```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);
```

```
65
66    // Include carriage return to start at beginning of line
67    if (data == '\n')
68    {
69      USARTWriteChar('\r');
70    }
71
72    USARTWriteChar(data);
73    return 0;
74  }
75
76  ISR(USART0_UDRE_vect) {
77    if (data_available > 0) {
78      char data = data_to_write[read_data_index];
79      ++read_data_index;
80      --data_available;
81
82      UDR0=data;
83      if (data_available == 0) {
84        read_data_index = 0;
85        write_data_index = 0;
86        UCSR0B &= ~(1<<UDRIE0);
87      }
88    } else {
89      USARTWriteChar('-');
90    }
91  }
```

```
/*
 * 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);
}

```

```
66 void notify_right_motor_speed_received(unsigned char speed) {
67   right_motor_speed_received(speed);
68 }
```

```
/*
 * 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 {
```

```
66        switch (addr) {
67          case DATA_CMD_LEN:
68            data_size = TWDR;
69            data_index = 0;
70            break;
71          case DATA_CMD_BYTE:
72            data_buffer[data_index] = TWDR;
73            ++data_index;
74            break;
75        }
76
77        i2c_state = I2C_STATE_READING_DATA;
78      }
79      TWCR |= (1<<TWINT); //Reset TWINT flag
80      break;
81
82    case (TW_SR_STOP) : //STOP or START condition received while selected
83      if(i2c_state == I2C_STATE_READING_DATA) {
84        //Eventually save data somewhere
85        i2c_state = I2C_STATE_WAITING_FOR_ADDR;
86        if(data_index == data_size) {
87          data_index = 0;
88          struct packet* p = malloc(sizeof(struct packet));
89          p->size = data_size;
90          memcpy(p->data, data_buffer, data_size);
91          queue_push(data_recieved, p);
92        }
93      }
94      TWCR |= (1<<TWINT); //Reset TWINT
95      break;
96
97    case (TW_ST_DATA_ACK) : //Data transmitted, ACK received
98    case (TW_ST_SLA_ACK) : //SLA+R received, ACK returned
99      if(i2c_state == I2C_STATE_WAITING_FOR_DATA) {
100       switch (addr) {
101         case DATA_CMD_LEN:
102           if (queue_empty(data_to_send)) {
103             TWDR = 0;
104           } else {
105             dts_index = 0;
106             struct packet* p = queue_front(data_to_send);
107             TWDR = p->size;
108           }
109           break;
110         case DATA_CMD_BYTE:
111           if (queue_empty(data_to_send)) {
112             TWDR = 0;
113           }
114           else {
115             struct packet* p = queue_front(data_to_send);
116             TWDR = p->data[dts_index];
117             ++dts_index;
118             if(dts_index == p->size) {
119               free(p);
120               queue_pop(data_to_send);
121             }
122           }
123           break;
124       }
125
126       i2c_state = I2C_STATE_WAITING_FOR_ADDR;
127     }
128     TWCR |= (1<<TWINT); //Reset TWINT
129     break;
130   case (TW_ST_DATA_NACK) : //Data received, NACK returned
```

```
131   case (TW_ST_LAST_DATA) : //last data byte transmitted, ACK received
132   case (TW_BUS_ERROR) : //Illegal start or stop condition
133   default:
134     TWCR |= (1 << TWINT); //Reset TWINT
135     i2c_state = I2C_STATE_WAITING_FOR_ADDR;
136   }
137   sei();
138 }
139
140 void send_data(struct packet* p) {
141   queue_push(data_to_send, p);
142 }
143
144 struct packet* get_received_data() {
145   if(queue_empty(data_recieved)) {
146     return NULL;
147   } else {
148     struct packet* p = queue_front(data_recieved);
149     queue_pop(data_recieved);
150     return p;
151   }
152 }
153
154 void initialize_i2c(unsigned char address)
155 {
156   data_to_send = queue_create();
157   data_recieved = queue_create();
158   //Initializing i2cslave
159   TWAR = (address<<1) & 0xFE; //Sets slavei2caddress and ignore general
160   TWDR = 0x00; //Initial data is set to 0
161
162   //Starts listening on i2c
163   //Reset TW-Interrupt, Enable TW-ACK, TW-Enabled, TW-Interrupt Enable
164   TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN) | (1<<TWIE);
165   sei();
166 }
```

```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));
```

```
66        break;
67      default:
68        break;
69    }
70
71    free(p);
72  }
```

```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) {
```

```
66      tail = tail->next;
67    }
68
69    struct queue* next = queue_create();
70    next->data = data;
71    tail->next = next;
72
73  }
74
75  bool queue_empty(struct queue* q) {
76    //assert(q->data == NULL);
77
78    return q->next == NULL;
79  }
```

```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
```

```
64    ICR1 = 16;                        // Set TOP
65
66  }
```

```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);
```

```
65
66    // Include carriage return to start at beginning of line
67    if (data == '\n')
68    {
69      USARTWriteChar('\r');
70    }
71
72    USARTWriteChar(data);
73    return 0;
74  }
```

```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);
```

```
65
66    // Include carriage return to start at beginning of line
67    if (data == '\n')
68    {
69      USARTWriteChar('\r');
70    }
71
72    USARTWriteChar(data);
73    return 0;
74  }
75
76  ISR(USART0_UDRE_vect) {
77    if (data_available > 0) {
78      char data = data_to_write[read_data_index];
79      ++read_data_index;
80      --data_available;
81
82      UDR0=data;
83      if (data_available == 0) {
84        read_data_index = 0;
