158, 157, 157, 157, 156, 156, 156, 155,
    155, 155, 154, 154, 154, 153, 153, 153, 152, 152, 152, 151, 151, 151, 150,
    150, 150, 150, 149, 149, 149, 148, 148, 148, 147, 147, 147, 146, 146, 146,
    145, 145, 145, 144, 144, 144, 144, 143, 143, 143, 142, 142, 142, 141, 141,
    141, 140, 140, 140, 140, 139, 139, 139, 138, 138, 138, 137, 137, 137, 137,
    136, 136, 136, 135, 135, 135, 135, 134, 134, 134, 133, 133, 133, 133, 132,
    132, 132, 131, 131, 131, 131, 130, 130, 130, 129, 129, 129, 129, 128, 128,
    128, 127, 127, 127, 127, 126, 126, 126, 126, 125, 125, 125, 124, 124, 124,
    124, 123, 123, 123, 123, 122, 122, 122, 122, 121, 121, 121, 121, 120, 120,
    120, 119, 119, 119, 119, 118, 118, 118, 118, 117, 117, 117, 117, 116, 116,
    116, 116, 115, 115, 115, 115, 114, 114, 114, 114, 113, 113, 113, 113, 113,
    112, 112, 112, 112, 111, 111, 111, 111, 110, 110, 110, 110, 109, 109, 109,
    109, 108, 108, 108, 108, 108, 107, 107, 107, 107, 106, 106, 106, 106, 105,
    105, 105, 105, 105, 104, 104, 104, 104, 103, 103, 103, 103, 103, 102, 102,
    102, 102, 101, 101, 101, 101, 101, 100, 100, 100, 100, 99, 99, 99, 99, 99,
    98, 98, 98, 98, 98, 97, 97, 97, 97, 97, 96, 96, 96, 96, 96, 95, 95, 95, 95,
    94, 94, 94, 94, 94, 93, 93, 93, 93, 93, 92, 92, 92, 92, 92, 91, 91, 91, 91,
    91, 90, 90, 90, 90, 90, 90, 89, 89, 89, 89, 89, 88, 88, 88, 88, 88, 87, 87,
    87, 87, 87, 86, 86, 86, 86, 86, 86, 85, 85, 85, 85, 85, 84, 84, 84, 84, 84,
    84, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 81, 81, 81, 81, 81, 81, 80,
    80, 80, 80, 80, 80, 79, 79, 79, 79, 79, 78, 78, 78, 78, 78, 78, 77, 77, 77,
    77, 77, 77, 76, 76, 76, 76, 76, 76, 76, 75, 75, 75, 75, 75, 75, 74, 74, 74,
    74, 74, 74, 73, 73, 73, 73, 73, 73, 72, 72, 72, 72, 72, 72, 72, 71, 71, 71,
    71, 71, 71, 70, 70, 70, 70, 70, 70, 70, 69, 69, 69, 69, 69, 69, 69, 68, 68,
    68, 68, 68, 68, 68, 67, 67, 67, 67, 67, 67, 66, 66, 66, 66, 66, 66, 66, 66,
    65, 65, 65, 65, 65, 65, 65, 64, 64, 64, 64, 64, 64, 64, 63, 63, 63, 63, 63,
    63, 63, 62, 62, 62, 62, 62, 62, 62, 62, 61, 61, 61, 61, 61, 61, 61, 60, 60,
    60, 60, 60, 60, 60, 60, 59, 59, 59, 59, 59, 59, 59, 59, 58, 58, 58, 58, 58,
    58, 58, 58, 57, 57, 57, 57, 57, 57, 57, 57, 56, 56, 56, 56, 56, 56, 56, 56,
    55, 55, 55, 55, 55, 55, 55, 55, 55, 54, 54, 54, 54, 54, 54, 54, 54, 53, 53,
    53, 53, 53, 53, 53, 53, 53, 52, 52, 52, 52, 52, 52, 52, 52, 52, 51, 51, 51,
    51, 51, 51, 51, 51, 51, 50, 50, 50, 50, 50, 50, 50, 50, 50, 49, 49, 49, 49,
    49, 49, 49, 49, 49, 49, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 47, 47, 47,
    47, 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 45, 45,
    45, 45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 43,
    43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 42, 42, 42, 42, 42, 42, 42, 42, 42,
    42, 42, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40,
    40, 40, 40, 40, 40, 40, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 38,
    38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 37, 37, 37, 37, 37, 37, 37, 37,
    37, 37, 37, 37, 37, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 35,
    35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 34, 34,
    34, 34, 34};
  return data[n - min];
}

// For the 10cm-80cm sensor
int to_cm_large(int n) {
  const int min = 165;
  if (n > min + 775 || min > n) return -1;
  int data[775] = {77, 77, 76, 76, 76, 75, 75, 75, 74, 74, 74, 73, 73, 73, 72,
    72, 72, 71, 71, 71, 70, 70, 70, 69, 69, 69, 68, 68, 68, 68, 67, 67, 67, 66,
    66, 66, 65, 65, 65, 65, 64, 64, 64, 63, 63, 63, 63, 62, 62, 62, 61, 61, 61,
    61, 60, 60, 60, 59, 59, 59, 59, 58, 58, 58, 58, 57, 57, 57, 57, 56, 56, 56,
    56, 55, 55, 55, 55, 54, 54, 54, 54, 53, 53, 53, 53, 52, 52, 52, 52, 51, 51,
    51, 51, 51, 50, 50, 50, 50, 49, 49, 49, 49, 49, 48, 48, 48, 48, 47, 47, 47,
    47, 47, 46, 46, 46, 46, 46, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 43, 43,
    43, 43, 43, 42, 42, 42, 42, 42, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40, 40,
    39, 39, 39, 39, 39, 39, 38, 38, 38, 38, 38, 37, 37, 37, 37, 37, 37, 36, 36,
    36, 36, 36, 36, 35, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 34, 33, 33,
    33, 33, 33, 33, 33, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31, 31, 31,
    30, 30, 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 29, 29, 29, 28, 28, 28, 28,
    28, 28, 28, 28, 27, 27, 27, 27, 27, 27, 27, 27, 26, 26, 26, 26, 26, 26, 26,
    26, 26, 25, 25, 25, 25, 25, 25, 25, 25, 24, 24, 24, 24, 24, 24, 24, 24, 24,
    24, 23, 23, 23, 23, 23, 23, 23, 23, 23, 22, 22, 22, 22, 22, 22, 22, 22, 22,
    22, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 20, 20, 20, 20, 20, 20, 20,
    20, 20, 20, 20, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 18, 18, 18, 18,
    18, 18, 18, 18, 18, 18, 18, 18, 18, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
    17, 17, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 15, 15, 15,
    15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 14, 14, 14, 14, 14, 14,
    14, 14, 14, 14, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
    13, 13, 13, 13, 13, 13, 13, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
    12, 12, 12, 12, 12, 12, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
    11, 11, 11, 11, 11, 11, 11, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
    10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
    9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
    5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
    5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
    4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
    4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2};
  return data[n - min];

}


/*
intervall 280 < n < 1000
Mätdata från höger sensor [1023, 872, 761, 664, 590, 531, 476, 430, 400, 374,
    350, 327, 304, 289, 271, 258, 239]


//Python för att sampla funktionen
import math
from numpy import arange

min = 160
max = 1000
samples = arange(min, max, 5)
vals = list([int(str(round(291 * math.exp(-0.002155 * x), 0)
).replace(’.0’, ’’)) for x in range(min, max)])

print(vals)
*/



/*
intervall 165 < n < 940
Mätdata från höger sensor bak [940, 660, 515, 425, 365, 320, 285, 260, 240, 225,
     207, 200, 185, 175, 167]

import math
from numpy import arange

min = 160
max = 940
samples = arange(min, max, 5)
vals = list([int(str(round(159 * math.exp(-0.00453 * x), 0)
).replace(’.0’, ’’)) for x in range(min, max)])

print(vals)

*/
```

```c
/*
 * styrenhet.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Sebastian Callh
 * Patrik Sletmo
 */

/*
 * styrenhet.c
 *
 * Created: 11/8/2016 11:32:34 AM
 *  Author: antda685
 */

#include "common/main.h"
#include "common/protocol.h"
#include <limits.h>
#include <avr/io.h>
#include "common/debug.h"

const TOP = 16000;
const MAX_SPEED = 16000;

int converted_speed(unsigned char speed) {
  return (speed / 100.0) * MAX_SPEED;
}

void handle_motor_speed_received(struct motor_speed* ms) {
  signed char l = ms->left_speed;
  signed char r = ms->right_speed;

  printf("wheel speed received: l: %d, r: %d \n", l, r);


  if (l < 0) PORTA &= ~(1<<PORTA0);
  if (l >= 0) PORTA |= (1<<PORTA0);

  if (r < 0) PORTA &= ~(1<<PORTA2);
  if (r >= 0) PORTA |= (1<<PORTA2);

  unsigned char a_l = abs(l) + UINT_MAX + 1;
  unsigned char a_r = abs(r) + UINT_MAX + 1;

  printf("abs: l: %d, r: %d \n", a_l, a_r);

  int c_l = converted_speed(a_l);
  int c_r = converted_speed(a_r);

  printf("converted: l: %d, r: %d \n", c_l, c_r);

  OCR1A = c_l;  // Set speed left wheels
  OCR1B = c_r;  // Set speed right wheels
}

void handle_left_motor_speed_received(signed char speed) {
  OCR1A = converted_speed(speed);
}

void handle_right_motor_speed_received(signed char speed) {
  OCR1B = converted_speed(speed);
}

```

```
void handle_loop()
{

}

void initialize_PWM() {
  DDRA = 0xFF;  // Set Data Direction on PortA
  DDRD = 0xFF;  // Set Data Direction on PortD
  TCNT1 = 0;    // Reset Timer1 counter

  TCCR1A = 0;   // Clear Timer1 settings
  TCCR1B = 0;   // Clear Timer1 settings

  PORTA = 0x05; // Set direction of wheels (Pin 40 for left wheels, pin 38 for right wheels) (PORTA[0] left and PORTA[2] right)

  TCCR1B|= (1<<WGM12)|(1<<CS10)|(1<<WGM13);     // No prescaler, Timer1 settings
  TCCR1A|= (1<<COM1A1)|(1<<WGM11)|(1<<COM1B1);    // Timer1 settings
  ICR1 = TOP;                     // Set TOP

  // Set initial speed
  OCR1A = 0;
  OCR1B = 0;
}

int main(void)
{
  // TODO: Register handlers
  initialize_uart();
  initialize_i2c(0x40);
  initialize_PWM();

  listen_for_motor_speed_received(&handle_motor_speed_received);
  listen_for_left_motor_speed_received(&handle_left_motor_speed_received);
  listen_for_right_motor_speed_received(&handle_right_motor_speed_received);

  return run_program(&handle_loop);
}
```

```c
/*
 * debug.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * debug.c
 *
 * Created: 11/1/2016 3:59:37 PM
 *  Author: patsl736
 */

#include "config.h"
#include "debug.h"

#include <avr/io.h>
#include <inttypes.h>
#include <assert.h>
#include <stdbool.h>

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void USARTWriteChar(char data);
int USARTPutChar(char data, FILE *stream);

static FILE uart_stdout = FDEV_SETUP_STREAM(USARTPutChar, NULL,
    _FDEV_SETUP_WRITE);

bool uart_initialized = false;


void initialize_uart() {
  // Set baud rate
  UBRR0L = BAUD_PRESCALE;
  UBRR0H = (BAUD_PRESCALE>>8);

  // Enable RX and TX for USART
  UCSR0B=(1<<RXEN0)|(1<<TXEN0);

  // Redirect stdout to UART
  stdout = &uart_stdout;

  // Keep track of state to detect errors early
  uart_initialized = true;
}

void USARTWriteChar(char data)
{
  while(!(UCSR0A & (1<<UDRE0)))
  {
    // Busy wait until we can send data
  }

  UDR0=data;
}


int USARTPutChar(char data, FILE *stream)
{
  // Fail hard if UART is not initialized
  assert(uart_initialized);

  // Include carriage return to start at beginning of line
  if (data == ’\n’)
  {
    USARTWriteChar(’\r’);
  }

  USARTWriteChar(data);
  return 0;
}
```

```
/*
 * sensorenhet.c
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Sebastian Callh
 * Matilda Dahlström
 * Patrik Sletmo
 */

/*
 * sensorenhet.c
 *
 * Created: 11/2/2016 8:27:57 AM
 *  Author: matsj696
 */


#include <avr/io.h>
#include <stdbool.h>
#include <math.h>
#include <util/delay.h>
#include "debug.h"

void adc_init(void);
void adc_start(uint8_t channel);
bool adc_ready();
int to_mm(int n);
uint16_t adc_synch(uint8_t channel);

void adc_init(void) {
   ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));  //FEL MATTE 16Mhz/128 = 125Khz the ADC reference clock
   ADMUX |= ((1<<REFS0)|(1<<REFS1));  //2.56 internal voltage as reference
   //ADMUX |= (1<<REFS0);             //Voltage reference, koppla 3.3v till AREF
   ADCSRA |= (1<<ADEN);             //Turn on ADC
   ADCSRA |= (1<<ADSC);             //Do an initial conversion because this one is the slowest and to ensure that everything is up and running
 }

 void adc_start(uint8_t channel){
   ADMUX &= 0xE0;             //Clear the older channel that was read
   ADMUX |= channel;          //Defines the new ADC channel to be read
   ADCSRA |= (1<<ADSC);       //Starts a new conversion
}

bool adc_ready(){
  return !(ADCSRA & (1<<ADSC));
}

uint16_t adc_synch(uint8_t channel){
  adc_start(channel);
  while(!adc_ready()) {};            //Wait until the conversion is done
  return ADCW;                    //Returns the ADC value of the chosen channel
}

int main(void)
{
  unsigned channel = MUX0;
  uint16_t ir_left = 0;
  uint16_t ir_right = 0;

  initialize_uart();
  adc_init();

  while(1)
    {
    if (adc_ready()) {
      if (channel == MUX0) {
        ir_right = to_mm(ADCW);
        channel = MUX1;
      }
      else if (channel == MUX1) {
        ir_left = to_mm(ADCW);
        channel = MUX0;
      }
      adc_start(channel);
    }
    printf("left: %d, right: %d\n", ir_left, ir_right);
  }
}

int to_mm(int n) {
  const int min = 280;
  if (n > min + 720 || min > n) return -1;
  int data[720] = {159, 159, 158, 158, 158, 157, 157, 157, 156, 156, 156, 155,
    155, 155, 154, 154, 154, 153, 153, 153, 152, 152, 152, 151, 151, 151, 150,
    150, 150, 150, 149, 149, 149, 148, 148, 148, 147, 147, 147, 146, 146, 146,
    145, 145, 145, 144, 144, 144, 144, 143, 143, 143, 142, 142, 142, 141, 141,
    141, 140, 140, 140, 140, 139, 139, 139, 138, 138, 138, 137, 137, 137, 137,
    136, 136, 136, 135, 135, 135, 135, 134, 134, 134, 133, 133, 133, 133, 132,
    132, 132, 131, 131, 131, 131, 130, 130, 130, 129, 129, 129, 129, 128, 128,
    128, 127, 127, 127, 127, 126, 126, 126, 126, 125, 125, 125, 124, 124, 124,
    124, 123, 123, 123, 123, 122, 122, 122, 122, 121, 121, 121, 121, 120, 120,
    120, 119, 119, 119, 119, 118, 118, 118, 118, 117, 117, 117, 117, 116, 116,
    116, 116, 115, 115, 115, 115, 114, 114, 114, 114, 113, 113, 113, 113, 113,
    112, 112, 112, 112, 111, 111, 111, 111, 110, 110, 110, 110, 109, 109, 109,
    109, 108, 108, 108, 108, 108, 107, 107, 107, 107, 106, 106, 106, 106, 105,
    105, 105, 105, 105, 104, 104, 104, 104, 103, 103, 103, 103, 103, 102, 102,
    102, 102, 101, 101, 101, 101, 101, 100, 100, 100, 100, 99, 99, 99, 99, 99,
    98, 98, 98, 98, 98, 97, 97, 97, 97, 97, 96, 96, 96, 96, 96, 95, 95, 95, 95,
    94, 94, 94, 94, 94, 93, 93, 93, 93, 93, 92, 92, 92, 92, 92, 91, 91, 91, 91,
    91, 90, 90, 90, 90, 90, 90, 89, 89, 89, 89, 89, 88, 88, 88, 88, 88, 87, 87,
    87, 87, 87, 86, 86, 86, 86, 86, 86, 85, 85, 85, 85, 85, 84, 84, 84, 84, 84,
    84, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 81, 81, 81, 81, 81, 81, 80,
    80, 80, 80, 80, 80, 79, 79, 79, 79, 79, 78, 78, 78, 78, 78, 78, 77, 77, 77,
    77, 77, 77, 76, 76, 76, 76, 76, 76, 76, 75, 75, 75, 75, 75, 75, 74, 74, 74,
    74, 74, 74, 73, 73, 73, 73, 73, 73, 72, 72, 72, 72, 72, 72, 72, 71, 71, 71,
    71, 71, 71, 70, 70, 70, 70, 70, 70, 70, 69, 69, 69, 69, 69, 69, 69, 68, 68,
    68, 68, 68, 68, 68, 67, 67, 67, 67, 67, 67, 66, 66, 66, 66, 66, 66, 66, 66,
    65, 65, 65, 65, 65, 65, 65, 64, 64, 64, 64, 64, 64, 64, 63, 63, 63, 63, 63,
    63, 63, 62, 62, 62, 62, 62, 62, 62, 62, 61, 61, 61, 61, 61, 61, 61, 60, 60,
    60, 60, 60, 60, 60, 60, 59, 59, 59, 59, 59, 59, 59, 59, 58, 58, 58, 58, 58,
    58, 58, 58, 57, 57, 57, 57, 57, 57, 57, 57, 56, 56, 56, 56, 56, 56, 56, 56,
    55, 55, 55, 55, 55, 55, 55, 55, 55, 54, 54, 54, 54, 54, 54, 54, 54, 53, 53,
    53, 53, 53, 53, 53, 53, 53, 52, 52, 52, 52, 52, 52, 52, 52, 52, 51, 51, 51,
    51, 51, 51, 51, 51, 51, 50, 50, 50, 50, 50, 50, 50, 50, 50, 49, 49, 49, 49,
    49, 49, 49, 49, 49, 49, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 47, 47, 47,
    47, 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 45, 45,
    45, 45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 43,
    43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 42, 42, 42, 42, 42, 42, 42, 42, 42,
    42, 42, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40,
    40, 40, 40, 40, 40, 40, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 38,
    38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 37, 37, 37, 37, 37, 37, 37, 37,
    37, 37, 37, 37, 37, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 35,
    35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 34, 34,
    34, 34, 34};
  return data[n - min];
}
```