85        write_data_index = 0;
86        UCSR0B &= ~(1<<UDRIE0);
87      }
88    } else {
89      USARTWriteChar('-');
90    }
91  }
```

```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) {
```

```
66      tail = tail->next;
67    }
68
69    struct queue* next = queue_create();
70    next->data = data;
71    tail->next = next;
72
73  }
74
75  bool queue_empty(struct queue* q) {
76    //assert(q->data == NULL);
77
78    return q->next == NULL;
79  }
```

```
/*
 * 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;
```

```
66  switch (twi_status) {
67  case (TW_SR_SLA_ACK) : //SLA+R received, ACK returned
68    i2c_state = I2C_STATE_WAITING_FOR_ADDR;
69    TWCR |= (1<<TWINT); //Reset TWINT flag
70    break;
71
72  case (TW_SR_DATA_ACK) : //Data received, ACK returned
73    if(i2c_state == I2C_STATE_WAITING_FOR_ADDR) {
74      addr = TWDR; //Saving address
75      i2c_state = I2C_STATE_WAITING_FOR_DATA;
76    } else {
77      switch (addr) {
78        case DATA_CMD_LEN:
79          data_size = TWDR;
80          data_index = 0;
81          break;
82        case DATA_CMD_BYTE:
83          data_buffer[data_index] = TWDR;
84          ++data_index;
85          break;
86      }
87
88      i2c_state = I2C_STATE_READING_DATA;
89    }
90    TWCR |= (1<<TWINT); //Reset TWINT flag
91    break;
92
93  case (TW_SR_STOP) : //STOP or START condition received while selected
94    if(i2c_state == I2C_STATE_READING_DATA) {
95      //Eventually save data somewhere
96      i2c_state = I2C_STATE_WAITING_FOR_ADDR;
97      if(data_index == data_size) {
98        data_index = 0;
99        struct packet* p = malloc(sizeof(struct packet));
100       p->size = data_size;
101       memcpy(p->data, data_buffer, data_size);
102       queue_push(data_recieved, p);
103     }
104   }
105   TWCR |= (1<<TWINT); //Reset TWINT
106   break;
107
108 case (TW_ST_DATA_ACK) : //Data transmitted, ACK received
109 case (TW_ST_SLA_ACK) : //SLA+R received, ACK returned
110   if(i2c_state == I2C_STATE_WAITING_FOR_DATA) {
111     switch (addr) {
112       case DATA_CMD_LEN:
113         if (queue_empty(data_to_send)) {
114           TWDR = 0;
115         } else {
116           dts_index = 0;
117           struct packet* p = queue_front(data_to_send);
118           TWDR = p->size;
119         }
120         break;
121       case DATA_CMD_BYTE:
122         if (queue_empty(data_to_send)) {
123           TWDR = 0;
124         }
125         else {
126           struct packet* p = queue_front(data_to_send);
127           TWDR = p->data[dts_index];
128           ++dts_index;
129           if(dts_index == p->size) {
130             free(p);
```

```
131                 queue_pop(data_to_send);
132               }
133             }
134             break;
135         }
136
137         i2c_state = I2C_STATE_WAITING_FOR_ADDR;
138       }
139       TWCR |= (1<<TWINT); //Reset TWINT
140       break;
141     case (TW_ST_DATA_NACK) : //Data received, NACK returned
142     case (TW_ST_LAST_DATA) : //last data byte transmitted, ACK received
143     case (TW_BUS_ERROR) : //Illegal start or stop condition
144     default:
145       TWCR |= (1 << TWINT); //Reset TWINT
146       i2c_state = I2C_STATE_WAITING_FOR_ADDR;
147     }
148     sei();
149 }
150
151 void send_data(struct packet* p) {
152   queue_push(data_to_send, p);
153 }
154
155 struct packet* get_received_data() {
156   if(queue_empty(data_recieved)) {
157     return NULL;
158   } else {
159     struct packet* p = queue_front(data_recieved);
160     queue_pop(data_recieved);
161     return p;
162   }
163 }
164 int main(void)
165 {
166   data_to_send = queue_create();
167   data_recieved = queue_create();
168   initialize_uart();
169   printf("Boooooooted\n");
170   //Initializing i2cslave
171   TWAR = (SLAVE_ADDRESS<<1) & 0xFE; //Sets slavei2caddress and ignore general
172   TWDR = 0x00; //Initial data is set to 0
173
174   //Starts listening on i2c
175   //Reset TW-Interrupt, Enable TW-ACK, TW-Enabled, TW-Interrupt Enable
176   TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN) | (1<<TWIE);
177   sei();
178   for(;;) {
179
180   }
181   return 1;
182 }
```

```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);
```

```
66    adc_init();
67
68    printf("BOOT\n");
69
70    listen_for_sensor_data_request(&handle_data_request);
71
72    return run_program(&handle_loop);
73  }
74
75
76  void adc_init(void) {
77    ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));  //FEL MATTE 16Mhz/128 = 125Khz
        the ADC reference clock
78    ADMUX |= ((1<<REFS0)|(1<<REFS1)); //2.56 internal voltage as reference
79    //ADMUX |= (1<<REFS0);              //Voltage reference, koppla 3.3v till AREF
80    ADCSRA |= (1<<ADEN);              //Turn on ADC
81    ADCSRA |= (1<<ADSC);              //Do an initial conversion because this one
        is the slowest and to ensure that everything is up and running
82  }
83
84  void adc_start(uint8_t channel){
85    ADMUX &= 0xE0;              //Clear the older channel that was read
86    ADMUX |= channel;           //Defines the new ADC channel to be read
87    ADCSRA |= (1<<ADSC);        //Starts a new conversion
88  }
89
90  bool adc_ready(){
91    return !(ADCSRA & (1<<ADSC));
92  }
93
94  uint16_t adc_synch(uint8_t channel){
95    adc_start(channel);
96    while(!adc_ready()) {};            //Wait until the conversion is done
97    return ADCW;                     //Returns the ADC value of the chosen channel
98  }
99
100 int to_mm(int n) {
101   const int min = 160;
102   if (n > min + 840 || min > n) return -1;
103   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, 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, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 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};
104   return data[n - min];
105 }
106
107 // For the 10cm-80cm sensor
108 int to_cm_large(int n) {
109   const int min = 165;
110   if (n > min + 775 || min > n) return -1;
111   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, 28, 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, 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, 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, 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, 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)

*/
```

```
/*
 * 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);
}

```

```
66 void handle_loop()
67 {
68
69 }
70
71 void initialize_PWM() {
72   DDRA = 0xFF;  // Set Data Direction on PortA
73   DDRD = 0xFF;  // Set Data Direction on PortD
74   TCNT1 = 0;    // Reset Timer1 counter
75
76   TCCR1A = 0;   // Clear Timer1 settings
77   TCCR1B = 0;   // Clear Timer1 settings
78
79   PORTA = 0x05; // Set direction of wheels (Pin 40 for left wheels, pin 38 for
       right wheels) (PORTA[0] left and PORTA[2] right)
80
81   TCCR1B|= (1<<WGM12)|(1<<CS10)|(1<<WGM13);     // No prescaler, Timer1 settings
82   TCCR1A|= (1<<COM1A1)|(1<<WGM11)|(1<<COM1B1);    // Timer1 settings
83   ICR1 = TOP;                       // Set TOP
84
85   // Set initial speed
86   OCR1A = 0;
87   OCR1B = 0;
88 }
89
90 int main(void)
91 {
92   // TODO: Register handlers
93   initialize_uart();
94   initialize_i2c(0x40);
95   initialize_PWM();
96
97   listen_for_motor_speed_received(&handle_motor_speed_received);
98   listen_for_left_motor_speed_received(&handle_left_motor_speed_received);
99   listen_for_right_motor_speed_received(&handle_right_motor_speed_received);
100
101   return run_program(&handle_loop);
102 }
```

```
/*
 * 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);
```

```
65
66    // Include carriage return to start at beginning of line
67    if (data == '\n')
68    {
69      USARTWriteChar('\r');
70    }
71
72    USARTWriteChar(data);
73    return 0;
74  }
```

```
/*
 * 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();
```

```
64
65    while(1)
66      {
67      if (adc_ready()) {
68        if (channel == MUX0) {
69          ir_right = to_mm(ADCW);
70          channel = MUX1;
71        }
72        else if (channel == MUX1) {
73          ir_left = to_mm(ADCW);
74          channel = MUX0;
75        }
76        adc_start(channel);
77      }
78      printf("left: %d, right: %d\n", ir_left, ir_right);
79    }
80  }
81
82  int to_mm(int n) {
83    const int min = 280;
84    if (n > min + 720 || min > n) return -1;
85    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, 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, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 51, 51, 51,
        51, 51, 51, 51, 51, 50, 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, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 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};
86    return data[n - min];
87  }
```

```
88
89 /*
90 intervall 280 < n < 1000
91 Mätdata från vänster sensor [1023, 872, 761, 664, 590, 531, 476, 430, 400, 374,
       350, 327, 304, 289, 271, 258, 239]
92
93
94 //Python för att sampla funktionen
95 import math
96 from numpy import arange
97
98 min = 280
99 max = 1000
100 samples = arange(min, max, 5)
101 vals = list([int(str(round(29.1 * math.exp(-0.002155 * x), 0)
102 ).replace('.0', '')) for x in range(min, max)])
103
104 print(vals)
105 */
```

```
/*
 * 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
/*
 * 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_ */
```

```c
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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_ */
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                bt_client.py                                #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-12-17                      #
7  #                                                                            #
8  #                               Rebecca Lindblom                             #
9  #                              Matildha Sjöstedt                             #
10 #                                Patrik Sletmo                               #
11 #                                                                            #
12 ##############################################################################
13
14 import bluetooth
15 import time
16 import traceback
17 import threading
18 import queue
19 import protocol
20 from bt_task import BT_task
21 from ast import literal_eval
22
23
24 class BT_client(threading.Thread):
25     PI_ADDR = "B8:27:EB:FC:55:27"
26     PORT = 3
27
28     def __init__(self, queue_handler):
29         self.queue_handler = queue_handler
30         self.exit_demanded = False
31         self.restart_demanded = False
32         self.client_sock = None
33         threading.Thread.__init__(self)
34         self.daemon = True
35         self.is_connected = False
36
37
38     '''
39     Creates a new client sock and attempts to connect to
40     addr via port. Timeout can be specified, default value is
41     10 seconds. The created socket is returned if connection
42     was succesful, else return None.