```

/*
intervall 280 < n < 1000
Mätdata från vänster sensor [1023, 872, 761, 664, 590, 531, 476, 430, 400, 374, 350, 327, 304, 289, 271, 258, 239]


//Python för att sampla funktionen
import math
from numpy import arange

min = 280
max = 1000
samples = arange(min, max, 5)
vals = list([int(str(round(29.1 * math.exp(-0.002155 * x), 0)
).replace('.0', '')) for x in range(min, max)])

print(vals)
*/
```

```
/*
 * config.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * config.h
 *
 * Created: 11/1/2016 4:00:18 PM
 *  Author: patsl736
 */


#ifndef CONFIG_H_
#define CONFIG_H_

// Processor frequency
#define F_CPU 16000000UL

// USART speed
#define USART_BAUDRATE 9600


#endif /* CONFIG_H_ */
```

```c
/*
 * debug.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * debug.h
 *
 * This header includes a function for initializing the UART functions by enabling the
 * ports RXD0 and TXD0, as well as the internal UART function. The UART interface is
 * made available by substituting stdout.
 *
 * This header also includes stdio.h which exports printf.
 *
 * Created: 11/1/2016 3:59:45 PM
 *  Author: patsl736
 */

#include <stdio.h>

#ifndef DEBUG_H_
#define DEBUG_H_

void initialize_uart();
void USARTWriteChar(char data);

#endif /* DEBUG_H_ */
```

```
/*
 * event.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * event.h
 *
 * Created: 11/7/2016 1:53:46 PM
 *  Author: antda685
 */


#ifndef EVENT_H_
#define EVENT_H_
#include "protocol.h"
typedef void(*void_func)();
typedef void(*sensor_data_func)(struct sensor_data* sd);

typedef void(*motor_speed_func)(struct motor_speed* ms);
typedef void(*left_motor_speed_func)(unsigned char speed);
typedef void(*right_motor_speed_func)(unsigned char speed);

extern void_func sensor_data_request;
extern sensor_data_func sensor_data_returned;

extern motor_speed_func motor_speed_received;
extern left_motor_speed_func left_motor_speed_received;
extern right_motor_speed_func right_motor_speed_received;

//-----------------SENSORENHET-------------------------
void listen_for_sensor_data_request(void_func vf);
void listen_for_sensor_data_returned(sensor_data_func sdf);

void notify_sensor_data_request();
void notify_sensor_data_returned(struct sensor_data* sd);

//-----------------STYRENHET-------------------------
void listen_for_motor_speed_received(motor_speed_func msf);
void listen_for_left_motor_speed_received(left_motor_speed_func lmsf);
void listen_for_right_motor_speed_received(right_motor_speed_func rmsf);

void notify_motor_speed_received(struct motor_speed* ms);
void notify_left_motor_speed_received(unsigned char speed);
void notify_right_motor_speed_received(unsigned char speed);



#endif /* EVENT_H_ */
```

```
/*
 * i2cslave.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * i2cslave.h
 *
 * Created: 11/2/2016 3:11:19 PM
 *  Author: antda685
 */


#ifndef I2CSLAVE_H_
#define I2CSLAVE_H_
#include "queue.h"
#include "packet.h"

void send_data(struct packet*);
struct packet* get_received_data();
void initialize_i2c (unsigned char address);

#endif /* I2CSLAVE_H_ */
```

```
/*
 * indexed_packet.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * packet_reader.h
 *
 * Created: 11/7/2016 1:18:42 PM
 *  Author: antda685
 */


#ifndef PACKET_READER_H_
#define PACKET_READER_H_

#include "packet.h"

struct indexed_packet
{
  unsigned int index;
  struct packet* p;
};

unsigned char read_byte(struct indexed_packet* p);
void write_byte(struct indexed_packet* p, unsigned char byte);

#endif /* PACKET_READER_H_ */
```

```
/*
 * main.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * main.h
 *
 * Created: 11/7/2016 11:41:03 AM
 *  Author: antda685
 */


#ifndef MAIN_H_
#define MAIN_H_

typedef void(*loopHandler)();

int run_program(loopHandler handler);

#endif /* MAIN_H_ */
```

```
/*
 * outbound.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * outbound.h
 *
 * Created: 11/7/2016 2:11:52 PM
 *  Author: antda685
 */


#ifndef OUTBOUND_H_
#define OUTBOUND_H_

#include "protocol.h"

void request_sensor_data();
void return_sensor_data(struct sensor_data* sd);


#endif /* OUTBOUND_H_ */
```

```
/*
 * packet.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * packet.h
 *
 * Created: 11/4/2016 3:35:21 PM
 *  Author: antda685
 */


#ifndef PACKET_H_
#define PACKET_H_

struct packet
{
  unsigned char data[256];
  unsigned int size;
};



#endif /* PACKET_H_ */
```

```c
/*
 * packet_parser.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * packet_parser.h
 *
 * Created: 11/7/2016 1:23:26 PM
 *  Author: antda685
 */


#ifndef PACKET_PARSER_H_
#define PACKET_PARSER_H_
#include "packet.h"

void parse_and_execute(struct packet* p);



#endif /* PACKET_PARSER_H_ */
```

```
/*
 * protocol.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-24
 *
 * Matilda Dahlström
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * protocol.h
 *
 * Created: 11/7/2016 1:24:20 PM
 *  Author: antda685
 */


#ifndef PROTOCOL_H_
#define PROTOCOL_H_


#define CMD_REQUEST_SENSOR_DATA 1
#define CMD_RETURN_SENSOR_DATA 2
#define CMD_PING 3
#define CMD_PONG 4
#define CMD_SET_MOTOR_SPEED 5
#define CMD_SET_LEFT_MOTOR_SPEED 6
#define CMD_SET_RIGHT_MOTOR_SPEED 7

struct sensor_data
{
  int ir_left_mm;
  int ir_right_mm;
  int ir_left_back_mm;
  int ir_right_back_mm;
};

struct motor_speed
{
  signed char left_speed;
  signed char right_speed;
};

#endif /* PROTOCOL_H_ */
```

```c
/*
 * queue.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * queue.h
 *
 * Created: 11/4/2016 11:30:10 AM
 *  Author: antda685
 */


#ifndef QUEUE_H_
#define QUEUE_H_

#include <stddef.h>
#include <stdbool.h>

struct queue
{
  struct queue* next;
  void* data;
};

struct queue* queue_create();
void queue_free(struct queue* q);
void* queue_front(struct queue* q);
void queue_pop(struct queue* q);
void queue_push(struct queue* q, void* data);
bool queue_empty(struct queue* q);

#endif /* QUEUE_H_ */
```

```
/*
 * config.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * config.h
 *
 * Created: 11/1/2016 4:00:18 PM
 *  Author: patsl736
 */


#ifndef CONFIG_H_
#define CONFIG_H_

// Processor frequency
#define F_CPU 16000000UL

// USART speed
#define USART_BAUDRATE 9600


#endif /* CONFIG_H_ */
```

```c
/*
 * debug.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Patrik Sletmo
 */

/*
 * debug.h
 *
 * This header includes a function for initializing the UART functions by enabling the
 * ports RXD0 and TXD0, as well as the internal UART function. The UART interface is
 * made available by substituting stdout.
 *
 * This header also includes stdio.h which exports printf.
 *
 * Created: 11/1/2016 3:59:45 PM
 *  Author: patsl736
 */

#include <stdio.h>

#ifndef DEBUG_H_
#define DEBUG_H_

void initialize_uart();

#endif /* DEBUG_H_ */
```

```
/*
 * config.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * config.h
 *
 * Created: 11/1/2016 4:00:18 PM
 *  Author: patsl736
 */


#ifndef CONFIG_H_
#define CONFIG_H_

// Processor frequency
#define F_CPU 16000000UL

// USART speed
#define USART_BAUDRATE 9600


#endif /* CONFIG_H_ */
```

```c
/*
 * debug.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * debug.h
 *
 * This header includes a function for initializing the UART functions by enabling the
 * ports RXD0 and TXD0, as well as the internal UART function. The UART interface is
 * made available by substituting stdout.
 *
 * This header also includes stdio.h which exports printf.
 *
 * Created: 11/1/2016 3:59:45 PM
 *  Author: patsl736
 */

#include <stdio.h>

#ifndef DEBUG_H_
#define DEBUG_H_

void initialize_uart();
void USARTWriteChar(char data);

#endif /* DEBUG_H_ */
```

```c
/*
 * packet.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * packet.h
 *
 * Created: 11/4/2016 3:35:21 PM
 *  Author: antda685
 */


#ifndef PACKET_H_
#define PACKET_H_

struct packet
{
  unsigned char data[256];
  unsigned int size;
};



#endif /* PACKET_H_ */
```

```c
/*
 * queue.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * queue.h
 *
 * Created: 11/4/2016 11:30:10 AM
 *  Author: antda685
 */


#ifndef QUEUE_H_
#define QUEUE_H_

#include <stddef.h>
#include <stdbool.h>

struct queue
{
  struct queue* next;
  void* data;
};

struct queue* queue_create();
void queue_free(struct queue* q);
void* queue_front(struct queue* q);
void queue_pop(struct queue* q);
void queue_push(struct queue* q, void* data);
bool queue_empty(struct queue* q);

#endif /* QUEUE_H_ */
```

```c
/*
 * i2cslave.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Anton Dalgren
 * Patrik Sletmo
 */

/*
 * i2cslave.h
 *
 * Created: 11/2/2016 3:11:19 PM
 *  Author: antda685
 */


#ifndef I2CSLAVE_H_
#define I2CSLAVE_H_
#include "common/queue.h"
#include "common/packet.h"
#define SLAVE_ADDRESS 0x30

void send_data(struct packet*);
struct packet* get_received_data();

#endif /* I2CSLAVE_H_ */
```

```
/*
 * config.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Matildha Sjöstedt
 * Patrik Sletmo
 */

/*
 * config.h
 *
 * Created: 11/1/2016 4:00:18 PM
 *  Author: patsl736
 */


#ifndef CONFIG_H_
#define CONFIG_H_

// Processor frequency
#define F_CPU 8000000UL

// USART speed
#define USART_BAUDRATE 9600


#endif /* CONFIG_H_ */
```

```c
/*
 * debug.h
 *
 * Version 1.0
 * Senast modifierad 2016-11-11
 *
 * Matildha Sjöstedt
 * Patrik Sletmo
 */

/*
 * debug.h
 *
 * This header includes a function for initializing the UART functions by enabling the
 * ports RXD0 and TXD0, as well as the internal UART function. The UART interface is
 * made available by substituting stdout.
 *
 * This header also includes stdio.h which exports printf.
 *
 * Created: 11/1/2016 3:59:45 PM
 *  Author: patsl736
 */

#include <stdio.h>

#ifndef DEBUG_H_
#define DEBUG_H_

void initialize_uart();

#endif /* DEBUG_H_ */
```

```
################################################################################
#                                                                              #
#                                 bt_client.py                                 #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-17                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

import bluetooth
import time
import traceback
import threading
import queue
import protocol
from bt_task import BT_task
from ast import literal_eval


class BT_client(threading.Thread):
    PI_ADDR = "B8:27:EB:FC:55:27"
    PORT = 3

    def __init__(self, queue_handler):
        self.queue_handler = queue_handler
        self.exit_demanded = False
        self.restart_demanded = False
        self.client_sock = None
        threading.Thread.__init__(self)
        self.daemon = True
        self.is_connected = False


    '''
    Creates a new client sock and attempts to connect to
    addr via port. Timeout can be specified, default value is
    10 seconds. The created socket is returned if connection
    was succesful, else return None.
    '''

    def _setup_bt_client(self, timeout=10):
        self.client_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
        self.client_sock.setblocking(True)

        while timeout > 0:
            try:
                self.client_sock.connect((self.PI_ADDR, self.PORT))
                print("Successfully connected to ", self.PI_ADDR)
                timeout = -1
            except bluetooth.btcommon.BluetoothError:
                time.sleep(1)
                timeout -= 1
                print("Waiting for connection...")
                continue
            except OSError:
                time.sleep(1)
                timeout -= 1
                print("OSError in _setup_bt_client (waiting connection)")
                continue

        if timeout == 0:
            print("Connection timeout. Could not connect to server!")
            return None
        else:
            print("Connected to ", self.PI_ADDR)
            self.is_connected = True

            return self.client_sock

    '''
         Loops send(),_recieve() until a shutdown or exit command is issued
         '''

    def _start_bt_client(self):
        self.client_sock = self._setup_bt_client()

        if self.client_sock:
            while True:
                self._send()
                status = self._receive()
                if status != "":
                    return status
        else:
            print("Could not connect to server, no socket created")

    def _send(self):
        bt_out_task = self.queue_handler.pop_out_queue()
        if bt_out_task:
            self.current_out_task = bt_out_task
            try:
                self.client_sock.send(str(bt_out_task.cmd_id))
            except bluetooth.btcommon.BluetoothError:
                pass
            except OSError:
                pass

            if bt_out_task.cmd_id == protocol.BT_SERVER_SHUTDOWN:
                self.exit_demanded = True
            elif bt_out_task.cmd_id == protocol.BT_SERVER_RESTART:
                self.restart_demanded = True
        else:
            self.current_out_task = BT_task(0, 0)


    def _receive(self):
        # Wait for incoming messages for 0.1 seconds
        recv_timeout = 0.1  # Receive timeout 0.1 seconds
        data = ""
        self.client_sock.settimeout(recv_timeout)
        try:
            data = self.client_sock.recv(1024).decode('utf-8')
        except bluetooth.btcommon.BluetoothError:
            # Recieved when server responds to shutdown
            pass
        except OSError:
            pass
        self.client_sock.settimeout(0)

        if self.restart_demanded:
            # Restart requested
            self.client_sock.close()
            del self.client_sock
            return "RESTART"
        elif self.exit_demanded:
            # Shutdown requested
            self.client_sock.close()
            del self.client_sock
            return "EXIT"

        if data:
            data = literal_eval(data)
            bt_in_task = BT_task(data[0], data[1])

            self.queue_handler.post_in_queue(bt_in_task)
            print("Bt client received: ", str(data[1]))

        return ""

    '''
    Overriden run()-method form threading.Thread.
    Updates client until a shutdown command is
    issued.
    '''

    def run(self):
        status = ""
        while not status == "EXIT":
            self.restart_demanded = False
            self.exit_demanded = False
            status = self._start_bt_client()
            if status == "ERROR":
                # TODO Add a task to out_queue
                print("A Bluetooth error occurred!")
            # Sleep so server has time to restart
            time.sleep(2)
```

```python
################################################################################
#                                                                              #
#                                  bt_task.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-27                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                                                              #
################################################################################

class BT_task:
    # NOTE If the client always asks for certain data
    # it might not need to check it when it arrives?

    def __init__(self, cmd_id=0, data=0):
        self.cmd_id = int(cmd_id)
        self.data = data
```

```
################################################################################
#                                                                              #
#                              bt_task_handler.py                              #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-27                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                                                              #
################################################################################

import pickle
from bt_task import BT_task

def clean_queue_files():
    # Create files or erase previous content
    answer_queue = open("bt_answers.txt", "w")
    answer_queue.seek(0)
    answer_queue.truncate()
    command_queue = open("bt_commands.txt", "w")
    command_queue.seek(0)
    command_queue.truncate()
    answer_queue.close()
    command_queue.close()


# kallas från main
def post_outgoing(bt_task):
    global busy_outgoing
    print("in post_outgoing and dumpint task with id", bt_task.cmd_id)
    answer_queue = open("bt_answers.txt", "wb")
    print("could open file")
    pickle.dump(bt_task, answer_queue)
    print("have dumped to pickle!")
    answer_queue.close()
    print("closing file and returning to main!")