43     '''
44
45     def _setup_bt_client(self, timeout=10):
46         self.client_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
47         self.client_sock.setblocking(True)
48
49         while timeout > 0:
50             try:
51                 self.client_sock.connect((self.PI_ADDR, self.PORT))
52                 print("Successfully connected to ", self.PI_ADDR)
53                 timeout = -1
54             except bluetooth.btcommon.BluetoothError:
55                 time.sleep(1)
56                 timeout -= 1
57                 print("Waiting for connection...")
58                 continue
59             except OSError:
60                 time.sleep(1)
61                 timeout -= 1
62                 print("OSError in _setup_bt_client (waiting connection)")
63                 continue
64
65         if timeout == 0:
```

```
66              print("Connection timeout. Could not connect to server!")
67              return None
68          else:
69              print("Connected to ", self.PI_ADDR)
70              self.is_connected = True
71
72              return self.client_sock
73
74      '''
75           Loops send(),_recieve() until a shutdown or exit command is issued
76           '''
77
78      def _start_bt_client(self):
79          self.client_sock = self._setup_bt_client()
80
81          if self.client_sock:
82              while True:
83                  self._send()
84                  status = self._receive()
85                  if status != "":
86                      return status
87          else:
88              print("Could not connect to server, no socket created")
89
90      def _send(self):
91          bt_out_task = self.queue_handler.pop_out_queue()
92          if bt_out_task:
93              self.current_out_task = bt_out_task
94              try:
95                  self.client_sock.send(str(bt_out_task.cmd_id))
96              except bluetooth.btcommon.BluetoothError:
97                  pass
98              except OSError:
99                  pass
100
101             if bt_out_task.cmd_id == protocol.BT_SERVER_SHUTDOWN:
102                 self.exit_demanded = True
103             elif bt_out_task.cmd_id == protocol.BT_SERVER_RESTART:
104                 self.restart_demanded = True
105         else:
106             self.current_out_task = BT_task(0, 0)
107
108
109     def _receive(self):
110         # Wait for incoming messages for 0.1 seconds
111         recv_timeout = 0.1  # Receive timeout 0.1 seconds
112         data = ""
113         self.client_sock.settimeout(recv_timeout)
114         try:
115             data = self.client_sock.recv(1024).decode('utf-8')
116         except bluetooth.btcommon.BluetoothError:
117             # Recieved when server responds to shutdown
118             pass
119         except OSError:
120             pass
121         self.client_sock.settimeout(0)
122
123         if self.restart_demanded:
124             # Restart requested
125             self.client_sock.close()
126             del self.client_sock
127             return "RESTART"
128         elif self.exit_demanded:
129             # Shutdown requested
130             self.client_sock.close()
```

```
131             del self.client_sock
132             return "EXIT"
133 
134         if data:
135             data = literal_eval(data)
136             bt_in_task = BT_task(data[0], data[1])
137 
138             self.queue_handler.post_in_queue(bt_in_task)
139             print("Bt client received: ", str(data[1]))
140 
141         return ""
142 
143     '''
144     Overriden run()-method form threading.Thread.
145     Updates client until a shutdown command is
146     issued.
147     '''
148 
149     def run(self):
150         status = ""
151         while not status == "EXIT":
152             self.restart_demanded = False
153             self.exit_demanded = False
154             status = self._start_bt_client()
155             if status == "ERROR":
156                 # TODO Add a task to out_queue
157                 print("A Bluetooth error occurred!")
158             # Sleep so server has time to restart
159             time.sleep(2)
```

```
################################################################################
#                                                                              #
#                                   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))
```

```
66      pickle.dump(bt_task, command_queue)
67      print("Could dump to pickle in post_incoming")
68      # pickle.Pickler.clear_memo(self=)
69      command_queue.close()
70      print("Closing file and return to bt_server")
71
72
73  # kallas från server
74  def pop_outgoing():
75      answer_queue = open("bt_answers.txt", "rb")
76      task = None
77
78      task_q = []
79      while (True):
80          try:
81              task_i = pickle.load(answer_queue)
82              task_q.append(task_i)
83          except EOFError:
84              break
85      if task_q:
86          task = BT_task(task_q[0].cmd_id, task_q[0].data)
87          del task_q[0]
88          answer_queue = open("bt_answers.txt", "wb")
89          for task_i in task_q:
90              pickle.dump(task_i, answer_queue)
91
92      answer_queue.close()
93      return task
```

```
1  ##############################################################################
2  #                                                                            #
3  #                              bt_test_client.py                             #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-12-14                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                               Matildha Sjöstedt                            #
10 #                                                                            #
11 ##############################################################################
12
13 import bluetooth
14 import time
15 import traceback
16 import protocol
17
18 PI_ADDR = "B8:27:EB:FC:55:27"
19 USB_BT_ADDR = ""
20 PORT = 3
21
22
23 def setup_bt_client(addr, port):
24     client_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
25     client_sock.setblocking(True)
26     timeout = 10
27     while timeout > 0:
28         try:
29             client_sock.connect((addr, port))
30             timeout = -1
31         except bluetooth.btcommon.BluetoothError:
32             time.sleep(1)
33             timeout -= 1
34             print("Waiting for connection...")
35             continue
36
37     if timeout == 0:
38         print("Could not connect to server! PLZ try again and hope for better
       luck")
39         return client_sock
40     else:
41         print("Successfully connected to ", addr)
42         return client_sock
43
44
45 def main():
46     restart = ""
47     while not restart == "EXIT":
48         restart = run()
49         time.sleep(1)
50
51
52 def run():
53     client_sock = setup_bt_client(PI_ADDR, PORT)
54
55     while (True):
56         msg = input("To server: ")
57
58         client_sock.send(msg)
59
60         data = ""
61
62         try:
63             while data == "":
64                 data = client_sock.recv(1024).decode('utf-8')
```

```
65              if len(data) == 0:
66                  break
67          except bluetooth.btcommon.BluetoothError:
68              # Recieved when server responds to shutdown
69              client_sock.close()
70              del client_sock
71              if int(msg) == protocol.BT_SERVER_RESTART:
72                  # Restart requested
73                  return "RESTART"
74              elif int(msg) == protocol.BT_SERVER_SHUTDOWN:
75                  # Shutdown requested
76                  return "EXIT"
77
78
79  main()
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                client_main.py                              #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-12-15                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                               Matildha Sjöstedt                            #
10 #                                 Patrik Sletmo                              #
11 #                                                                            #
12 ##############################################################################
13
14 from gui import GUI
15 import outbound
16 from protocol import *
17 from eventbus import EventBus
18 import datetime
19 from ast import literal_eval
20 import random
21
22 try:
23     from bt_client import BT_client
24     bluetooth_enabled = True
25 except:
26     bluetooth_enabled = False
27
28 DATA_REQUEST_INTERVAL = 500  # milliseconds
29 IP_REQUEST_INTERVAL = 10     # seconds
30 UPDATE_INTERVAL = 250        # milliseconds
31
32 gui = None
33 bt_client = None
34 request_type = 0
35 last_data_request_time = datetime.datetime.now()
36 last_ip_request_time = datetime.datetime.now()
37
38
39 def run_bt_client(queue_handler):
40     global bt_client
41     bt_client = BT_client(queue_handler)
42     bt_client.start()
43
44
45 def update_ip(data):
46     global gui
47     gui.update_ip(data)
48
49
50 def add_sensor_data(data):
51     global gui
52     gui.add_sensor_data(data)
53
54
55 def add_servo_data(data):
56     global gui
57     gui.add_servo_data(data)
58
59
60 def update_map(data):
61     global gui
62     print(data)
63     gui.update_map(data)
64
65
```

```
66 def update_selected_mode(mode):
67     gui.update_selected_mode(mode)
68
69
70 def setup_subscriptions():
71     EventBus.subscribe(RETURN_PI_IP, update_ip)
72     EventBus.subscribe(BT_RETURN_SENSOR_DATA, add_sensor_data)
73     EventBus.subscribe(BT_RETURN_SERVO_DATA, add_servo_data)
74     EventBus.subscribe(BT_RETURN_MAP_DATA, update_map)
75     EventBus.subscribe(CMD_MODE_SET, update_selected_mode)
76
77
78 def request_data():
79     global last_ip_request_time, request_type
80     if request_type == 0:
81         outbound.bt_request_sensor_data()
82         request_type = 1
83     else:
84         outbound.bt_request_servo_data()
85         request_type = 0
86     outbound.bt_request_map_data()
87     if (datetime.datetime.now() - last_ip_request_time) > datetime.timedelta(
88             seconds=IP_REQUEST_INTERVAL):
89         outbound.request_ip()
90         last_ip_request_time = datetime.datetime.now()
91     pass
92
93
94
95 def update():
96     global gui, last_data_request_time, curr_test_corn, bt_client
97     if not gui.exit_demanded:
98         if gui.finished_setup:
99             EventBus.receive()
100            if (datetime.datetime.now() - last_data_request_time) > datetime.
    timedelta(
101                     milliseconds=DATA_REQUEST_INTERVAL):
102                 if bt_client is not None and bt_client.is_connected:
103                     request_data()
104
105                 last_data_request_time = datetime.datetime.now()
106         else:
107             gui.setup_after_main_loop()
108         gui.canvas.after(UPDATE_INTERVAL, update)
109     else:
110         print("Exit gui in client main")
111         outbound.bt_restart()
112         while bt_client is not None and not bt_client.restart_demanded:
113             pass
114         gui.close_window()
115
116
117 def start_gui():
118     global gui
119     gui.canvas.after(UPDATE_INTERVAL, update)
120     gui.root.mainloop()
121
122
123 def main():
124     global gui
125     queue_handler = EventBus.queue_handler
126     setup_subscriptions()
127
128     # MacOS has no support for PyBluez so by disabling the use of it we
129     # can still provide a semi-functional experience for Mac users.
```

```
130     if bluetooth_enabled:
131         run_bt_client(queue_handler)
132     else:
133         print('NOTICE: PyBluez module could not be loaded!')
134         print('Bluetooth functionality has been disabled.')