# kallas från main
def pop_incoming():
    command_queue = open("bt_commands.txt", "rb")
    task = None

    task_q = []
    while (True):
        try:
            task_i = pickle.load(command_queue)
            task_q.append(task_i)
        except EOFError:
            break
    if task_q:
        task = BT_task(task_q[0].cmd_id, task_q[0].data)
        del task_q[0]
        command_queue = open("bt_commands.txt", "wb")
        for task_i in task_q:
            pickle.dump(task_i, command_queue)

    command_queue.close()
    return task


# kallas från server
def post_incoming(bt_task):
    command_queue = open("bt_commands.txt", "wb")
    print("task type in post_incoming ", type(bt_task))
    pickle.dump(bt_task, command_queue)
    print("Could dump to pickle in post_incoming")
    # pickle.Pickler.clear_memo(self=)
    command_queue.close()
    print("Closing file and return to bt_server")


# kallas från server
def pop_outgoing():
    answer_queue = open("bt_answers.txt", "rb")
    task = None

    task_q = []
    while (True):
        try:
            task_i = pickle.load(answer_queue)
            task_q.append(task_i)
        except EOFError:
            break
    if task_q:
        task = BT_task(task_q[0].cmd_id, task_q[0].data)
        del task_q[0]
        answer_queue = open("bt_answers.txt", "wb")
        for task_i in task_q:
            pickle.dump(task_i, answer_queue)

    answer_queue.close()
    return task
```

```
################################################################################
#                                                                              #
#                              bt_test_client.py                               #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-14                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

import bluetooth
import time
import traceback
import protocol

PI_ADDR = "B8:27:EB:FC:55:27"
USB_BT_ADDR = ""
PORT = 3


def setup_bt_client(addr, port):
    client_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
    client_sock.setblocking(True)
    timeout = 10
    while timeout > 0:
        try:
            client_sock.connect((addr, port))
            timeout = -1
        except bluetooth.btcommon.BluetoothError:
            time.sleep(1)
            timeout -= 1
            print("Waiting for connection...")
            continue

    if timeout == 0:
        print("Could not connect to server! PLZ try again and hope for better luck")
        return client_sock
    else:
        print("Successfully connected to ", addr)
        return client_sock


def main():
    restart = ""
    while not restart == "EXIT":
        restart = run()
        time.sleep(1)


def run():
    client_sock = setup_bt_client(PI_ADDR, PORT)

    while (True):
        msg = input("To server: ")

        client_sock.send(msg)

        data = ""

        try:
            while data == "":
                data = client_sock.recv(1024).decode('utf-8')
            if len(data) == 0:
                break
        except bluetooth.btcommon.BluetoothError:
            # Recieved when server responds to shutdown
            client_sock.close()
            del client_sock
            if int(msg) == protocol.BT_SERVER_RESTART:
                # Restart requested
                return "RESTART"
            elif int(msg) == protocol.BT_SERVER_SHUTDOWN:
                # Shutdown requested
                return "EXIT"


main()
```

```
################################################################################
#                                                                              #
#                                client_main.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from gui import GUI
import outbound
from protocol import *
from eventbus import EventBus
import datetime
from ast import literal_eval
import random

try:
    from bt_client import BT_client
    bluetooth_enabled = True
except:
    bluetooth_enabled = False

DATA_REQUEST_INTERVAL = 500  # milliseconds
IP_REQUEST_INTERVAL = 10     # seconds
UPDATE_INTERVAL = 250        # milliseconds

gui = None
bt_client = None
request_type = 0
last_data_request_time = datetime.datetime.now()
last_ip_request_time = datetime.datetime.now()


def run_bt_client(queue_handler):
    global bt_client
    bt_client = BT_client(queue_handler)
    bt_client.start()


def update_ip(data):
    global gui
    gui.update_ip(data)


def add_sensor_data(data):
    global gui
    gui.add_sensor_data(data)


def add_servo_data(data):
    global gui
    gui.add_servo_data(data)


def update_map(data):
    global gui
    print(data)
    gui.update_map(data)
```

```
def update_selected_mode(mode):
    gui.update_selected_mode(mode)


def setup_subscriptions():
    EventBus.subscribe(RETURN_PI_IP, update_ip)
    EventBus.subscribe(BT_RETURN_SENSOR_DATA, add_sensor_data)
    EventBus.subscribe(BT_RETURN_SERVO_DATA, add_servo_data)
    EventBus.subscribe(BT_RETURN_MAP_DATA, update_map)
    EventBus.subscribe(CMD_MODE_SET, update_selected_mode)


def request_data():
    global last_ip_request_time, request_type
    if request_type == 0:
        outbound.bt_request_sensor_data()
        request_type = 1
    else:
        outbound.bt_request_servo_data()
        request_type = 0
    outbound.bt_request_map_data()
    if (datetime.datetime.now() - last_ip_request_time) > datetime.timedelta(
            seconds=IP_REQUEST_INTERVAL):
        outbound.request_ip()
        last_ip_request_time = datetime.datetime.now()
    pass



def update():
    global gui, last_data_request_time, curr_test_corn, bt_client
    if not gui.exit_demanded:
        if gui.finished_setup:
            EventBus.receive()
            if (datetime.datetime.now() - last_data_request_time) > datetime.
    timedelta(
                    milliseconds=DATA_REQUEST_INTERVAL):
                if bt_client is not None and bt_client.is_connected:
                    request_data()

                last_data_request_time = datetime.datetime.now()
        else:
            gui.setup_after_main_loop()
        gui.canvas.after(UPDATE_INTERVAL, update)
    else:
        print("Exit gui in client main")
        outbound.bt_restart()
        while bt_client is not None and not bt_client.restart_demanded:
            pass
        gui.close_window()


def start_gui():
    global gui
    gui.canvas.after(UPDATE_INTERVAL, update)
    gui.root.mainloop()


def main():
    global gui
    queue_handler = EventBus.queue_handler
    setup_subscriptions()

    # MacOS has no support for PyBluez so by disabling the use of it we
    # can still provide a semi-functional experience for Mac users.
    if bluetooth_enabled:
        run_bt_client(queue_handler)
    else:
        print('NOTICE: PyBluez module could not be loaded!')
        print('Bluetooth functionality has been disabled.')

    gui = GUI()
    start_gui()

try:
    main()
except:
    print("Some error in client main")
    outbound.bt_restart()
    while bt_client is not None and not bt_client.restart_demanded:
        pass
    gui.close_window()
```

```python
################################################################################
#                                                                              #
#                                 eventbus.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-30                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

"""
Distributed event bus which is shared between all units on the main bus and via
Bluetooth.

The event bus provides a way to send data back and forth between different
units on the I2C bus and via Bluetooth by applying asynchronous transmission of
all events, it is therefore not guaranteed that messages are received on the
other end. As not all commands must be subscribed to it is also not certain
that the receiving unit actually reacts on the commands it receive.

In order for the event bus to function both ways the bus must be manually
polled for incoming messages by calling EventBus.receive(). This will read
pending commands from all connected AVR units and then call their respective
handlers if the command has been subscribed to.

Supported commands and their arguments are defined in protocol.py.
"""

from observer import Observer
from protocol import BLUETOOTH_ADDR
from queue_handlers import Queue_handler

# As reading from the bus is a blocking operation it might cause actual program
# code to execute too late if there are many pending commands available. In
# order to prevent the read operation to consume too much time the amount of
# messages read each iteration is limited.

MAX_READ_COUNT = 10


class EventBus:
    observers = {}
    queue_handler = Queue_handler()


    @staticmethod
    def post(addr, message):
        EventBus.queue_handler.post_out_queue(message)

    @staticmethod
    def pop(addr):
        return EventBus.queue_handler.pop_in_queue()

    @staticmethod
    def receive():
        EventBus.receive_from_addr(BLUETOOTH_ADDR)

    @staticmethod
    def receive_from_addr(unit_addr):
        for i in range(MAX_READ_COUNT):
            data = EventBus.pop(unit_addr)
            if data is None:
                break

            EventBus.notify(data.cmd_id, data.data)

    @staticmethod
    def subscribe(command_id, handler):
        observer = EventBus._get_observer_for_command(command_id)
        observer.subscribe(handler)

    @staticmethod
    def notify(command_id, *args):
        observer = EventBus._get_observer_for_command(command_id)
        observer.notify(*args)

    @staticmethod
    def _get_observer_for_command(command_id):
        if command_id not in EventBus.observers:
            EventBus.observers[command_id] = Observer()

        return EventBus.observers[command_id]
```

```
################################################################################
#                                                                              #
#                                    gui.py                                    #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from tkinter import *
import outbound
from map_grid import MapGrid
import datetime


class GUI:
    WINDOW_X = 800
    WINDOW_Y = 600
    CANVAS_X = int(WINDOW_X * 0.6)
    CANVAS_Y = CANVAS_X
    LIST_FRAME_X = int(WINDOW_X * 0.3)
    LIST_FRAME_Y = int(WINDOW_Y * 0.75)
    LIST_BOX_Y = int(LIST_FRAME_Y * 0.5)
    LIST_BOX_X = int(LIST_FRAME_X * 0.5)
    BTN_FRAME_X = int(WINDOW_X)
    BTN_FRAME_Y = int(WINDOW_Y * 0.2)
    BG_COLOR = "orange"

    MAX_LIST_ITEMS = 13
    MIN_TIME_KEY_EVENT = 250  # milliseconds

    ROBOT_MODE_MANUAL = 0
    ROBOT_MODE_AUTONOMOUS = 1

    MODES =[("Manual", ROBOT_MODE_MANUAL),
            ("Automatic", ROBOT_MODE_AUTONOMOUS)]

    def __init__(self):
        self.pi_ip = ""
        self.exit_demanded = False
        self.finished_setup = False

        self.map_grid = MapGrid()

        self.root = Tk()
        self.root.protocol("WM_DELETE_WINDOW", self.exit)
        self.root.title("Kartoffel control")
        self.main_frame = Frame(self.root, width=self.WINDOW_X, height=self.WINDOW_Y, bg=self.BG_COLOR)
        self.main_frame.focus_set()  # Set all frame as listening to keyboard events
        self.main_frame.grid()

        # Keybindings
        # Run functions when certain keys are pressed. Bind to same as buttons.
        # Arrow keys bind to root instead of main frame because of keyboard focus
        self.main_frame.bind('<w>', self.forward)
        self.root.bind('<Up>', self.forward)
        self.main_frame.bind('<s>', self.back)
        self.root.bind('<Down>', self.back)
        self.main_frame.bind('<a>', self.left)
        self.root.bind('<Left>', self.left)
        self.main_frame.bind('<d>', self.right)
        self.root.bind('<Right>', self.right)
        self.main_frame.bind('<q>', self.forward_left)
        self.main_frame.bind('<e>', self.forward_right)

        self.last_key_event_time = datetime.datetime.now()

        # --- Canvas ---
        self.canvas = Canvas(self.main_frame,
                             width=GUI.CANVAS_X,
                             height=GUI.CANVAS_Y, bg="#CCCCCC")
        self.canvas.grid(column=0, row=0, padx=10, pady=10)

        # --- Lists ---
        self.list_frame = Frame(self.main_frame, width=self.LIST_FRAME_X, height=self.LIST_FRAME_Y,
                                bg=self.BG_COLOR)
        self.list_frame.grid(row=0, column=1, padx=10)

        self.ir_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
        self.ir_list.grid(row=1, column=0)
        self.ir_list_nr_items = 0
        self.ir_label = Label(self.list_frame, text="IR data (mm) \n (left_back, left, right, right_back)",
                              fg="black", bg=self.BG_COLOR)
        self.ir_label.grid(row=0, column=0)

        self.laser_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
        self.laser_list.grid(row=1, column=1)
        self.laser_list_nr_items = 0
        self.laser_label = Label(self.list_frame, text="Laser data", fg="black", bg=self.BG_COLOR)
        self.laser_label.grid(row=0, column=1)

        self.gyro_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
        self.gyro_list.grid(row=3, column=0)
        self.gyro_list_nr_items = 0
        self.gyro_label = Label(self.list_frame, text="Gyro data \n (degrees)", fg="black", bg=self.BG_COLOR)
        self.gyro_label.grid(row=2, column=0)

        self.servo_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
        self.servo_list.grid(row=3, column=1)
        self.servo_list_nr_items = 0
        self.servo_label = Label(self.list_frame, text="Servo data \n (speed)", fg="black", bg=self.BG_COLOR)
        self.servo_label.grid(row=2, column=1)

        # --- Buttons ---
        self.btn_frame = Frame(self.main_frame, width=self.BTN_FRAME_X, height=self.BTN_FRAME_Y)
        self.btn_frame.grid(row=1, column=0, pady=10, padx=10)

        self.btn_forward = Button(self.btn_frame, text="Forward", command=self.forward)
        self.btn_forward.grid(row=1, column=3)

        self.btn_back = Button(self.btn_frame, text="Back", command=self.back)
        self.btn_back.grid(row=1, column=4)

        self.btn_right = Button(self.btn_frame, text="Right", command=self.right)
        self.btn_right.grid(row=1, column=5)

        self.btn_left = Button(self.btn_frame, text="Left", command=self.left)
        self.btn_left.grid(row=1, column=1)

        self.btn_forward_right = Button(self.btn_frame, text="Forward left", command=self.forward_left)
        self.btn_forward_right.grid(row=0, column=3, padx=5, pady=2)

        self.btn_forward_left = Button(self.btn_frame, text="Forward right", command=self.forward_right)
        self.btn_forward_left.grid(row=0, column=4, padx=5, pady=2)

        self.mode = IntVar()
        self.mode.set(1)
        self.radio_frame = Frame(self.btn_frame)
        self.radio_frame.grid(row=0, column=6)
        self.btn_auto_mode = Radiobutton(self.radio_frame, text=self.MODES[0][0], variable=self.mode,
                                 command=self.change_mode, indicatoron=0, value=self.MODES[0][1])
        self.btn_auto_mode.grid(row=0, column=0, padx=2, pady=2, sticky="W")
        self.btn_manual_mode = Radiobutton(self.radio_frame, text=self.MODES[1][0], variable=self.mode,
                                         command=self.change_mode, indicatoron=0, value=self.MODES[1][1])
        self.btn_manual_mode.grid(row=1, column=0, padx=2, pady=2, sticky="W")

        self.ip_box = Label(self.main_frame, text="Pi IP: ", width=25, bg="white")
        self.ip_box.grid(row=1, column=1)

        # --- Image ----
        self.image_frame = Frame(self.btn_frame)
        self.image_frame.grid(row=0, column=0, rowspan=2)
        logo = PhotoImage(file="Logo.gif")
        self.resampled_logo = logo.subsample(3, 3)
        self.logo_box = Label(self.image_frame, image=self.resampled_logo)
        self.logo_box.grid(row=1, column=0, padx=10, sticky=W+E+N+S)

        self.map_grid.draw_grid(self.canvas)

    def setup_after_main_loop(self):
        self.map_grid.draw_grid(self.canvas)
        self.btn_auto_mode.select()
        self.btn_manual_mode.deselect()
        self.finished_setup = True

    '''
    Values should be a list containing of [ir_left,ir_right,ir_left_back,
    ir_right_back,laser,gyro]
    '''

    def add_sensor_data(self, values):
        ir_values = str(values[2]) + ", " + str(values[0]) + ", " + str(values[1]) + ", " + str(values[3])

        self.ir_list.insert(0, ir_values)
        if self.ir_list_nr_items >= self.MAX_LIST_ITEMS:
            self.ir_list.delete(self.MAX_LIST_ITEMS)
        else:
            self.ir_list_nr_items += 1

        self.laser_list.insert(0, str(values[4]))
        if self.laser_list_nr_items >= self.MAX_LIST_ITEMS:
            self.laser_list.delete(self.MAX_LIST_ITEMS)
        else:
            self.laser_list_nr_items += 1

        self.gyro_list.insert(0, str(values[5]))
        if self.gyro_list_nr_items >= self.MAX_LIST_ITEMS:
            self.gyro_list.delete(self.MAX_LIST_ITEMS)
        else:
            self.gyro_list_nr_items += 1

    '''
    Values should be a list containing of [left_speed, right_speed].
    '''

    def add_servo_data(self, values):
        self.servo_list.insert(0, str(values[0]) + ', ' + str(values[1]))
        if self.servo_list_nr_items >= self.MAX_LIST_ITEMS:
            self.servo_list.delete(self.MAX_LIST_ITEMS)
        else:
            self.servo_list_nr_items += 1

    def update_map(self, values):
        self.map_grid.update_map(values, self.canvas)

    '''
    Ip expected to be in format [ip]
    '''

    def update_ip(self, ip):
        self.ip_box.config(text="Pi IP: " + str(ip[0]))

    def exit(self):
        self.exit_demanded = True

    def close_window(self):
        self.root.destroy()

    def check_key_event_time(self):
        return (datetime.datetime.now() - self.last_key_event_time) > datetime.
    timedelta(
            milliseconds=self.MIN_TIME_KEY_EVENT)

    '''Functions for handling key press.
    Takes forced event, but ignores it and calls correct driver function.
    '''

    def forward(self, event=None):
        self.event_handler(outbound.bt_drive_forward, event=event, repetition=5)

    def back(self, event=None):
        self.event_handler(outbound.bt_drive_back, event=event, repetition=5)

    def left(self, event=None):
        self.event_handler(outbound.bt_turn_left, event=event, repetition=3)

    def right(self, event=None):
        self.event_handler(outbound.bt_turn_right, event=event, repetition=3)

    def forward_right(self, event=None):
        self.event_handler(outbound.bt_forward_right, event=event, repetition=5)

    def forward_left(self, event=None):
        self.event_handler(outbound.bt_forward_left, event=event, repetition=5)

    '''
    Makes sure event can not occur faster than a predefined time interval.
    If event option is left out or set to None, event_handler will interpret that as if
    it was called from a button and call the command function number of times
    specified in repetition option.
    '''

    def event_handler(self, command, **options):
        if self.check_key_event_time():
            if not options["event"]:
                for i in range(0, options["repetition"]):
                    while not self.check_key_event_time():
                        continue
                    command()
                    self.last_key_event_time = datetime.datetime.now()
            else:
                command()
                self.last_key_event_time = datetime.datetime.now()

    def change_mode(self):
        mode = self.mode.get()
        if self.MODES[mode][0] == "Manual":
            outbound.bt_switch_to_manual()
        else:
            outbound.bt_switch_to_auto()

    def update_selected_mode(self, mode):
        # btn_auto_mode and btn_manual_mode are "reversed", so btn_auto_mode
        # has the text "Manual" and btn_manual_mode the text "Automatic"
        if mode == GUI.ROBOT_MODE_MANUAL:
            self.btn_auto_mode.select()
            self.btn_manual_mode.deselect()
        elif mode == GUI.ROBOT_MODE_AUTONOMOUS:
            self.btn_auto_mode.deselect()
            self.btn_manual_mode.select()
```

```
################################################################################
#                                                                              #
#                                  inbound.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-30                         #
#                                                                              #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################
```

```python
################################################################################
#                                                                              #
#                                 map_grid.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-14                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

class MapGrid:
    MAX_MAP_SIZE = 15 * 2  # 1 cell is 40x40 cm, 28 x 28 cells in map
    NR_ROWS = MAX_MAP_SIZE
    NR_COLS = NR_ROWS
    OFFSET = 15

    def __init__(self):
        # List of coordinates of all corners on the map,
        # listed in the order of discovery
        self.raw_map_data = []
        self.actual_map_data = []
        self.new_raw_map_data = []  # The latest map data received
        self.new_actual_map_data = []
        self.start_position = (self.OFFSET, self.OFFSET)
        # Offset calculated from given coordinates to match coordinates on map grid

    def update_map(self, data, canvas):
        self._update_map_data(data)
        self.new_actual_map_data = []
        self._calc_actual_coords(canvas)
        self._draw_blocks(canvas)

    def draw_grid(self, canvas):
        size = canvas.winfo_width() / self.NR_ROWS
        for row in range(0, self.NR_ROWS):
            canvas.create_line(0, row * size, canvas.winfo_width(), row * size, fill="#FFFFFF")
            canvas.create_line(row * size, 0, row * size, canvas.winfo_width(), fill="#FFFFFF")

    '''
    Expects data to be a list containing ALL map data coordinates.
    '''

    def _update_map_data(self, data):
        visited = data[0]
        nr_new_items = len(visited) - len(self.raw_map_data)
        self.new_raw_map_data = visited[:nr_new_items]
        self.raw_map_data = visited
        self.new_raw_map_data = visited

    '''
    Appends internal list of corners with actual coordinates corresponding to
    size of map in pixels.
    '''

    def _calc_actual_coords(self, canvas):
        visited = self.new_raw_map_data
        for block in visited:
            # Match coordinates to grid
            if block:
                x = block[0]
                y = 0 - block[1] # Flip coordinate system on canvas
                x += self.OFFSET
                y += self.OFFSET

                # Convert raw coordinates to actual coordinates corresponding to canvas pixels
                actual_x = (canvas.winfo_width() * x) / self.NR_COLS
                actual_y = (canvas.winfo_height() * y) / self.NR_ROWS
                self.new_actual_map_data.append((actual_x, actual_y))

    '''
    Draw lines between corners. Replaces map data with actual coordinates with only the last
    visited corner.
    '''

    def _draw_blocks(self, canvas):
        block_size = canvas.winfo_height() / self.NR_ROWS

        visited = self.new_actual_map_data
        for block in visited:
            canvas.create_rectangle(block[0], block[1], block[0] + block_size, block[1] + block_size,
                                    fill="#FFFFFF", outline="white")
            self.actual_map_data.append(block)
```

```python
################################################################################
#                                                                              #
#                                 observer.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-27                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                                                              #
################################################################################

"""
Simple implementation of the observer pattern.

See https://en.wikipedia.org/wiki/Observer_pattern for more information.
"""


class Observer:
    def __init__(self):
        self.subscribers = []

    def subscribe(self, func):
        if func not in self.subscribers:
            self.subscribers.append(func)

    def notify(self, *args, **kwargs):
        for subscriber in self.subscribers:
            subscriber(*args, **kwargs)
```

```
################################################################################
#                                                                              #
#                                 outbound.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-07                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

"""
This file contains functions for interacting with the two different AVR units.
All functions defined here are outbound which mean they go from the main unit
to one of the AVR units.

The message passing function is implemented as a distributed event bus which
in its distributed nature depends on asynchronous functionality. This means
that messages sent are not executed immediately and there is no guarantee that
the sent command is actually executed on the receiving unit.

For more information see eventbus.py.
"""

from eventbus import EventBus
from protocol import *
from bt_task import BT_task


# NOTE: Function comments are purposely left out from this file in favor of the
# complete definitions of every found command in proctol.py.

def request_ip():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            REQUEST_PI_IP
        )
    )


def bt_request_sensor_data():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_REQUEST_SENSOR_DATA
        )
    )


def bt_request_servo_data():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_REQUEST_SERVO_DATA
        )
    )


def bt_request_map_data():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_REQUEST_MAP_DATA
        )
    )


def bt_drive_forward():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_DRIVE_FORWARD
        )
    )


def bt_drive_back():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_DRIVE_BACK
        )
    )


def bt_turn_right():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_TURN_RIGHT
        )
    )

def bt_forward_right():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_DRIVE_FORWARD_RIGHT
        )
    )

def bt_turn_left():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_TURN_LEFT
        )
    )

def bt_forward_left():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_DRIVE_FORWARD_LEFT
        )
    )

def bt_shutdown():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_SERVER_SHUTDOWN
        )
    )


def bt_restart():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_SERVER_RESTART
        )
    )

def bt_switch_to_auto():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            AUTONOMOUS_MODE
        )
    )


def bt_switch_to_manual():
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            MANUAL_MODE
        )
    )
```

```
################################################################################
#                                                                              #
#                                 protocol.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-17                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
The protocol for the robot consist of various commands, or events, passed along
the main bus using a distributed event bus. Each command is identified by its
command id and is transmitted before eventual arguments.

All commands may be sent in any direction but the implementations will probably
choose to ignore irrelevant one. For messages originating from the main unit,
see outbound.py.
"""

# Addresses for the units on the bus. Note that the laser cannot be queried
# using the protocol described in bus.py.


BLUETOOTH_ADDR = 0xBEEF

# Packet addresses
PACKET_HEADER = 0
PACKET_DATA = 1

# Request data from the sensor unit
CMD_REQUEST_SENSOR_DATA = 1
"""
Issues a request to the sensor unit prompting it to send its most recent sensor
data back to the main unit. As the data transfer is asynchronous the sensor
unit responds to the request by posting a CMD_RETURN_SENSOR_DATA command on the
bus.

Target: Sensor unit

Arguments: None
"""

# Return sensor data from the sensor unit
CMD_RETURN_SENSOR_DATA = 2
"""
Command sent from the sensor unit after a request for its sensor data has been
made. As data from various sensors are reported independently it is not certain
that the value from sensor A and value from sensor B reflect the world at the
same point in time.

Target: Main unit

Arguments:
ir_left_mm (2 bytes, two's complement)
    Distance recorded by the left IR sensor in mm, or -1 if the distance is not
    within the supported range.
ir_right_mm (2 bytes, two's complement)
    Distance recorded by the right IR sensor in mm, or -1 if the distance is
    not within the supported range.
"""

```

```
# Ping a unit
CMD_PING = 3
"""
Dummy command which will only trigger the other unit to respond with a PONG
command. Preferably used to test a connection without forcing the other party
to perform any actual action.

Target: Any AVR unit

Arguments: None
"""

# Pong a unit
CMD_PONG = 4
"""
Reply to a PING command.

Target: Main unit

Arguments: None
"""

# Set both motor speeds in the control unit
CMD_SET_MOTOR_SPEED = 5
"""
Sets the speed of both the left and right motors on the robot. Values can
be both positive and negative and the range -100 to 100 is supported, where the
value represents a percentage of the max speed. It seems like the different
directions have various max speeds, which has to be considered when
implementing on-spot-rotation or other actions which assume equal speed forward
and backwards.

Target: Control unit

Arguments:
left_motor_speed (1 byte, positive or negative)
    Left motor speed in percentage of max speed ranging from -100 to 100.
right_motor_speed (1 byte, positive or negative)
    Right motor speed in percentage of max speed ranging from from -100 to 100.
"""

# Set left motor speed only in control unit
CMD_SET_LEFT_MOTOR_SPEED = 6
"""
Sets the speed of the left motors on the robot. It is only possible to pass
positive values with the command in the range 0 to 100, where the value
represent a percentage of the max speed.

Target: Control unit

Arguments:
speed (1 byte, positive)
    Left motor speed in percentage of max speed ranging from 0 to 100.
"""

# Set right motor speed in the control unit
CMD_SET_RIGHT_MOTOR_SPEED = 7
"""
Sets the speed of the right motors on the robot. It is only possible to pass
positive values with the command in the range 0 to 100, where the value
represent a percentage of the max speed.

Target: Control unit

Arguments:
```

```
speed (1 byte, positive)
    Right motor speed in percentage of max speed ranging from 0 to 100.
"""
# Indicates that the robot has started turning
"""
Event called internally within the main unit to indicate that a simple 90
degree turn has been initiated.

Target: Main unit

Arguments: None
"""
CMD_TURN_STARTED = 8

# Indicates that the robot has stopped turning
"""
Event called internally within the main unit to indicate that a simple 90
degree turn has finished.

Target: Main unit

Arguments:
is_right_turn (1 byte, boolean)
    True for right turn, false for left turn.
"""
CMD_TURN_FINISHED = 9

# -------------------- Bluetooth commands ---------------------

REQUEST_PI_IP = 10
RETURN_PI_IP = 11

BT_SERVER_RESTART = 12

BT_SERVER_SHUTDOWN = 13

BT_REQUEST_SENSOR_DATA = 14
BT_RETURN_SENSOR_DATA = 15

BT_REQUEST_SERVO_DATA = 16
BT_RETURN_SERVO_DATA = 17

BT_REQUEST_MAP_DATA = 18
BT_RETURN_MAP_DATA = 19

BT_DRIVE_FORWARD = 20
BT_DRIVE_BACK = 21

BT_TURN_RIGHT = 22
BT_TURN_LEFT = 23

BT_DRIVE_FORWARD_RIGHT = 24
BT_DRIVE_FORWARD_LEFT = 25

AUTONOMOUS_MODE = 26
MANUAL_MODE = 27

# Indicates that the robot has changed to a new navigator mode
CMD_MODE_SET = 29
"""
Command to toggle between the available modes (autonomous and manual) instead
of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.

Target: Main unit

```

```
Arguments:
new_mode (1 byte)
    Integer representation of the new mode:
      0 = Manual mode
      1 = Autonomous mode
"""

BT_CLIENT_COMMANDS = [REQUEST_PI_IP, BT_SERVER_RESTART,
                      BT_SERVER_SHUTDOWN, BT_REQUEST_SENSOR_DATA,
                      BT_REQUEST_MAP_DATA, BT_REQUEST_SERVO_DATA,
    BT_DRIVE_FORWARD, BT_DRIVE_BACK,
                      BT_TURN_RIGHT, BT_TURN_LEFT, BT_DRIVE_FORWARD_RIGHT,
    BT_DRIVE_FORWARD_LEFT]

BT_SERVER_COMMANDS = [RETURN_PI_IP, BT_RETURN_SENSOR_DATA,
                      BT_RETURN_SERVO_DATA, BT_RETURN_MAP_DATA, CMD_MODE_SET]
```

```
################################################################################
#                                                                              #
#                              queue_handlers.py                               #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

import queue

QUEUE_MAX_SIZE = 40


class Queue_handler:
    def __init__(self):
        (self.in_queue, self.out_queue) = self.create_task_queues()

    def create_task_queues(self):
        self.in_queue = queue.Queue(QUEUE_MAX_SIZE)
        self.out_queue = queue.Queue(QUEUE_MAX_SIZE)
        return (self.in_queue, self.out_queue)

    def pop_in_queue(self):
        try:
            next_task = self.in_queue.get(False)
            self.in_queue.task_done()
        except queue.Empty:
            next_task = None
        print("In queue size: ", int(self.in_queue.qsize()))
        return next_task

    def post_in_queue(self, task):
        try:
            self.in_queue.put(task, timeout=0.75)
        except queue.Full:
            print("In_queue is full, ignoring new messages.")
        #print("In queue size: ", int(self.in_queue.qsize()))

    def pop_out_queue(self):
        try:
            next_task = self.out_queue.get(False)
            self.out_queue.task_done()
        except queue.Empty:
            next_task = None
        print("Out queue size: ", int(self.out_queue.qsize()))
        return next_task

    def post_out_queue(self, task):
        try:
            self.out_queue.put(task, timeout=0.75)
        except queue.Full:
            print("Out_queue is full, ignoring new messages.")
        #print("Out queue size: ", int(self.out_queue.qsize()))
```

```
################################################################################
#                                                                              #
#                                   accel.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-20                         #
#                                                                              #
#                                Anton Dalgren                                 #
#                                                                              #
################################################################################

from time import sleep

import Adafruit_LSM303

lsm303 = Adafruit_LSM303.LSM303()
class Accel:

    @staticmethod
    def read_data():
        try:
            accel, mag = lsm303.read()
            accel_x, accel_y, accel_z = accel
            mag_x, mag_y, mag_z = mag
            accel_x = accel_x * 0.001 * 9.82
            if(abs(accel_x) < 0.2):
                return 0
            return accel_x
        except:
            return -1
```

```
################################################################################
#                                                                              #
#                                acceldriver.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-20                         #
#                                                                              #
#                                Anton Dalgren                                 #
#                                                                              #
################################################################################

from time import sleep
#from datetime import datetime, timedelta

from accel import Accel
#from driver import Driver

accel = Accel()

while (True):
    data = accel.read_data()
    print("x-accel: " + str(data))
    sleep(0.2)
```

```python
################################################################################
#                                                                              #
#                              autocontroller.py                               #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

import datetime
from math import floor

time_last_regulation = datetime.datetime.now()
use_derivate = True
old_error = 0
integral = 0
last_diff = 0
last_valid_diffs = []
last_valid_diff = 0

QUEUE_SIZE = 5

class AutoController:
    DESIRED_DISTANCE = 120  # Desired distance to wall
    STANDARD_SPEED = 40
    MAX_REGULATION = 30

    def auto_control(self, ir_right_mm, ir_right_back_mm, reg_side):
        global use_derivate, time_last_regulation, old_error, integral, last_diff, last_valid_diff, last_valid_diffs

        Kp = float(0.2)
        Ka = float(0.3)

        time_now = datetime.datetime.now()
        sensor_data_front = ir_right_mm
        sensor_data_back = ir_right_back_mm
        dist_diff = (sensor_data_back - sensor_data_front)

        regulation_error = self.DESIRED_DISTANCE - sensor_data_front + abs(dist_diff / 10)


        if (sensor_data_front == -1 or sensor_data_back == -1 or abs(dist_diff) > 70):
            dist_diff = 0
            regulation_error = 0
        else:
            if len(last_valid_diffs) >= QUEUE_SIZE:
                last_valid_diffs = last_valid_diffs[1:QUEUE_SIZE] + [dist_diff]
            else:
                last_valid_diffs = last_valid_diffs + [dist_diff]

            last_valid_diff = last_valid_diffs[0]

        regulation = floor((Kp * regulation_error) + Ka * dist_diff)

        old_error = regulation_error
        last_diff = dist_diff

        if (regulation > self.MAX_REGULATION):
            regulation = self.MAX_REGULATION
        elif (regulation < -self.MAX_REGULATION):
            regulation = -self.MAX_REGULATION

        if (regulation > -10):
            speed_close_wall = self.get_speed(ir_right_mm, ir_right_back_mm) +
    regulation
        else:
            speed_close_wall = 10

        if (regulation < 10):
            speed_far_wall = self.get_speed(ir_right_mm, ir_right_back_mm) -
    regulation
        else:
            speed_far_wall = 10

        time_last_regulation = time_now

        return int(speed_close_wall), int(speed_far_wall), regulation

    def get_speed(self, ir_right_mm, ir_right_back_mm):
        if ir_right_mm == -1 and ir_right_back_mm != -1:
            return self.STANDARD_SPEED
        else:
            return self.STANDARD_SPEED
```

```
################################################################################
#                                                                              #
#                               bt_server.py                                   #
#                                                                              #
#                               Version 1.0                                    #
#                       Senast modifierad 2016-12-12                           #
#                                                                              #
#                             Rebecca Lindblom                                 #
#                            Matildha Sjöstedt                                 #
#                              Patrik Sletmo                                   #
#                                                                              #
################################################################################

import bluetooth
import bt_task_handler
from bt_task import BT_task


class BT_Server:
    """
    Class for handling the Bluetooth connection to a client.
    Gives an interface for sending and receiving data between
    robot and client.
    """

    def __init__(self, server_addr, port, backlog, client_addr=""):
        # Sever bluetooth mac-address
        self.server_addr = server_addr
        self.port = port
        # Number of unaccepted connections before refusing new ones
        self.backlog = backlog
        # (the only address from which the server will accept connections)
        self.client_addr = client_addr

        # Set up server socket
        self.server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
        self.server_sock.setblocking(True)
        self.server_sock.bind((server_addr, port))
        # Enable the server to accept connections
        self.server_sock.listen(backlog)

        # Data received from client
        self.incoming_data = None
        # Data to be sent to client
        self.outgoing_data = None

        self.client_sock = None
        self.accp_client_addr = None

    def accept_connection(self):
        """
        Connects the server to the client trying to connect.
        """
        # TODO: Accept connection from valid client (requires change of backlog)
        (self.client_sock, self.accp_client_addr) = self.server_sock.accept()
        print("Connected")

    def post_to_incoming(self):
        """
        Puts saved incoming data to queue to robot.
        """
        bt_task_handler.post_incoming(BT_task(self.incoming_data, ""))

    def send_data(self, data=None):
        """
        Sends data via bluetooth to connected client.
        Sends data saved in server, unless data is passed.
        """
        if not data:
            self.client_sock.send(self.outgoing_data)
        else:
            self.client_sock.send(data)

    def _pop_from_outgoing(self):
        return bt_task_handler.pop_outgoing()

    def update_incoming(self):
        """
        Updates incoming data aimed for the robot by receiving via bluetooth.
        Saves the new data in itself.
        Returns True/False whether or not new data was received.
        """
        has_new_incoming = False
        try:
            self.client_sock.settimeout(0.1)
            data = self.client_sock.recv(1024).decode('utf-8')
            if len(data) != 0:  # TODO or None? (using json)
                self.incoming_data = data
                has_new_incoming = True
        except bluetooth.btcommon.BluetoothError:
            pass
        finally:
            self.client_sock.settimeout(None)
        return has_new_incoming

    def update_outgoing(self):
        """
        Updates outgoing_data aimed for client by poping from robots queue.
        Saves the new data in itself.
        Returns true if data was updated, false otherwise
        """
        has_new_outgoing = False
        task = self._pop_from_outgoing()
        if type(task) == BT_task and task.cmd_id != 0:
            self.outgoing_data = str(task.cmd_id) + ", " + str(task.data)
            has_new_outgoing = True
        return has_new_outgoing

    def shutdown_server(self):
        """
        Shuts down itself by closing server and client sockets.
        """
        self.server_sock.shutdown(2)
        self.server_sock.close()
        self.client_sock.close()
        print("Closed connection.")
```

```
################################################################################
#                                                                              #
#                              bt_server_cmds.py                               #
#                                                                              #
#                                 Version 1.0                                  #
#                          Senast modifierad 2016-11-28                        #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

import os

"""
File for miscellaneous functions.
Functions can be moved if no need of file otherwise.
"""


def get_pi_ip():
    """
    Returns IP address for Rasberry Pi connected to Eduroam.
    """
    s = os.popen('ifconfig wlan0 | grep "inet\ addr" | cut -d: -f2 | cut -d" " -
    f1')
    pi_ip = s.read()
    return pi_ip
```