135 
136     gui = GUI()
137     start_gui()
138 
139 try:
140     main()
141 except:
142     print("Some error in client main")
143     outbound.bt_restart()
144     while bt_client is not None and not bt_client.restart_demanded:
145         pass
146     gui.close_window()
```

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

```
66
67              EventBus.notify(data.cmd_id, data.data)
68
69      @staticmethod
70      def subscribe(command_id, handler):
71          observer = EventBus._get_observer_for_command(command_id)
72          observer.subscribe(handler)
73
74      @staticmethod
75      def notify(command_id, *args):
76          observer = EventBus._get_observer_for_command(command_id)
77          observer.notify(*args)
78
79      @staticmethod
80      def _get_observer_for_command(command_id):
81          if command_id not in EventBus.observers:
82              EventBus.observers[command_id] = Observer()
83
84          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)
```

```
63          self.main_frame.bind('<a>', self.left)
64          self.root.bind('<Left>', self.left)
65          self.main_frame.bind('<d>', self.right)
66          self.root.bind('<Right>', self.right)
67          self.main_frame.bind('<q>', self.forward_left)
68          self.main_frame.bind('<e>', self.forward_right)
69
70          self.last_key_event_time = datetime.datetime.now()
71
72          # --- Canvas ---
73          self.canvas = Canvas(self.main_frame,
74                               width=GUI.CANVAS_X,
75                               height=GUI.CANVAS_Y, bg="#CCCCCC")
76          self.canvas.grid(column=0, row=0, padx=10, pady=10)
77
78          # --- Lists ---
79          self.list_frame = Frame(self.main_frame, width=self.LIST_FRAME_X, height
    =self.LIST_FRAME_Y,
80                                  bg=self.BG_COLOR)
81          self.list_frame.grid(row=0, column=1, padx=10)
82
83          self.ir_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
84          self.ir_list.grid(row=1, column=0)
85          self.ir_list_nr_items = 0
86          self.ir_label = Label(self.list_frame, text="IR data (mm) \n (left_back,
     left, right, right_back)",
87                                fg="black", bg=self.BG_COLOR)
88          self.ir_label.grid(row=0, column=0)
89
90          self.laser_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
91          self.laser_list.grid(row=1, column=1)
92          self.laser_list_nr_items = 0
93          self.laser_label = Label(self.list_frame, text="Laser data", fg="black",
     bg=self.BG_COLOR)
94          self.laser_label.grid(row=0, column=1)
95
96          self.gyro_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
97          self.gyro_list.grid(row=3, column=0)
98          self.gyro_list_nr_items = 0
99          self.gyro_label = Label(self.list_frame, text="Gyro data \n (degrees)",
    fg="black", bg=self.BG_COLOR)
100         self.gyro_label.grid(row=2, column=0)
101
102         self.servo_list = Listbox(self.list_frame, height=self.MAX_LIST_ITEMS)
103         self.servo_list.grid(row=3, column=1)
104         self.servo_list_nr_items = 0
105         self.servo_label = Label(self.list_frame, text="Servo data \n (speed)",
    fg="black", bg=self.BG_COLOR)
106         self.servo_label.grid(row=2, column=1)
107
108         # --- Buttons ---
109         self.btn_frame = Frame(self.main_frame, width=self.BTN_FRAME_X, height=
    self.BTN_FRAME_Y)
110         self.btn_frame.grid(row=1, column=0, pady=10, padx=10)
111
112         self.btn_forward = Button(self.btn_frame, text="Forward", command=self.
    forward)
113         self.btn_forward.grid(row=1, column=3)
114
115         self.btn_back = Button(self.btn_frame, text="Back", command=self.back)
116         self.btn_back.grid(row=1, column=4)
117
118         self.btn_right = Button(self.btn_frame, text="Right", command=self.right
    )
119         self.btn_right.grid(row=1, column=5)
```

```
120
121         self.btn_left = Button(self.btn_frame, text="Left", command=self.left)
122         self.btn_left.grid(row=1, column=1)
123
124         self.btn_forward_right = Button(self.btn_frame, text="Forward left",
    command=self.forward_left)
125         self.btn_forward_right.grid(row=0, column=3, padx=5, pady=2)
126
127         self.btn_forward_left = Button(self.btn_frame, text="Forward right",
    command=self.forward_right)
128         self.btn_forward_left.grid(row=0, column=4, padx=5, pady=2)
129
130         self.mode = IntVar()
131         self.mode.set(1)
132         self.radio_frame = Frame(self.btn_frame)
133         self.radio_frame.grid(row=0, column=6)
134         self.btn_auto_mode = Radiobutton(self.radio_frame, text=self.MODES