```
################################################################################
#                                                                              #
#                             bt_server_runner.py                              #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-12                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

import bt_server
import bt_task_handler
import protocol

PI_ADDR = "B8:27:EB:FC:55:27"
PORT = 3
BACKLOG = 1
GOT_DATA = 1
NO_DATA = 0


def setup_server():
    """
    Creates and returns a fresh bt_server connected to a client.
    """
    bt_task_handler.clean_queue_files()
    server = bt_server.BT_Server(PI_ADDR, PORT, BACKLOG)
    server.accept_connection()

    print("Server connected.")
    return server


def is_valid_cmd(command):
    """
    Checks if given command is a valid command from a client.
    Returns True/False.
    """
    return True if command in protocol.BT_CLIENT_COMMANDS else False


def send(server):
    """
    Sends data to client if given bt_server has new outgoing
    data from main unit.
    Returns True/False whether or not data was sent.
    """
    has_new_outgoing = server.update_outgoing()
    if (has_new_outgoing):
        server.send_data()
    return has_new_outgoing


def recieve(server):
    """
    Post incoming data to queue to main unit if given bt_server
    has new data.
    Returns NO_DATA, NEW_DATA, SHUTDOWN or RESTART depending on
    data from client.
    """
    has_new_incoming = server.update_incoming()

    if has_new_incoming:
        if int(server.incoming_data) == protocol.BT_SERVER_RESTART:
            return protocol.BT_SERVER_RESTART
        elif int(server.incoming_data) == protocol.BT_SERVER_SHUTDOWN:
            return protocol.BT_SERVER_SHUTDOWN
        else:
            server.post_to_incoming()
            return GOT_DATA
    return NO_DATA


def main():
    """
    The main function initializes instance of the server.
    Runs the main control of data flow in the bluetooth connection.
    """
    server = setup_server()
    exit = NO_DATA
    while exit != protocol.BT_SERVER_SHUTDOWN:
        exit = recieve(server)
        send(server)
        if exit == protocol.BT_SERVER_RESTART or exit == protocol.BT_SERVER_SHUTDOWN:
            has_sent = False
            while has_sent:
                has_sent = send(server)

            server.shutdown_server()
            del server

            if exit == protocol.BT_SERVER_RESTART:
                server = setup_server()
                exit = NO_DATA
                # Breaks if exit == SHUTDOWN


main()
```

```
################################################################################
#                                                                              #
#                                  bt_task.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-28                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                                                              #
################################################################################

class BT_task:
    """
    Class for packing command ID and data when sending
    commands and corresponding answers over Bluetooth.
    """

    def __init__(self, cmd_id=0, data=0):
        self.cmd_id = int(cmd_id)
        self.data = data
```

```
################################################################################
#                                                                              #
#                              bt_task_handler.py                              #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-28                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

import pickle
from bt_task import BT_task


def clean_queue_files():
    """
    Creates clean files for queues between server and main unit.
    """
    answer_queue = open("bt_answers.txt", "w")
    answer_queue.seek(0)
    answer_queue.truncate()
    command_queue = open("bt_commands.txt", "w")
    command_queue.seek(0)
    command_queue.truncate()
    answer_queue.close()
    command_queue.close()


def post_outgoing(task):
    """
    Called from main unit.
    Posts BT_task processed by main unit to answer queue.
    """
    answer_queue = open("bt_answers.txt", "wb")
    pickle.dump(task, answer_queue)
    answer_queue.close()


def pop_incoming():
    """
    Called from main unit.
    Pops and returns next BT_task from commands queue,
    to be processed by the main unit.
    """
    command_queue = open("bt_commands.txt", "rb")
    task = None

    task_q = []
    while (True):
        try:
            task_i = pickle.load(command_queue)
            task_q.append(task_i)
        except EOFError:
            break
    if task_q:
        task = BT_task(task_q[0].cmd_id, task_q[0].data)
        del task_q[0]
        command_queue = open("bt_commands.txt", "wb")
        for task_i in task_q:
            pickle.dump(task_i, command_queue)

    command_queue.close()
    return task
```

```python


def post_incoming(task):
    """
    Called from server.
    Posts given BT_task to command queue,
    to be processed by main unit.
    """
    command_queue = open("bt_commands.txt", "wb")
    pickle.dump(task, command_queue)
    command_queue.close()


def pop_outgoing():
    """
    Called from server.
    Pops next outgoing BT_task from answer queue,
    tasks already processed by main unit.
    Returns popped task.
    """
    answer_queue = open("bt_answers.txt", "rb")
    task = None

    task_q = []
    while (True):
        try:
            task_i = pickle.load(answer_queue)
            task_q.append(task_i)
        except EOFError:
            break
    if task_q:
        task = BT_task(task_q[0].cmd_id, task_q[0].data)
        del task_q[0]
        answer_queue = open("bt_answers.txt", "wb")
        for task_i in task_q:
            pickle.dump(task_i, answer_queue)

    answer_queue.close()
    return task
```

```
################################################################################
#                                                                              #
#                                    bus.py                                    #
#                                                                              #
#                                 Version 1.0                                  #
#                          Senast modifierad 2016-11-16                        #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Wrapper for the I2C bus with added functionality to support the packet protocol
the robot is using.

Packet protocol
----
I2C supports two functions for interacting on the bus:
  Read address,
  Write address

There functions are in turn addressed to specific slaves and contain the
address requested/set and a value when writing data. Each function write or
read a single byte.

As the desired functionality requires data to be sent and received in
multi-byte chunks this functionality must be abstracted away using a looser
protocol. To accomplish this the read and written addresses are purposely used
incorrectly to specify different states of receiving or sending data.

The protocol use two "addresses":
  PACKET_HEADER,
  PACKET_DATA

Using these two addresses it is possible to send data of the length 255 bytes
by first sending the data length and then all bytes in the data array until
every byte has been sent.

PACKET_HEADER:
Contains the length of the following data on the range 0 to 255. If there is no
data to be sent the value returned is 0.

PACKET_DATA:
Contains the data of the n:th read byte since the PACKET_HEADER. Reading data
from a packet which has already been read to the end has undefined behaviour.

The master-slave problem
----
As requests to read and write data can only be made from the master unit there
are difficulties actually notifying of new data from a slave unit. The protocol
solves this by reading the "addresses" periodically in order to search for
pending data. When data read from PACKET_HEADER is not zero there is data
available and the program can then read its data.
"""

from busprovider import WIRED_BUS
from protocol import PACKET_HEADER, PACKET_DATA


class Bus:
    def __init__(self, interface=1):
        self.interface = interface
        self.bus = WIRED_BUS

    def send(self, data, unit_addr):
```

```python
        self._write_packet_start(len(data), unit_addr)
        self._write_packet_data(data, unit_addr)

    def try_receive(self, unit_addr):
        size = self._get_pending_packet_size(unit_addr)

        if size == 0:
            # No pending packet
            return None

        return self._read_packet_data(size, unit_addr)

    # Internal methods

    def _write_packet_start(self, packet_len, unit_addr):
        self.bus.write_byte_data(unit_addr, PACKET_HEADER, packet_len)

    def _write_packet_data(self, packet_data, unit_addr):
        for b in packet_data:
            self.bus.write_byte_data(unit_addr, PACKET_DATA, b)

    def _read_packet_data(self, packet_len, unit_addr):
        data = []
        for i in range(packet_len):
            data.append(self.bus.read_byte_data(unit_addr, PACKET_DATA))

        return data

    def _get_pending_packet_size(self, addr):
        return self.bus.read_byte_data(addr, PACKET_HEADER)
```

```
################################################################################
#                                                                              #
#                                busprovider.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-16                         #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from emulated_bus import EmulatedBus


def initialize_with_hardwarebus():
    try:
        import smbus
        return smbus.SMBus(1)

    except ImportError:
        return initialize_with_emulated_bus()


def initialize_with_emulated_bus():
    return EmulatedBus()


WIRED_BUS = initialize_with_hardwarebus()
```

```
################################################################################
#                                                                              #
#                            command_processors.py                             #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-01                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Contains various pre-notify command processors used to parse received data into
more usable form than single bytes.
"""
from protocol import CMD_RETURN_SENSOR_DATA
from utils import twos_comp

COMMAND_PROCESSORS = {}


def process_arguments(message_id, arguments):
    if message_id in COMMAND_PROCESSORS:
        return COMMAND_PROCESSORS[message_id](*arguments)

    return arguments


# Sensor data contains some arguments stored in 16-bit two's complement which
# must be parsed into it's corresponding python values.

def process_sensor_data(left_ir_mm_hi, left_ir_mm_lo,
                        right_ir_mm_hi, right_ir_mm_lo,
                        right_ir_back_mm_hi, right_ir_back_mm_lo,
                        left_ir_back_mm_hi, left_ir_back_mm_lo, *args):
    left_ir_mm = twos_comp((left_ir_mm_hi << 8) | left_ir_mm_lo, 16)
    right_ir_mm = twos_comp((right_ir_mm_hi << 8) | right_ir_mm_lo, 16)
    right_ir_back_mm = twos_comp((right_ir_back_mm_hi << 8) |
    right_ir_back_mm_lo, 16)
    left_ir_back_mm = twos_comp((left_ir_back_mm_hi << 8) | left_ir_back_mm_lo,
    16)

    return [left_ir_mm, right_ir_mm, right_ir_back_mm, left_ir_back_mm] + list(
    args)


COMMAND_PROCESSORS[CMD_RETURN_SENSOR_DATA] = process_sensor_data
```

```
################################################################################
#                                                                              #
#                               communicator.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-06                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                                                              #
################################################################################

import outbound

from utils import get_ip


class Communicator:
    def __init__(self, ir, laser, gyro, driver, navigator, position):
        self.ir = ir
        self.laser = laser
        self.gyro = gyro
        self.driver = driver
        self.navigator = navigator
        self.position = position

    '''
    Sends the sensor data as a string of integers on the format
    "ir_left, ir_left_back, ir_right, ir_right_back, laser, gyro"
    '''
    def send_sensor_data(self):
        outbound.bt_return_sensor_data(str([self.ir.get_ir_left()] +
                                       [self.ir.get_ir_left_back()] +
                                       [self.ir.get_ir_right()] +
                                       [self.ir.get_ir_right_back()] +
                                       [self.laser.get_data()] +
                                       [self.gyro.get_data()]))

    '''
    Sends the sensor data as a string of integers on the format
    "left_speed, right_speed"
    '''
    def send_servo_data(self):
        outbound.bt_return_servo_data(str([self.driver.get_left_speed()] +
                                      [self.driver.get_right_speed()]))

    '''
    Sends the map data as a list with tuples of integers corresponding to corner coordinates on the format
    "[(X1, Y1), (X2, Y2), ... , (Xn, Yn)]"
    '''
    def send_map_data(self):
        map_data = self.position.get_map_data()
        outbound.bt_return_map_data(str(self.position.get_map_data()))

    def send_ip(self):
        outbound.bt_return_ip([get_ip()])
        print("send_ip, done")

    def drive_forward(self):
        self.navigator.drive_forward()
        print("Communicator drove forward!")

    def drive_backward(self):
        self.navigator.drive_backward()

    def drive_forward_right(self):
        self.navigator.drive_forward_right()

    def drive_forward_left(self):
        self.navigator.drive_forward_left()

    def turn_left(self):
        self.navigator.turn_left()

    def turn_right(self):
        self.navigator.turn_right()
```

```
################################################################################
#                                                                              #
#                                  driver.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-13                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                               Rebecca Lindblom                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Methods for controlling the wheels
"""

from datetime import datetime, timedelta

import math

import autocontroller
from outbound import set_motor_speed

###### METHODS FOR CONTROLLING THE WHEELS #######
# Tasks should be reversed since we pop them from the list

STANDARD_SPEED = 50
FAST_SPEED = 70
SLOW_SPEED = 40

TURN_SPEED = 40
TURN_TIME = 900
TURN_DEGREES = 80
POST_TURN_TIME = 700
PRE_TURN_TIME = 500
WARMUP_TIME = 2000
POST_TURN_DISTANCE = 200
PRE_TURN_DISTANCE = 200
REMOTE_COMMAND_EXECUTE_TIME = 525

"""
Thoughts on post_turn distance:

It appears that after a turn (at least on the track created yesterday) the laser will set a
destination at around 900 in start, but then as soon as it starts running distance_task
the laser values jumps up to 1300, and therefore it will travel too far before begining
to auto control. Not sure if it reads the laser value too soon, as in before it has finished
turning and instead reads a value on a wall to the side instead of the opposite wall, or if
it's something else. But that apprears to be the problem, and only when the distance is quite
far, >1m or so.
"""


class Driver:
    def __init__(self, gyro, laser):
        self.drive_stop_time = 0
        self.tasks = []
        self.task = Task(None, lambda: True)
        self.gyro = gyro
        self.laser = laser
        self.right_speed = 0
        self.left_speed = 0

    #Returns true only if we have executed all provided tasks
    def idle(self):
        return self.task.done() and not self.tasks

    #Expected to be run every main loop. Checks whether the current task is
    #done, and if there are more to perform, starts performing them.
    def update(self):
        if self.task.done():
            if self.tasks:
                self.task = self.tasks.pop()
                print("Next task: " + str(self.task))
                self.task.start()
            else:
                #print("STANNA")
                self.stop()

    def drive(self, left_speed, right_speed):
        self.left_speed = left_speed
        self.right_speed = right_speed
        set_motor_speed(left_speed, right_speed)
        # print("Driver drive set motor speed to ", left_speed, right_speed)

    def outer_turn_right(self):
        print('outer turn right')

        current_degree = math.degrees(math.atan(autocontroller.last_valid_diff / 165))
        degree = TURN_DEGREES - current_degree

        # We won't need the value of last_valid_diff any longer, so reset it to
        # avoid rotating too far or too little in dead ends (not needed here)
        print('Last valid diff:', autocontroller.last_valid_diff)
        autocontroller.last_valid_diff = 0

        print('Current degree:', current_degree)
        print('Turning degree:', degree)

        self.task = Task(None, lambda: True)
        self.tasks = [DegreeTask(self._turn_right, degree, self.gyro)]

    def outer_turn_left(self):
        print('outer turn left')
        self.task = Task(None, lambda: True)
        self.tasks = [DistanceTask(self._post_turn, POST_TURN_DISTANCE, self.laser),
                      DegreeTask(self._turn_left, TURN_DEGREES, self.gyro),
                      DistanceTask(self._pre_turn, PRE_TURN_DISTANCE, self.laser)]

    def inner_turn_left(self):
        print('inner turn left')
        current_degree = math.degrees(math.atan(autocontroller.last_diff / 165))

        # Over turn for dead ends
        if autocontroller.last_diff == 0:
            current_degree = 2

        degree = TURN_DEGREES + current_degree

        # We won’t need the value of last_diff any longer, so reset it to avoid
        # rotating too far or too little in dead ends
        autocontroller.last_diff = 0

        print(’Current degree:’, current_degree)
        print(’Turning degree:’, degree)

        self.task = Task(None, lambda: True)
        self.tasks = [DegreeTask(self._turn_left, degree, self.gyro)]

    def inner_turn_right(self):
        print(’inner turn right’)
        self.task = Task(None, lambda: True)
        self.tasks = [DegreeTask(self._turn_right, TURN_DEGREES, self.gyro)]

    def warmup(self):
        print(’warming up’)
        self.tasks = [TimedTask(lambda: self.drive(0, 0), WARMUP_TIME)]

    def stop(self):
        #print(’stopping’)
        self.drive(0, 0)

    def get_right_speed(self):
        return self.right_speed

    def get_left_speed(self):
        return self.left_speed


    # Commands intended to be called while remote controlling

    def drive_forward(self):
        self.task = TimedTask(self._drive_forward, REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    def drive_backward(self):
        self.task = TimedTask(self._drive_backward, REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    def turn_left(self):
        self.task = TimedTask(self._turn_left, REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    def turn_right(self):
        self.task = TimedTask(self._turn_right, REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    def drive_forward_right(self):
        self.task = TimedTask(self._drive_forward_right,
    REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    def drive_forward_left(self):
        self.task = TimedTask(self._drive_forward_left,
    REMOTE_COMMAND_EXECUTE_TIME)
        self.task.start()

    # Not intended for public use

    def _turn_left(self):
        print(’turn left’)
        self.drive(-TURN_SPEED, TURN_SPEED)

    def _turn_right(self):
        print(’turn right’)
        self.drive(TURN_SPEED, -TURN_SPEED)

    def _drive_forward(self):
        print(’drive forward’)
        self.drive(STANDARD_SPEED, STANDARD_SPEED)

    def _drive_backward(self):
        print(’drive backward’)
        self.drive(-STANDARD_SPEED, -STANDARD_SPEED)

    def _drive_forward_right(self):
        print(’drive forward right’)
        self.drive(FAST_SPEED, SLOW_SPEED)

    def _drive_forward_left(self):
        print(’drive backward left’)
        self.drive(SLOW_SPEED, FAST_SPEED)

    def _post_turn(self):
        print(’post turn’)
        self.drive(STANDARD_SPEED, STANDARD_SPEED)

    def _pre_turn(self):
        print(’pre turn’)
        self.drive(STANDARD_SPEED, STANDARD_SPEED)


class Task:
    def __init__(self, task_func, done_func):
        self.task_func = task_func
        self.done_func = done_func

    def start(self):
        self.task_func()

    def done(self):
        return self.done_func()


class TimedTask(Task):
    def __init__(self, task_func, duration):
        Task.__init__(self, task_func, self.timed_task)
        self.duration = duration
        self.stop_time = datetime.now()

    def start(self):
        Task.start(self)
        self.stop_time = datetime.now() + timedelta(milliseconds=self.duration)
        print("Start timed task")

    def timed_task(self):
        #print(’time left driving: ’ + str(self.stop_time - datetime.now()))
        return self.stop_time <= datetime.now()


class DegreeTask(Task):
    def __init__(self, task_func, degrees, gyro):
        Task.__init__(self, task_func, self.degree_task)
        self.total_degrees = 0
        self.previous_time = datetime.now()
        self.degrees = degrees
        self.gyro = gyro

    def start(self):
        self.total_degrees = 0
        self.previous_time = datetime.now()
        Task.start(self)


    def degree_task(self):
        data = self.gyro.get_data()

        if data == -1:
            raise Exception(’Error reading gyro’)

        time_delta = (self.previous_time - datetime.now()).total_seconds()
        delta_degrees = data * time_delta
        #print("time_delta: " + str(time_delta))
        #print("delta_degrees: " + str(delta_degrees))
        self.previous_time = datetime.now()
        self.total_degrees += delta_degrees
        #print(’total degrees turned :’ + str(self.total_degrees))

        return abs(self.total_degrees) >= self.degrees


class DistanceTask(Task):
    def __init__(self, task_func, distance, laser):
        Task.__init__(self, task_func, self.distance_task)
        self.destination = 0
        self.previous_time = datetime.now()
        self.distance = distance
        self.laser = laser

    def start(self):
        laser_data = -1
        while laser_data == -1:
            laser_data = self.laser.get_data()
            print("RUN RUN RUN LASER READINGS")

        #print("Distance: " + str(self.distance))
        #print("Laser data: " + str(laser_data))

        self.destination = laser_data - self.distance
        #print("Destination: " + str(self.destination))
        self.previous_time = datetime.now()
        Task.start(self)

    def distance_task(self):
        laser_data = -1
        while laser_data == -1:
            laser_data = self.laser.get_data()

        return self.destination >= laser_data
```

```
################################################################################
#                                                                              #
#                               emulated_bus.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-17                         #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

import logging
from queue import Queue

from protocol import CMD_REQUEST_SENSOR_DATA, CMD_RETURN_SENSOR_DATA, STYR_ADDR, \
    SENSOR_ADDR, LASER_ADDR, PACKET_HEADER, PACKET_DATA

log = logging.getLogger(__name__)


class EmulatedBus:
    def __init__(self):
        self.slaves = {
            SENSOR_ADDR: EmulatedSlave(SENSOR_ADDR),
            STYR_ADDR: EmulatedSlave(STYR_ADDR),
            LASER_ADDR: EmulatedLaser
        }

    def write_byte_data(self, address, data_id, data):
        if address in self.slaves:
            self.slaves[address].write_byte_data(data_id, data)
        else:
            log.error('Trying to send data to nonexistant slave.')

    def read_byte_data(self, address, data_id):
        if address in self.slaves:
            return self.slaves[address].read_byte_data(data_id)
        else:
            log.error(
                'Trying to read data from nonexistant slave (address = {}).'
                .format(address)
            )
            return None


class EmulatedSlave:
    def __init__(self, address):
        self.address = address
        self.data_queue = Queue()

        # Receiving
        self.read_packet = None
        self.bytes_to_read = 0
        self.current_read_byte = 0

        # Transmitting
        self.transmitted_packet = None
        self.current_transmitted_byte = 0

    def write_byte_data(self, event_type, data):
        if event_type == PACKET_HEADER:
            if self.read_packet is not None:
                log.error(
                    'Data header incorrectly sent to address {} '
                    'which still waits for more bytes'
                    .format(self.address)
                )

            self.read_packet = []
            self.bytes_to_read = data
            self.current_read_byte = 0
        elif event_type == PACKET_DATA:
            if self.read_packet is None:
                log.error(
                    'Data incorrectly sent to address {} '
                    'without preceding packet header'
                    .format(self.address)
                )

            self.read_packet.append(data)
            self.current_read_byte += 1

            if self.current_read_byte >= self.bytes_to_read:
                if self.read_packet[0] == CMD_REQUEST_SENSOR_DATA:
                    self.data_queue.put([CMD_RETURN_SENSOR_DATA, 0, 0, 0, 0])

                self.read_packet = None

    def read_byte_data(self, data_id):
        if data_id == PACKET_HEADER:
            if self.transmitted_packet is not None:
                log.error(
                    'Data header incorrectly requested from address {} '
                    'which still sends bytes'
                    .format(self.address)
                )

            if self.data_queue.empty():
                return 0

            self.transmitted_packet = self.data_queue.get()
            self.current_transmitted_byte = 0

            return len(self.transmitted_packet)
        elif data_id == PACKET_DATA:
            if self.transmitted_packet is None:
                log.error(
                    'Data incorrectly requested from address {} '
                    'without preceding packet header'
                    .format(self.address)
                )

            data = self.transmitted_packet[self.current_transmitted_byte]
            self.current_transmitted_byte += 1

            if self.current_transmitted_byte >= len(self.transmitted_packet):
                self.transmitted_packet = None

            return data


class EmulatedLaser:
    def write_byte_data(self, *args):
        pass

    def read_byte_data(self, *args):
        return 0
```

```
################################################################################
#                                                                              #
#                                   event.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-16                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                                                              #
################################################################################

"""
Container used to realize a command/event on the event bus and provides
functions for formatting the command as packet data.

See protocol.py for actual commands and their arguments.
"""
from command_processors import process_arguments

# Indexes of various data in received packets
ID_INDEX = 0
ARG_START_INDEX = 1


class Event:
    def __init__(self, message_id, arguments=None):
        self.message_id = message_id
        self.arguments = arguments or []

    @staticmethod
    def parse(data):
        return Event(data[ID_INDEX], data[ARG_START_INDEX:])

    def process(self):
        self.arguments = process_arguments(self.message_id, self.arguments)

    def as_packet_data(self):
        return [self.message_id] + self.arguments
```

```python
################################################################################
#                                                                              #
#                                 eventbus.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-28                         #
#                                                                              #
#                               Rebecca Lindblom                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Distributed event bus which is shared between all units on the main bus and via
Bluetooth.

The event bus provides a way to send data back and forth between different
units on the I2C bus and via Bluetooth by applying asynchronous transmission of
all events, it is therefore not guaranteed that messages are received on the
other end. As not all commands must be subscribed to it is also not certain
that the receiving unit actually reacts on the commands it receive.

In order for the event bus to function both ways the bus must be manually
polled for incoming messages by calling EventBus.receive(). This will read
pending commands from all connected AVR units and then call their respective
handlers if the command has been subscribed to.

Supported commands and their arguments are defined in protocol.py.
"""

from bus import Bus
from event import Event
from observer import Observer
import bt_task_handler

from protocol import SENSOR_ADDR, STYR_ADDR, BLUETOOTH_ADDR

# As reading from the bus is a blocking operation it might cause actual program
# code to execute too late if there are many pending commands available. In
# order to prevent the read operation to consume too much time the amount of
# messages read each iteration is limited.

MAX_READ_COUNT = 10


class EventBus:
    bus = Bus()
    observers = {}

    @staticmethod
    def post(addr, message):
        if addr == BLUETOOTH_ADDR:
            bt_task_handler.post_outgoing(message)
        else:
            EventBus.bus.send(message.as_packet_data(), addr)

    @staticmethod
    def pop(unit_addr):
        if unit_addr == BLUETOOTH_ADDR:
            return bt_task_handler.pop_incoming()
        else:
            return EventBus.bus.try_receive(unit_addr)

    @staticmethod
    def receive():
        EventBus.receive_from_addr(BLUETOOTH_ADDR)
        EventBus.receive_from_addr(SENSOR_ADDR)
        EventBus.receive_from_addr(STYR_ADDR)

    @staticmethod
    def receive_from_addr(unit_addr):
        for i in range(MAX_READ_COUNT):
            data = EventBus.pop(unit_addr)
            if data is None:
                break

            if unit_addr == BLUETOOTH_ADDR:
                EventBus.notify(data.cmd_id, *data.data)
            else:
                command = Event.parse(data)
                command.process()

                EventBus.notify(command.message_id, *command.arguments)

    @staticmethod
    def subscribe(command_id, handler):
        observer = EventBus._get_observer_for_command(command_id)
        observer.subscribe(handler)

    @staticmethod
    def notify(command_id, *args):
        observer = EventBus._get_observer_for_command(command_id)
        observer.notify(*args)

    @staticmethod
    def _get_observer_for_command(command_id):
        if command_id not in EventBus.observers:
            EventBus.observers[command_id] = Observer()

        return EventBus.observers[command_id]
```

```
################################################################################
#                                                                              #
#                                    gyro.py                                   #
#                                                                              #
#                                  Version 1.0                                 #
#                          Senast modifierad 2016-11-28                        #
#                                                                              #
#                                Sebastian Callh                               #
#                                 Anton Dalgren                                #
#                                                                              #
################################################################################

from time import sleep

from protocol import GYRO_ADDR
from eventbus import EventBus
from utils import twos_comp

GYRO_LOWER_LIMIT = 10


class Gyro:
    def __init__(self):
        self.data = 0


    def get_data(self):
        return self.data

    def read_data(self):
        try:
            hi = EventBus.bus.bus.read_byte_data(GYRO_ADDR, 0x2d)
            lo = EventBus.bus.bus.read_byte_data(GYRO_ADDR, 0x2c)
            data = (hi << 8) | lo

            # Divided by gyro sensitivity 18/256 for 2000 dps
            two_comp_data = 18 * twos_comp(data, 16) / 256

            # To prevent garbage values while standing still
            if abs(two_comp_data) <= GYRO_LOWER_LIMIT:
                self.data = 0

            self.data = two_comp_data

        except:
            self.data = -1

    @staticmethod
    def initialize():
        # Set the PD flag to 1 to go from power-down mode to normal mode
        EventBus.bus.bus.write_byte_data(GYRO_ADDR, 0x20, 0x0F)
        EventBus.bus.bus.write_byte_data(GYRO_ADDR, 0x23, 0x30)
```

```
################################################################################
#                                                                              #
#                                gyro_driver.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-17                         #
#                                                                              #
#                               Sebastian Callh                                #
#                                                                              #
################################################################################

from time import sleep
from datetime import datetime, timedelta

from gyro import Gyro
from driver import Driver

gyro = Gyro()
gyro.initialize()

previous_time = datetime.now()
degrees_total = 0

GYRO_LOWER_LIMIT = 10

while(True):
    data = gyro.read_data()

    time_delta = (previous_time - datetime.now()).total_seconds()
    delta_degrees =  data * time_delta
    previous_time = datetime.now()

    #Has to be run after previous time is set
    if abs(data) <= GYRO_LOWER_LIMIT:
        continue

    degrees_total += delta_degrees
    print(degrees_total)
```

```python
################################################################################
#                                                                              #
#                                    ir.py                                     #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-01                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                                                                              #
################################################################################

from datetime import datetime, timedelta
import outbound

class IR:
    def __init__(self):
        self.busy = False
        self.ir_left = 0
        self.ir_left_back = 0
        self.ir_right = 0
        self.ir_right_back = 0
        self.last_request = datetime.now()
        self.request_period = timedelta(milliseconds=1)

    def get_ir_left(self):
        return self.ir_left

    def get_ir_left_back(self):
        return self.ir_left_back

    def get_ir_right(self):
        return self.ir_right

    def get_ir_right_back(self):
        return self.ir_right_back

    def sensor_data_received(self, ir_left_mm, ir_right_mm, ir_right_back_mm, ir_left_back_mm):
        self.ir_left = ir_left_mm
        self.ir_right = ir_right_mm
        self.ir_right_back = ir_right_back_mm
        self.ir_left_back = ir_left_back_mm
        self.busy = False

    def request_data(self):
        if not self.busy and datetime.now() - self.last_request > self.request_period:
            self.busy = True
            self.last_request = datetime.now()
            outbound.request_sensor_data()
```

```
################################################################################
#                                                                              #
#                                   laser.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-19                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from eventbus import EventBus
from protocol import LASER_ADDR
from time import sleep

DEBUG_LASER = True

class Laser:
    DELTA_LIMIT = 100
    def __init__(self):
        self.data = 0
        self.last_data = 0
        self.last_last_data = 0

        if DEBUG_LASER:
            self.debug_file = open(’laser_measurements.dat’, ’w’)
        else:
            self.debug_file = None

    def get_data(self):
        if  abs(self.data - self.last_data) < Laser.DELTA_LIMIT and \
            abs(self.data - self.last_last_data) < Laser.DELTA_LIMIT:
            return self.data
        else:
            return -1

    def read_data(self):
        hi = 0
        lo = 0
        data = 0

        try:
            hi = EventBus.bus.bus.read_byte_data(LASER_ADDR, 0x0f)
            lo = EventBus.bus.bus.read_byte_data(LASER_ADDR, 0x10)
            data = (hi << 8) | lo
        except:
            print(’eeverything is exception’)
            self.data = -1

        if hi & 0x80 == 128 or (lo == 1 and hi == 0):
            print(’everything is terrible’)
            self.data = -1
        else:
            new_data = data * 10
            if new_data != self.data:
                self.last_last_data = self.last_data
                self.last_data = self.data
                self.data = new_data

        if self.debug_file is not None:
            self.debug_file.write(str(self.get_data()) + ’\n’)
            self.debug_file.flush()

    def reset(self):
        print('resetting laser (but actually not)')

    @staticmethod
    def initialize():
        # Was bus.write_byte_data, but that method was renamed/removed
        EventBus.bus.bus.write_byte_data(LASER_ADDR, 0x00, 0x00)  # Resets FPGA registers
        sleep(1)
        EventBus.bus.bus.write_byte_data(LASER_ADDR, 0x11, 0xff)  # sets laser to read forever
        EventBus.bus.bus.write_byte_data(LASER_ADDR, 0x00, 0x04)  # sets laser to start reading
```

```
################################################################################
#                                                                              #
#                                 lasertest.py                                 #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-28                         #
#                                                                              #
#                                Anton Dalgren                                 #
#                                                                              #
################################################################################

from laser import Laser

laser = Laser()

def setup():
    Laser.initialize()

setup()
while(True):
    laser.read_data()
    laser_distance = laser.get_data()
    if laser_distance >0 :
        print("Laser avstånd:"+str(laser_distance)+"\n")
    else:
        print("Error: \n")
```

```
##############################################################################
#                                                                            #
#                                   lcd.py                                   #
#                                                                            #
#                                 Version 1.0                                #
#                          Senast modifierad 2016-11-20                      #
#                                                                            #
#                               Sebastian Callh                              #
#                                                                            #
##############################################################################

import RPi.GPIO as GPIO
from time import sleep

'''
Anslutningar till GPIO-portar. Alla portar ar i board-numbering
GPIO  29, 31, 32, 33, 35, 36, 37, 38
GROUND  30
5V/VCC  02
E   40
A   02
K   06
'''


#Times in ns
E_CYCLE = 500
E_RISE_FALL = 20
E_PULSE_WIDTH = 230

DATA = [0 for x in range(0, 7)]


#29 is MSB, 38 LSB
PINS = [29, 31, 32, 33, 35, 36, 37, 38]
RS = 22
E = 40

FUNCTION_SET = [0, 0, 1, 1, 0, 0, 0, 0]
DISPLAY_ONOFF_CONTROL = [0, 0, 0, 0, 1, 0, 1, 1] #display, cursor, blink on
DISPLAY_CLEAR = [0, 0, 0, 0, 0, 0, 0, 1]
ENTRY_MODE_SET = [0, 0, 0, 0, 0, 1, 1, 0] #increment mode, entire shift off
DISPLAY_ON = [0, 0, 0, 0, 1, 1, 1, 1]
RESET_CURSOR = [0, 0, 0, 0, 0, 0, 1, 0]

BIT_PATTERN = {
    'A': [0,1,0,0,0,0,0,1],
    'B': [0,1,0,0,0,0,1,0],
    'C': [0,1,0,0,0,0,1,1],
    'D': [0,1,0,0,0,1,0,0],
    'E': [0,1,0,0,0,1,0,1],
    'F': [0,1,0,0,0,1,1,0],
    'G': [0,1,0,0,0,1,1,1],
    'H': [0,1,0,0,1,0,0,0],
    'I': [0,1,0,0,1,0,0,1],
    'J': [0,1,0,0,1,0,1,0],
    'K': [0,1,0,0,1,0,1,1],
    'L': [0,1,0,0,1,1,0,0],
    'M': [0,1,0,0,1,1,0,1],
    'N': [0,1,0,0,1,1,1,0],
    'O': [0,1,0,0,1,1,1,1],
    'P': [0,1,0,1,0,0,0,0],
    'Q': [0,1,0,1,0,0,0,1],
    'R': [0,1,0,1,0,0,1,0],
    'S': [0,1,0,1,0,0,1,1],
    ’T’: [0,1,0,1,0,1,0,0],
    ’U’: [0,1,0,1,0,1,0,1],
    ’V’: [0,1,0,1,0,1,1,0],
    ’W’: [0,1,0,1,0,1,1,1],
    ’X’: [0,1,0,1,1,0,0,0],
    ’Y’: [0,1,0,1,1,0,0,1],
    ’Z’: [0,1,0,1,1,0,1,0],
    ’0’: [0,0,1,1,0,0,0,0],
    ’1’: [0,0,1,1,0,0,0,1],
    ’2’: [0,0,1,1,0,0,1,0],
    ’3’: [0,0,1,1,0,0,1,1],
    ’4’: [0,0,1,1,0,1,0,0],
    ’5’: [0,0,1,1,0,1,0,1],
    ’6’: [0,0,1,1,0,1,1,0],
    ’7’: [0,0,1,1,0,1,1,1],
    ’8’: [0,0,1,1,1,0,0,0],
    ’9’: [0,0,1,1,1,0,0,1],
    ’.’: [0,0,1,0,1,1,1,0],
    ’ ’: [0,0,1,0,0,0,0,0]
}

class LCD:


    def initialize(self):
        #Use board numbering
        GPIO.setmode(GPIO.BOARD)

        #Configure output pins
        GPIO.setup(RS, GPIO.OUT)
        GPIO.setup(E, GPIO.OUT)
        GPIO.setup(PINS, GPIO.OUT)


        #Power up sequence
        GPIO.output(RS, 0)        #Select instruction register
        sleep_ms(100)       #Make sure at least 30 ms has passed since power on


        self._send(FUNCTION_SET)
        sleep_ms(10)

        self._send(FUNCTION_SET)
        sleep_us(200)

        self._send(FUNCTION_SET)
        sleep_ms(200)


        self._send(FUNCTION_SET)
        sleep_us(80)        #Make super sure that 39 us has passed
        self._send(DISPLAY_ONOFF_CONTROL)
        sleep_us(80)        #Make super sure that 39 us has passed again
        self._send(DISPLAY_CLEAR)
        sleep_ms(4)           #Make sure that 1.53 ms has passed
        self._send(ENTRY_MODE_SET)
        sleep_ms(10)

        #End of power up sequence. Display is off
        #Turn on display
        self._send(DISPLAY_ON)


    def clear(self):
        GPIO.output(RS, 0)
        self._send(DISPLAY_CLEAR)


    def reset_cursor(self):
        GPIO.output(RS, 0)
        self._send(RESET_CURSOR)


    def send(self, data):
        self.clear()
        self.reset_cursor()

        #ugly ugly
        self._send(self.bit_pattern(' '))
        self._send(self.bit_pattern(' '))
        self._send(self.bit_pattern(' '))

        GPIO.output(RS, 1)    #Select data register

        for d in data.replace('\n', ''):
            self._send(self.bit_pattern(d))


    def bit_pattern(self, char):
        return BIT_PATTERN[char.upper()]


    def _send(self, data):
        GPIO.output(E, 0)     #Make sure E is initially low

        #Put the data on the pins
        for d, p in zip(data, PINS):
            GPIO.output(p, d)