    [0][0], variable=self.mode,
135                                          command=self.change_mode, indicatoron=0, value=
    self.MODES[0][1])
136         self.btn_auto_mode.grid(row=0, column=0, padx=2, pady=2, sticky="W")
137         self.btn_manual_mode = Radiobutton(self.radio_frame, text=self.MODES
    [1][0], variable=self.mode,
138                                            command=self.change_mode, indicatoron
    =0, value=self.MODES[1][1])
139         self.btn_manual_mode.grid(row=1, column=0, padx=2, pady=2, sticky="W")
140
141         self.ip_box = Label(self.main_frame, text="Pi IP: ", width=25, bg="white
    ")
142         self.ip_box.grid(row=1, column=1)
143
144         # --- Image ----
145         self.image_frame = Frame(self.btn_frame)
146         self.image_frame.grid(row=0, column=0, rowspan=2)
147         logo = PhotoImage(file="Logo.gif")
148         self.resampled_logo = logo.subsample(3, 3)
149         self.logo_box = Label(self.image_frame, image=self.resampled_logo)
150         self.logo_box.grid(row=1, column=0, padx=10, sticky=W+E+N+S)
151
152         self.map_grid.draw_grid(self.canvas)
153
154     def setup_after_main_loop(self):
155         self.map_grid.draw_grid(self.canvas)
156         self.btn_auto_mode.select()
157         self.btn_manual_mode.deselect()
158         self.finished_setup = True
159
160     '''
161     Values should be a list containing of [ir_left,ir_right,ir_left_back,
162     ir_right_back,laser,gyro]
163     '''
164
165     def add_sensor_data(self, values):
166         ir_values = str(values[2]) + ", " + str(values[0]) + ", " + str(values
    [1]) + ", " + str(values[3])
167
168         self.ir_list.insert(0, ir_values)
169         if self.ir_list_nr_items >= self.MAX_LIST_ITEMS:
170             self.ir_list.delete(self.MAX_LIST_ITEMS)
171         else:
172             self.ir_list_nr_items += 1
173
174         self.laser_list.insert(0, str(values[4]))
175         if self.laser_list_nr_items >= self.MAX_LIST_ITEMS:
176             self.laser_list.delete(self.MAX_LIST_ITEMS)
```

```
177             else:
178                 self.laser_list_nr_items += 1
179 
180             self.gyro_list.insert(0, str(values[5]))
181             if self.gyro_list_nr_items >= self.MAX_LIST_ITEMS:
182                 self.gyro_list.delete(self.MAX_LIST_ITEMS)
183             else:
184                 self.gyro_list_nr_items += 1
185 
186     '''
187     Values should be a list containing of [left_speed, right_speed].
188     '''
189 
190     def add_servo_data(self, values):
191         self.servo_list.insert(0, str(values[0]) + ', ' + str(values[1]))
192         if self.servo_list_nr_items >= self.MAX_LIST_ITEMS:
193             self.servo_list.delete(self.MAX_LIST_ITEMS)
194         else:
195             self.servo_list_nr_items += 1
196 
197     def update_map(self, values):
198         self.map_grid.update_map(values, self.canvas)
199 
200     '''
201     Ip expected to be in format [ip]
202     '''
203 
204     def update_ip(self, ip):
205         self.ip_box.config(text="Pi IP: " + str(ip[0]))
206 
207     def exit(self):
208         self.exit_demanded = True
209 
210     def close_window(self):
211         self.root.destroy()
212 
213     def check_key_event_time(self):
214         return (datetime.datetime.now() - self.last_key_event_time) > datetime.
    timedelta(
215             milliseconds=self.MIN_TIME_KEY_EVENT)
216 
217     '''Functions for handling key press.
218     Takes forced event, but ignores it and calls correct driver function.
219     '''
220 
221     def forward(self, event=None):
222         self.event_handler(outbound.bt_drive_forward, event=event, repetition=5)
223 
224     def back(self, event=None):
225         self.event_handler(outbound.bt_drive_back, event=event, repetition=5)
226 
227     def left(self, event=None):
228         self.event_handler(outbound.bt_turn_left, event=event, repetition=3)
229 
230     def right(self, event=None):
231         self.event_handler(outbound.bt_turn_right, event=event, repetition=3)
232 
233     def forward_right(self, event=None):
234         self.event_handler(outbound.bt_forward_right, event=event, repetition=5)
235 
236     def forward_left(self, event=None):
237         self.event_handler(outbound.bt_forward_left, event=event, repetition=5)
238 
239     '''
240     Makes sure event can not occur faster than a predefined time interval.
```

```
241     If event option is left out or set to None, event_handler will interpret
        that as if
242     it was called from a button and call the command function number of times
243     specified in repetition option.