        #Write the data
        GPIO.output(E, 1)
        sleep_us(1)       #Larger than recommended wait
        GPIO.output(E, 0)
        sleep_us(1)       #Larger than recommended wait


    def cleanup(self):
        GPIO.cleanup()


#Sleep in ms
def sleep_ms(t):
    sleep(t / 1000)


#Sleep in us
def sleep_us(t):
    sleep(t / 10000)
```

```
################################################################################
#                                                                              #
#                                lcd_runner.py                                 #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-16                         #
#                                                                              #
#                               Sebastian Callh                                #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from time import sleep

from lcd import LCD

A = [0, 1, 0, 0, 0, 0, 0, 1]
ones = [1, 1, 1, 1, 1, 1, 1, 1]
zeroes = [0, 0, 0, 0, 0, 0, 0, 0]

try:
    lcd = LCD()
    lcd.init()
    while True:
        lcd.send(ones)
        sleep(2)
        lcd.send(zeroes)

    lcd.cleanup()
except:
    lcd.cleanup()
```

```
################################################################################
#                                                                              #
#                                   main.py                                    #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-18                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Main program code - where all the magic happens
"""

from datetime import datetime, timedelta

from navigator import Navigator
from driver import Driver
from laser import Laser
from gyro import Gyro
from ir import IR
from communicator import Communicator

from eventbus import EventBus
from position import Position

from protocol import *
from safety import Safety

ir = IR()
laser = Laser()
gyro = Gyro()
driver = Driver(gyro, laser)
navigator = Navigator(Navigator.MANUAL, ir, driver, laser)
position = Position(laser, ir, navigator)
communicator = Communicator(ir, laser, gyro, driver, navigator, position)


def setup():
    Safety.setup_terminal_abort()
    EventBus.subscribe(BT_REQUEST_SENSOR_DATA, communicator.send_sensor_data)
    EventBus.subscribe(BT_REQUEST_SERVO_DATA, communicator.send_servo_data)
    EventBus.subscribe(BT_REQUEST_MAP_DATA, communicator.send_map_data)
    EventBus.subscribe(BT_DRIVE_FORWARD, communicator.drive_forward)
    EventBus.subscribe(BT_DRIVE_BACK, communicator.drive_backward)
    EventBus.subscribe(BT_TURN_RIGHT, communicator.turn_right)
    EventBus.subscribe(BT_TURN_LEFT, communicator.turn_left)
    EventBus.subscribe(BT_DRIVE_FORWARD_RIGHT, communicator.drive_forward_right)
    EventBus.subscribe(BT_DRIVE_FORWARD_LEFT, communicator.drive_forward_left)
    EventBus.subscribe(AUTONOMOUS_MODE, (lambda: navigator.set_mode(Navigator.
    AUTONOMOUS)))
    EventBus.subscribe(MANUAL_MODE, (lambda: navigator.set_mode(Navigator.MANUAL
    )))
    EventBus.subscribe(CMD_TOGGLE_MODE, navigator.toggle_mode)
    EventBus.subscribe(REQUEST_PI_IP, communicator.send_ip)
    EventBus.subscribe(CMD_RETURN_SENSOR_DATA, ir.sensor_data_received)
    Laser.initialize()
    Gyro.initialize()

```

```

def main():
    setup()
    while True:
        laser.read_data()
        gyro.read_data()
        ir.request_data()

        EventBus.receive()
        position.update()
        navigator.navigate()

Safety.run_safely(main)
```

```
################################################################################
#                                                                              #
#                                 navigator.py                                 #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-18                         #
#                                                                              #
#                               Sebastian Callh                                #
#                              Matilda Dahlström                               #
#                                Anton Dalgren                                 #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
State machine for navigational states
"""
from datetime import datetime, timedelta

from autocontroller import AutoController
from eventbus import EventBus
from outbound import notify_mode_changed
from protocol import CMD_TURN_STARTED, CMD_TURN_FINISHED

TURN_DIRECTION_RIGHT = True
TURN_DIRECTION_LEFT = False
UPDATE_FREQUENCY = 50


class State:
    def run(self, data):
        assert 0, "run not implemented"


class AutoControl(State):
    auto_controller = AutoController()

    def is_at_right_turn(self, data):
        return data[’ir’].get_ir_right() == -1 and \
               data[’ir’].get_ir_right_back() == -1 and \
               Navigator.right_turn_enabled

    def run(self, data):
        if self.is_at_right_turn(data):
            data[’driver’].outer_turn_right()
            print(’NAVIGATOR: outer turn right’)
            return Turn(TURN_DIRECTION_RIGHT)

        laser_data = data[’laser’].get_data()

        if not Navigator.right_turn_enabled:
            Navigator.right_turn_enabled = (data[’ir’].get_ir_right() != -1 and data[’ir’].get_ir_right_back() != -1)

        # Inner turn
        if Navigator.force_left_turn or (laser_data <= Navigator.FACING_WALL_DIST and laser_data != -1 and (not Navigator.right_turn_enabled or data[’ir’].get_ir_right() != -1)):
            Navigator.force_left_turn = False
            if data[’side’] == Navigator.RIGHT_SIDE:
                data[’driver’].inner_turn_left()
                print(’NAVIGATOR: inner turn left’)
                return Turn(TURN_DIRECTION_LEFT)

        right_speed, left_speed, regulation = AutoControl.auto_controller.auto_control(data['ir'].get_ir_right(),
           data['ir'].get_ir_right_back(),
           data['side'])
        data['driver'].drive(left_speed, right_speed)

        return self


class Warmup(State):
    def run(self, data):
        if data['driver'].idle():
            print('NAVIGATOR: changin to auto control')
            return AutoControl()
        else:
            return self


class Turn(State):
    def __init__(self, is_right_turn):
        self.is_right_turn = is_right_turn

    def run(self, data):
        if Navigator.right_turn_enabled:
            Navigator.right_turn_enabled = False

        if data['driver'].idle():
            print('NAVIGATOR: changing to auto control')
            return Stabilize(self.is_right_turn)
        else:
            return self

class Stabilize(State):
    def __init__(self, is_right_turn):
        self.is_right_turn = is_right_turn
        self.angle_threshold = 2
        self.speed_scaling = 26

    def run(self, data):
        if self.is_right_turn or Navigator.force_left_turn:
            data['laser'].reset()
            return AutoControl()
        else:
            ir_front = data['ir'].get_ir_right()
            ir_back = data['ir'].get_ir_right_back()
            diff = ir_front - ir_back
            print("Stabilize diff:", diff)
            if abs(diff) < self.angle_threshold:
                data['laser'].reset()
                return AutoControl()

            turn_speed = int(abs(diff)/diff*self.speed_scaling)
            data['driver'].drive(turn_speed, -turn_speed)
            return self

###### NAVIGATOR CLASS #######
class Navigator:
    LEFT_SIDE = 0
    RIGHT_SIDE = 1

    MANUAL = 0
    AUTONOMOUS = 1

    DISCONTINUITY_DIST = 25.0  # mm
    FACING_WALL_DIST = 200  # mm

    right_turn_enabled = True
    force_left_turn = False

    def __init__(self, mode, ir, driver, laser):
        self.data = {
            'driver': driver,
            'laser': laser,
            'ir': ir,
            'side': Navigator.RIGHT_SIDE,
        }
        self.mode = mode
        self.state = Warmup()
        self.last_updated_time = datetime.now()

        # Stand still waiting for sensors
        self.data['driver'].warmup()

    # Runs the state. The states run method returns the next state
    def navigate(self):
        self.data['driver'].update()
        if self.mode == Navigator.AUTONOMOUS:
            next_state = self.state.run(self.data)

            curr_type = type(self.state)
            next_type = type(next_state)

            if curr_type is not Turn and next_type is Turn:
                EventBus.notify(CMD_TURN_STARTED)

            if curr_type is Stabilize and next_type is not Stabilize:
                EventBus.notify(CMD_TURN_FINISHED, self.state.is_right_turn)

            self.state = next_state

    def drive_forward(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].drive_forward()

    def drive_backward(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].drive_backward()

    def drive_forward_right(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].drive_forward_right()

    def drive_forward_left(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].drive_forward_left()

    def turn_left(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].turn_left()

    def turn_right(self):
        if self.mode == Navigator.MANUAL:
            self.data['driver'].turn_right()

    def set_mode(self, mode):
        self.mode = mode
        notify_mode_changed(mode)
    def toggle_mode(self):
        if self.mode == Navigator.MANUAL:
            self.data[’laser’].reset()
            self.set_mode(Navigator.AUTONOMOUS)
        else:
            self.set_mode(Navigator.MANUAL)
```

```
################################################################################
#                                                                              #
#                                 observer.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-11-12                         #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Simple implementation of the observer pattern.

See https://en.wikipedia.org/wiki/Observer_pattern for more information.
"""


class Observer:
    def __init__(self):
        self.subscribers = []

    def subscribe(self, func):
        if func not in self.subscribers:
            self.subscribers.append(func)

    def notify(self, *args, **kwargs):
        for subscriber in self.subscribers:
            subscriber(*args, **kwargs)
```

```python
################################################################################
#                                                                              #
#                                 outbound.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-12                         #
#                                                                              #
#                               Sebastian Callh                                #
#                                Anton Dalgren                                 #
#                               Rebecca Lindblom                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
This file contains functions for interacting with the two different AVR units.
All functions defined here are outbound which mean they go from the main unit
to one of the AVR units.

The message passing function is implemented as a distributed event bus which
in its distributed nature depends on asynchronous functionality. This means
that messages sent are not executed immediately and there is no guarantee that
the sent command is actually executed on the receiving unit.

For more information see eventbus.py.
"""

from event import Event
from eventbus import EventBus
from protocol import *
from bt_task import BT_task


# NOTE: Function comments are purposely left out from this file in favor of the
# complete definitions of every found command in proctol.py.

def request_sensor_data():
    EventBus.post(
        SENSOR_ADDR,
        Event(
            message_id=CMD_REQUEST_SENSOR_DATA
        )
    )


def set_motor_speed(left_speed, right_speed=None):
    if right_speed is None:
        right_speed = left_speed

    EventBus.post(
        STYR_ADDR,
        Event(
            message_id=CMD_SET_MOTOR_SPEED,
            arguments=[
                left_speed,
                right_speed
            ]
        )
    )


def set_left_motor_speed(speed):
    EventBus.post(
        STYR_ADDR,
        Event(
            message_id=CMD_SET_LEFT_MOTOR_SPEED,
            arguments=[
                speed
            ]
        )
    )


def set_right_motor_speed(speed):
    EventBus.post(
        STYR_ADDR,
        Event(
            message_id=CMD_SET_RIGHT_MOTOR_SPEED,
            arguments=[
                speed
            ]
        )
    )


def bt_return_ip(ip):
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            RETURN_PI_IP, ip
        )
    )


def bt_return_sensor_data(data):
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_RETURN_SENSOR_DATA, data
        )
    )


def bt_return_servo_data(data):
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_RETURN_SERVO_DATA, data
        )
    )


def bt_return_map_data(data):
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            BT_RETURN_MAP_DATA, data
        )
    )


def notify_mode_changed(new_mode):
    EventBus.post(
        BLUETOOTH_ADDR,
        BT_task(
            CMD_MODE_SET, new_mode
        )
    )
```

```
################################################################################
#                                                                              #
#                                 position.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                          Senast modifierad 2016-12-18                        #
#                                                                              #
#                               Sebastian Callh                                #
#                                Anton Dalgren                                 #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

import traceback

from eventbus import EventBus
from laser import Laser
from navigator import Navigator
from protocol import CMD_TURN_STARTED, CMD_TURN_FINISHED
from section import Section, NORTH, EAST, SOUTH, WEST, BLOCK_LENGTH_MM
from threading import Thread

STATE_MEASURING = 0
STATE_WAITING = 1
STATE_FINISHED = 2

MAPPING_STATE_FOLLOWING_OUTER_WALL = 0
MAPPING_STATE_RETURNING_TO_ISLAND = 1
MAPPING_STATE_FOLLOWING_ISLAND = 2
MAPPING_STATE_RETURNING_TO_GARAGE = 3


class Position:
    def __init__(self, laser, ir, navigator):
        self.laser = laser
        self.ir = ir
        self.navigator = navigator
        self.state = STATE_MEASURING
        self.mapping_state = MAPPING_STATE_FOLLOWING_OUTER_WALL
        self.current_section = Section(NORTH)
        self.saved_sections = []
        self.map_data = [(0, 0)]
        self.current_x = 0
        self.current_y = 0
        self.kitchen_start_x = 0
        self.kitchen_start_y = 0
        self.kitchen_num_mapped = 0
        self.num_kitchen_turns = 0

        self.kitchen_mapping = {}
        self.invalid_kitchens = []
        self.island_data = []
        self.edges_data = []

        self.num_ok_close = 0
        self.num_ok_far = 0

        EventBus.subscribe(CMD_TURN_STARTED, self.on_turning_started)
        EventBus.subscribe(CMD_TURN_FINISHED, self.on_turning_finished)

    def update(self):
        if self.state == STATE_MEASURING:
            distance = self.laser.get_data()
            self.current_section.add_distance_sample(distance)
            if self.mapping_state == MAPPING_STATE_FOLLOWING_OUTER_WALL:
                self.looking_for_kitchen_sections()
                self.remove_invalid_kitchen_sections()
            elif self.mapping_state == MAPPING_STATE_RETURNING_TO_ISLAND:
                self.check_if_returned_to_island()
            elif self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND:
                self.current_section.finish()
                temporary_x, temporary_y = self.transform_map_data(self.current_section, self.current_x, self.current_y, estimate=True, offset=0.1)
                if temporary_x == self.kitchen_start_x and temporary_y == self.kitchen_start_y \
                        and self.kitchen_num_mapped > 3:
                    self.map_island_data()
                    self.map_remaining_data()

                    print(self.map_data)
                    print(self.island_data)

                    Navigator.force_left_turn = True

                    # Last island section is not properly stored because of a
                    # turn so we must store it here
                    self.store_ongoing_section()

                    self.mapping_state = MAPPING_STATE_RETURNING_TO_GARAGE

    def looking_for_kitchen_sections(self):
        if self.has_close_kitchen_left():
            self.add_kitchen_mapping(1, 0.5)

        if self.has_far_kitchen_left():
            self.add_kitchen_mapping(2, 0.25)

    def has_close_kitchen_left(self):
        match = self.ir.get_ir_left() > 10
        if match:
            self.num_ok_close += 1
        else:
            self.num_ok_close = 0

        if self.num_ok_close > 7:
            self.num_ok_close = 0
            return True
        else:
            return False

    def has_far_kitchen_left(self):
        match = 250 < self.ir.get_ir_left_back() < 650
        if match:
            self.num_ok_far += 1
        else:
            self.num_ok_far = 0

        if self.num_ok_far > 7:
            self.num_ok_far = 0
            return True
        else:
            return False

    def add_kitchen_mapping(self, displacement, offset):
        distance = self.current_section.estimate_block_distance(offset)
        cur_x, cur_y = self.transform_partial_map_data(distance, self.current_section.direction, self.current_x, self.current_y)
        key = self.get_left_block_coordinates(cur_x, cur_y, displacement)

```

```python
        if key not in self.kitchen_mapping and key not in self.invalid_kitchens:
            print('Adding new mapping (' + str(key[0]) + ', ' + str(key[1]) + ') -> (' + str(cur_x) + ', ' + str(cur_y) + ')')
            self.kitchen_mapping[key] = (cur_x, cur_y)

    def remove_invalid_kitchen_sections(self):
        temporary_x, temporary_y = self.transform_map_data(self.current_section,
                                                           self.current_x,
                                                           self.current_y)
        key = self.get_right_block_coordinates(temporary_x, temporary_y, 1)
        popped = self.kitchen_mapping.pop(key, None)
        if popped is not None:
            print('Removed kitchen mapping for block (' + str(key[0]) + ', ' + str(key[1]) + ')')

        if key not in self.invalid_kitchens:
            print('Forbidding kitchen block at ' + str(key))
            self.invalid_kitchens.append(key)

    def check_if_returned_to_island(self):
        coordinates = self.transform_map_data(self.current_section,
                                              self.current_x,
                                              self.current_y,
                                              estimate=True)

        if coordinates in self.kitchen_mapping.values():
            print('Starting to map island!')

            self.mapping_state = MAPPING_STATE_FOLLOWING_ISLAND
            self.kitchen_num_mapped = 0
            self.num_kitchen_turns = 0
            key1 = self.transform_partial_map_data(1, self.current_section.direction, coordinates[0], coordinates[1])
            key2 = self.transform_partial_map_data(2, self.current_section.direction, coordinates[0], coordinates[1])
            if key1 not in self.kitchen_mapping and key2 not in self.kitchen_mapping:
                Navigator.force_left_turn = True
            else:
                Navigator.right_turn_enabled = False
                self.num_kitchen_turns = 1

    def map_island_data(self):
        if len(self.island_data) == 0:
            print('ERROR: No island to map!')
            return

        initial_island_data = list(self.island_data)
        self.island_data = []
        for pos in initial_island_data:
            self.go_deeper(pos[0], pos[1], self.island_data, self.map_data)

    def map_remaining_data(self):
        avoid_data = list(self.edges_data) + list(self.island_data)
        initial_map_data = list(self.island_data)
        self.map_data = []
        for pos in initial_map_data:
            self.go_deeper(pos[0], pos[1], self.map_data, avoid_data, skip_avoid_test=True)

        self.map_data += self.edges_data

    def go_deeper(self, x, y, lst, avoid_lst, skip_avoid_test=False):
        pos = (x, y)
        if pos not in avoid_lst or skip_avoid_test:
            if pos not in lst:
                if not skip_avoid_test:
                   lst.append(pos)

                try:
                    self.go_deeper(x + 1, y, lst, avoid_lst)
                    self.go_deeper(x - 1, y, lst, avoid_lst)
                    self.go_deeper(x, y + 1, lst, avoid_lst)
                    self.go_deeper(x, y - 1, lst, avoid_lst)
                except:
                    traceback.print_exc()
                    print('Your island or map is too big')

    def save_current_section(self):
        if self.mapping_state == MAPPING_STATE_FOLLOWING_OUTER_WALL:
            self.process_finished_section()

            print('---- SECTION SAVED ----')
            print('  direction: ' + str(self.current_section.direction))
            print('  distance: ' + str(self.current_section.block_distance))
            print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
            print('  kitchen mappings: \n' + self.get_kitchen_debug_data())
            print('-----------------------')
        elif self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND:
            self.process_finished_section()

            print('---- ISLAND ROUNDED ----')
            print('  direction: ' + str(self.current_section.direction))
            print('  distance: ' + str(self.current_section.block_distance))
            print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
            print('-----------------------')

            self.kitchen_num_mapped += 1
            self.num_kitchen_turns += 1
        else:
            self.process_finished_section(store_data=False)

            print('---- SECTION TRAVELLED ----')
            print('  direction: ' + str(self.current_section.direction))
            print('  distance: ' + str(self.current_section.block_distance))
            print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
            print('-----------------------')

    def get_kitchen_debug_data(self):
        data = []
        for k, v in self.kitchen_mapping.items():
            data.append('    ' + str(k) + ' -> ' + str(v))

        return '\n'.join(data)

    def begin_next_section(self, is_right_turn):
        if is_right_turn:
            self.current_section = self.current_section.for_right_turn()
        else:
            self.current_section = self.current_section.for_left_turn()

    def on_turning_started(self):
        if self.mapping_state == MAPPING_STATE_RETURNING_TO_GARAGE:
            if self.has_returned_to_start(include_direction=False):
                self.state = STATE_FINISHED
                self.navigator.set_mode(Navigator.MANUAL)
                print(self.map_data)
                print(self.island_data)
        else:
            self.state = STATE_WAITING

        self.save_current_section()

    def on_turning_finished(self, is_right_turn):
        self.begin_next_section(is_right_turn)
        if self.has_returned_to_start() or self.mapping_state == MAPPING_STATE_RETURNING_TO_ISLAND:
            self.mapping_state = MAPPING_STATE_RETURNING_TO_ISLAND
            self.island_data = list(self.kitchen_mapping.keys())
            self.edges_data = list(self.map_data)

        if self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND and self.num_kitchen_turns == 2:
            self.kitchen_start_x = self.current_x
            self.kitchen_start_y = self.current_y

        self.state = STATE_MEASURING

    def has_returned_to_start(self, include_direction=True):

        if self.current_x == 0 and self.current_y == 0 and len(self.map_data) > 0:
            return not include_direction or self.current_section.direction == NORTH
        else:
            return False

    def transform_map_data(self, section, current_x, current_y, estimate=False, offset=None):
        if estimate:
            if offset is None:
                distance = section.estimate_block_distance()
            else:
                distance = section.estimate_block_distance(offset)
        else:
            section.finish()
            distance = section.block_distance

        return self.transform_partial_map_data(distance, section.direction, current_x, current_y)

    def transform_partial_map_data(self, distance, direction, current_x, current_y):
        if direction == NORTH:
            current_y += distance
        elif direction == EAST:
            current_x += distance
        elif direction == SOUTH:
            current_y -= distance
        elif direction == WEST:
            current_x -= distance
        return current_x, current_y

    def store_ongoing_section(self):
        self.current_section.finish()

        distance = self.current_section.block_distance
        direction = self.current_section.direction

        for i in range(1, distance + 1):
            pos_x, pos_y = self.transform_partial_map_data(i, direction, self.current_x, self.current_y)
```

```
            self.map_data.append((pos_x, pos_y))

    def process_finished_section(self, store_data=True):
        self.current_section.finish(debug_limits=True)

        if store_data:
            distance = self.current_section.block_distance
            direction = self.current_section.direction

            for i in range(1, distance + 1):
                coordinates = self.transform_partial_map_data(i, direction, self.current_x, self.current_y)
                self.map_data.append(coordinates)
                self.kitchen_mapping.pop(coordinates, None)
                if coordinates not in self.invalid_kitchens:
                    self.invalid_kitchens.append(coordinates)

            self.saved_sections.append(self.current_section)

        # Could be optimized to use last pos_x, pos_y instead
        self.current_x, self.current_y = self.transform_map_data(self.current_section, self.current_x, self.current_y)

    '''
    Returns a list of tuples with coordinates of grids visited
    '''
    def get_map_data(self):
        return [self.map_data]

    '''
    Returns the robots last know x,y coordinates
    '''
    def get_current_position(self):
        return self.current_x, self.current_y

    def get_left_block_coordinates(self, x, y, displacement):
        if self.current_section.direction == NORTH:
            x -= displacement
        elif self.current_section.direction == EAST:
            y += displacement
        elif self.current_section.direction == SOUTH:
            x += displacement
        elif self.current_section.direction == WEST:
            y -= displacement

        return x, y

    def get_right_block_coordinates(self, x, y, displacement):
        return self.get_left_block_coordinates(x, y, -displacement)
```

```
################################################################################
#                                                                              #
#                                 protocol.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-17                         #
#                                                                              #
#                               Sebastian Callh                                #
#                                Anton Dalgren                                 #
#                               Rebecca Lindblom                               #
#                              Matildha Sjöstedt                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
The protocol for the robot consist of various commands, or events, passed along
the main bus using a distributed event bus. Each command is identified by its
command id and is transmitted before eventual arguments.

All commands may be sent in any direction but the implementations will probably
choose to ignore irrelevant one. For messages originating from the main unit,
see outbound.py.
"""

# Addresses for the units on the bus. Note that the laser cannot be queried
# using the protocol described in bus.py.


SENSOR_ADDR = 0x30
STYR_ADDR = 0x40
BLUETOOTH_ADDR = 0xBEEF
LASER_ADDR = 0x62
GYRO_ADDR = 0x6b
ACCEL_ADDR = 0x19

# Packet addresses
PACKET_HEADER = 0
PACKET_DATA = 1

# Request data from the sensor unit
CMD_REQUEST_SENSOR_DATA = 1
"""
Issues a request to the sensor unit prompting it to send its most recent sensor
data back to the main unit. As the data transfer is asynchronous the sensor
unit responds to the request by posting a CMD_RETURN_SENSOR_DATA command on the
bus.

Target: Sensor unit

Arguments: None
"""

# Return sensor data from the sensor unit
CMD_RETURN_SENSOR_DATA = 2
"""
Command sent from the sensor unit after a request for its sensor data has been
made. As data from various sensors are reported independently it is not certain
that the value from sensor A and value from sensor B reflect the world at the
same point in time.

Target: Main unit

Arguments:
ir_left_mm (2 bytes, two's complement)
```

```
    Distance recorded by the left IR sensor in mm, or -1 if the distance is not
    within the supported range.
ir_right_mm (2 bytes, two’s complement)
    Distance recorded by the right IR sensor in mm, or -1 if the distance is
    not within the supported range.
"""

# Ping a unit
CMD_PING = 3
"""
Dummy command which will only trigger the other unit to respond with a PONG
command. Preferably used to test a connection without forcing the other party
to perform any actual action.

Target: Any AVR unit

Arguments: None
"""

# Pong a unit
CMD_PONG = 4
"""
Reply to a PING command.

Target: Main unit

Arguments: None
"""

# Set both motor speeds in the control unit
CMD_SET_MOTOR_SPEED = 5
"""
Sets the speed of both the left and right motors on the robot. Values can
be both positive and negative and the range -100 to 100 is supported, where the
value represents a percentage of the max speed. It seems like the different
directions have various max speeds, which has to be considered when
implementing on-spot-rotation or other actions which assume equal speed forward
and backwards.

Target: Control unit

Arguments:
left_motor_speed (1 byte, positive or negative)
    Left motor speed in percentage of max speed ranging from -100 to 100.
right_motor_speed (1 byte, positive or negative)
    Right motor speed in percentage of max speed ranging from from -100 to 100.
"""

# Set left motor speed only in control unit
CMD_SET_LEFT_MOTOR_SPEED = 6
"""
Sets the speed of the left motors on the robot. It is only possible to pass
positive values with the command in the range 0 to 100, where the value
represent a percentage of the max speed.

Target: Control unit

Arguments:
speed (1 byte, positive)
    Left motor speed in percentage of max speed ranging from 0 to 100.
"""

# Set right motor speed in the control unit
CMD_SET_RIGHT_MOTOR_SPEED = 7
"""
```

```
Sets the speed of the right motors on the robot. It is only possible to pass
positive values with the command in the range 0 to 100, where the value
represent a percentage of the max speed.

Target: Control unit

Arguments:
speed (1 byte, positive)
    Right motor speed in percentage of max speed ranging from 0 to 100.
"""
# Indicates that the robot has started turning
"""
Event called internally within the main unit to indicate that a simple 90
degree turn has been initiated.

Target: Main unit

Arguments: None
"""
CMD_TURN_STARTED = 8

# Indicates that the robot has stopped turning
"""
Event called internally within the main unit to indicate that a simple 90
degree turn has finished.

Target: Main unit

Arguments:
is_right_turn (1 byte, boolean)
    True for right turn, false for left turn.
"""
CMD_TURN_FINISHED = 9

# -------------------- Bluetooth commands ---------------------

# Ask for IP address of robot
REQUEST_PI_IP = 10
"""
Command sent from Bluetooth client to Bluetooth server. Issues a request
to the to the server prompting it to send its IP address back to the client.

Target: Bluetooth server

Arguments: None
"""

# Return IP address from the robot
RETURN_PI_IP = 11
"""
Command sent from Bluetooth server to Bluetooth client after a request
for its IP address has been made.

Target: Bluetooth client

Arguments: ip
    Sent as a string.
"""

# Demand Bluetooth server to restart

BT_SERVER_RESTART = 12

BT_SERVER_SHUTDOWN = 13

```

```
BT_REQUEST_SENSOR_DATA = 14
BT_RETURN_SENSOR_DATA = 15

BT_REQUEST_SERVO_DATA = 16
BT_RETURN_SERVO_DATA = 17

BT_REQUEST_MAP_DATA = 18
BT_RETURN_MAP_DATA = 19

BT_DRIVE_FORWARD = 20
BT_DRIVE_BACK = 21

BT_TURN_RIGHT = 22
BT_TURN_LEFT = 23

BT_DRIVE_FORWARD_RIGHT = 24
BT_DRIVE_FORWARD_LEFT = 25

AUTONOMOUS_MODE = 26
MANUAL_MODE = 27

# Toggle between autonomous and manual mode
CMD_TOGGLE_MODE = 28
"""
Command to toggle between the available modes (autonomous and manual) instead
of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.
Target: Main unit
Arguments: None
"""

# Indicates that the robot has changed to a new navigator mode
CMD_MODE_SET = 29
"""
Command to toggle between the available modes (autonomous and manual) instead
of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.

Target: Main unit

Arguments:
new_mode (1 byte)
    Integer representation of the new mode:
      0 = Manual mode
      1 = Autonomous mode
"""

BT_CLIENT_COMMANDS = [REQUEST_PI_IP, BT_SERVER_RESTART,
                      BT_SERVER_SHUTDOWN, BT_REQUEST_SENSOR_DATA,
                      BT_REQUEST_MAP_DATA, BT_REQUEST_SERVO_DATA,
                      BT_DRIVE_FORWARD, BT_DRIVE_BACK,
                      BT_TURN_RIGHT, BT_TURN_LEFT, BT_DRIVE_FORWARD_RIGHT,
    BT_DRIVE_FORWARD_LEFT]

BT_SERVER_COMMANDS = [REQUEST_PI_IP, BT_RETURN_SENSOR_DATA,
                      BT_RETURN_SERVO_DATA, BT_RETURN_MAP_DATA, CMD_MODE_SET]
```

```
################################################################################
#                                                                              #
#                                  safety.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                          Senast modifierad 2016-11-16                        #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
As the robot will move autonomously it is crucial that it can be stopped both
fast and reliably in case of a failure. This class serves to mitigate possible
errors and provides means for externally aborting the operation of the robot.
"""
import signal

import sys
import traceback

from outbound import set_motor_speed


class Safety:
    @staticmethod
    def setup_terminal_abort():
        signal.signal(signal.SIGINT, Safety.handle_abort)

    @staticmethod
    def handle_abort(*args):
        # Stop motors to avoid robot running amok
        set_motor_speed(0)
        sys.exit(0)

    @staticmethod
    def run_safely(func):
        # Wrap entire program after the motors has been started in a try-catch
        # to avoid risking not being able to shut down the robot remotely in
        # case of a failure.
        try:
            func()
        except:
            set_motor_speed(0)
            traceback.print_exc()

            sys.exit(0)
```

```
################################################################################
#                                                                              #
#                                  section.py                                  #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                                Anton Dalgren                                 #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from datetime import datetime
from math import floor


NORTH = 0
EAST = 1
SOUTH = 2
WEST = 3

BLOCK_LENGTH_MM = 400


class Section:
    def __init__(self, direction):
        self.direction = direction
        self.measurements = []
        self.block_distance = None

    def add_distance_sample(self, distance):
        if 50 < distance < 6000:
            self.measurements.append((distance, datetime.now()))

    def finish(self, debug_limits=False):
        # If we're somehow detecting the next turn for a dead end corner before
        # any measurements have been received our normal algorithm won't work.
        # Handle this by explicitly setting the distance to zero.
        if len(self.measurements) == 0:
            self.block_distance = 0
            return

        # Takes the difference between max measurement and min measurement and divide by block length.
        self.block_distance = round((self.get_max(debug_limits) - self.get_min(debug_limits)) / BLOCK_LENGTH_MM)

    def estimate_block_distance(self, offset=0.25):
        if len(self.measurements) == 0:
            return 0

        # Takes the difference between max measurement and min measurement and divide by block length.
        return floor((self.get_max() - self.get_min()) / BLOCK_LENGTH_MM + offset)

    def for_right_turn(self):
        return Section((self.direction + 1) % 4)

    def for_left_turn(self):
        return Section((self.direction - 1) % 4)

    def get_max(self, debug_limits=False):
        if len(self.measurements) == 0:
            return 0
```

```python

        value = max(self.measurements, key=lambda x: x[0])[0]
        if debug_limits:
            print('Max', value)

        return value

    def get_min(self, debug_limits=False):
        if len(self.measurements) == 0:
            return 0

        value = min(self.measurements, key=lambda x: x[0])[0]
        if debug_limits:
            print('Min', value)

        return value
```

```
################################################################################
#                                                                              #
#                               test_mapping.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-15                         #
#                                                                              #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

from position import Position

pos = Position(None, None, None)
pos.map_data = [(0, 0), (0, 1), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4),
    (3, 5), (3, 6), (3, 7), (2, 7), (1, 7), (0, 7), (-1, 7), (-2, 7), (-2, 6),
    (-2, 5), (-3, 5), (-4, 5), (-5, 5), (-6, 5), (-5, 5), (-4, 5), (-4, 4), (-4,
     3), (-4, 2), (-4, 1), (-3, 1), (-2, 1), (-1, 1), (0, 1), (0, 0), (3, 2),
    (3, 3), (2, 3), (1, 3), (0, 3), (-1, 3), (-1, 4), (-1, 5), (-1, 6), (0, 6),
    (1, 6), (1, 5), (1, 6), (2, 6), (3, 6), (3, 5), (3, 4), (3, 3)]
pos.edges_data = [(0, 0), (0, 1), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4)
    , (3, 5), (3, 6), (3, 7), (2, 7), (1, 7), (0, 7), (-1, 7), (-2, 7), (-2, 6),
     (-2, 5), (-3, 5), (-4, 5), (-5, 5), (-6, 5), (-5, 5), (-4, 5), (-4, 4),
    (-4, 3), (-4, 2), (-4, 1), (-3, 1), (-2, 1), (-1, 1), (0, 1), (0, 0)]
pos.island_data = [(1, 5), (2, 3), (-1, 5), (0, 6), (0, 5), (2, 5), (2, 4)]

pos.map_island_data()
print(pos.island_data)

pos.map_remaining_data()
print(pos.map_data)
```

```
################################################################################
#                                                                              #
#                               test_sensors.py                                #
#                                                                              #
#                                 Version 1.0                                  #
#                         Senast modifierad 2016-12-01                         #
#                                                                              #
#                              Matilda Dahlström                               #
#                                                                              #
################################################################################

"""
Main program code - where all the magic happens
"""
from math import floor
from datetime import datetime, timedelta

from navigator import Navigator
from driver import Driver
from laser import Laser
from gyro import Gyro

from eventbus import EventBus
from outbound import request_sensor_data
from position import Position

from protocol import CMD_RETURN_SENSOR_DATA
from safety import Safety

laser = Laser()
gyro = Gyro()

# Update frequency
last_request = datetime.now()
request_period = timedelta(milliseconds=1)
busy = False

l_r = datetime.now()
r_p = timedelta(milliseconds=250)


def setup():
    Safety.setup_terminal_abort()
    EventBus.subscribe(CMD_RETURN_SENSOR_DATA, sensor_data_received)
    Laser.initialize()
    Gyro.initialize()


def sensor_data_received(ir_left_mm, ir_right_mm, ir_right_back_mm,
    ir_left_back_mm):
    global busy, navigator
    busy = False
    print("LF: " + str(ir_left_mm))
    print("RF: " + str(ir_right_mm))
    print("RBack: " + str(ir_right_back_mm))
    print("LBack: " + str(ir_left_back_mm))


def request_data():
    global busy, last_request, request_period
    if not busy and datetime.now() - last_request > request_period:
        busy = True
        last_request = datetime.now()

        # TODO: Uncomment below line when reading from laser
        laser.read_data()
        laser_distance = laser.get_data()
        print("Laser distance: " + str(laser_distance))
        request_sensor_data()


def main():
    global busy, last_request

    setup()

    while True:
        EventBus.receive()
        request_data()

Safety.run_safely(main)
```

```
################################################################################
#                                                                              #
#                                   utils.py                                   #
#                                                                              #
#                                 Version 1.0                                  #
#                          Senast modifierad 2016-11-30                        #
#                                                                              #
#                               Sebastian Callh                                #
#                               Rebecca Lindblom                               #
#                                Patrik Sletmo                                 #
#                                                                              #
################################################################################

"""
Various functions which does not belong in any other file.

NOTE: This file might have to be split if the list of functions grow too large.
"""
import os


# Returns the two's complementary value in it's python representation
def twos_comp(val, bits):
    if (val & (1 << (bits - 1))) != 0:
        val -= 1 << bits

    return val


def get_ip():
    s = os.popen('ifconfig wlan0 | grep "inet\ addr" | cut -d: -f2 | cut -d" " -
    f1')
    pi_ip = s.read()
    return pi_ip
```