244     '''
245
246     def event_handler(self, command, **options):
247         if self.check_key_event_time():
248             if not options["event"]:
249                 for i in range(0, options["repetition"]):
250                     while not self.check_key_event_time():
251                         continue
252                     command()
253                     self.last_key_event_time = datetime.datetime.now()
254             else:
255                 command()
256                 self.last_key_event_time = datetime.datetime.now()
257
258     def change_mode(self):
259         mode = self.mode.get()
260         if self.MODES[mode][0] == "Manual":
261             outbound.bt_switch_to_manual()
262         else:
263             outbound.bt_switch_to_auto()
264
265     def update_selected_mode(self, mode):
266         # btn_auto_mode and btn_manual_mode are "reversed", so btn_auto_mode
267         # has the text "Manual" and btn_manual_mode the text "Automatic"
268         if mode == GUI.ROBOT_MODE_MANUAL:
269             self.btn_auto_mode.select()
270             self.btn_manual_mode.deselect()
271         elif mode == GUI.ROBOT_MODE_AUTONOMOUS:
272             self.btn_auto_mode.deselect()
273             self.btn_manual_mode.select()
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                  inbound.py                                #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                           Senast modifierad 2016-11-30                     #
7  #                                                                            #
8  #                                Matildha Sjöstedt                           #
9  #                                                                            #
10 ##############################################################################
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                map_grid.py                                 #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-12-14                      #
7  #                                                                            #
8  #                               Rebecca Lindblom                             #
9  #                              Matildha Sjöstedt                             #
10 #                                Patrik Sletmo                               #
11 #                                                                            #
12 ##############################################################################
13
14 class MapGrid:
15     MAX_MAP_SIZE = 15 * 2  # 1 cell is 40x40 cm, 28 x 28 cells in map
16     NR_ROWS = MAX_MAP_SIZE
17     NR_COLS = NR_ROWS
18     OFFSET = 15
19
20     def __init__(self):
21         # List of coordinates of all corners on the map,
22         # listed in the order of discovery
23         self.raw_map_data = []
24         self.actual_map_data = []
25         self.new_raw_map_data = []  # The latest map data received
26         self.new_actual_map_data = []
27         self.start_position = (self.OFFSET, self.OFFSET)
28         # Offset calculated from given coordinates to match coordinates on map
       grid
29
30     def update_map(self, data, canvas):
31         self._update_map_data(data)
32         self.new_actual_map_data = []
33         self._calc_actual_coords(canvas)
34         self._draw_blocks(canvas)
35
36     def draw_grid(self, canvas):
37         size = canvas.winfo_width() / self.NR_ROWS
38         for row in range(0, self.NR_ROWS):
39             canvas.create_line(0, row * size, canvas.winfo_width(), row * size,
       fill="#FFFFFF")
40             canvas.create_line(row * size, 0, row * size, canvas.winfo_width(),
       fill="#FFFFFF")
41
42     '''
43     Expects data to be a list containing ALL map data coordinates.
44     '''
45
46     def _update_map_data(self, data):
47         visited = data[0]
48         nr_new_items = len(visited) - len(self.raw_map_data)
49         self.new_raw_map_data = visited[:nr_new_items]
50         self.raw_map_data = visited
51         self.new_raw_map_data = visited
52
53     '''
54     Appends internal list of corners with actual coordinates corresponding to
55     size of map in pixels.
56     '''
57
58     def _calc_actual_coords(self, canvas):
59         visited = self.new_raw_map_data
60         for block in visited:
61             # Match coordinates to grid
62             if block:
```

```
63                  x = block[0]
64                  y = 0 - block[1] # Flip coordinate system on canvas
65                  x += self.OFFSET
66                  y += self.OFFSET
67
68                  # Convert raw coordinates to actual coordinates corresponding to
     canvas pixels
69                  actual_x = (canvas.winfo_width() * x) / self.NR_COLS
70                  actual_y = (canvas.winfo_height() * y) / self.NR_ROWS
71                  self.new_actual_map_data.append((actual_x, actual_y))
72
73      '''
74      Draw lines between corners. Replaces map data with actual coordinates with
        only the last
75      visited corner.
76      '''
77
78      def _draw_blocks(self, canvas):
79          block_size = canvas.winfo_height() / self.NR_ROWS
80
81          visited = self.new_actual_map_data
82          for block in visited:
83              canvas.create_rectangle(block[0], block[1], block[0] + block_size,
        block[1] + block_size,
84                                      fill="#FFFFFF", outline="white")
85              self.actual_map_data.append(block)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                observer.py                                 #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                           Senast modifierad 2016-11-27                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                                                                            #
10 ##############################################################################
11
12 """
13 Simple implementation of the observer pattern.
14
15 See https://en.wikipedia.org/wiki/Observer_pattern for more information.
16 """
17
18
19 class Observer:
20     def __init__(self):
21         self.subscribers = []
22
23     def subscribe(self, func):
24         if func not in self.subscribers:
25             self.subscribers.append(func)
26
27     def notify(self, *args, **kwargs):
28         for subscriber in self.subscribers:
29             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
```

```
66          )
67      )
68
69
70  def bt_drive_forward():
71      EventBus.post(
72          BLUETOOTH_ADDR,
73          BT_task(
74              BT_DRIVE_FORWARD
75          )
76      )
77
78
79  def bt_drive_back():
80      EventBus.post(
81          BLUETOOTH_ADDR,
82          BT_task(
83              BT_DRIVE_BACK
84          )
85      )
86
87
88  def bt_turn_right():
89      EventBus.post(
90          BLUETOOTH_ADDR,
91          BT_task(
92              BT_TURN_RIGHT
93          )
94      )
95
96  def bt_forward_right():
97      EventBus.post(
98          BLUETOOTH_ADDR,
99          BT_task(
100             BT_DRIVE_FORWARD_RIGHT
101         )
102     )
103
104 def bt_turn_left():
105     EventBus.post(
106         BLUETOOTH_ADDR,
107         BT_task(
108             BT_TURN_LEFT
109         )
110     )
111
112 def bt_forward_left():
113     EventBus.post(
114         BLUETOOTH_ADDR,
115         BT_task(
116             BT_DRIVE_FORWARD_LEFT
117         )
118     )
119
120 def bt_shutdown():
121     EventBus.post(
122         BLUETOOTH_ADDR,
123         BT_task(
124             BT_SERVER_SHUTDOWN
125         )
126     )
127
128
129 def bt_restart():
130     EventBus.post(
```

```
131         BLUETOOTH_ADDR,
132         BT_task(
133             BT_SERVER_RESTART
134         )
135     )
136
137 def bt_switch_to_auto():
138     EventBus.post(
139         BLUETOOTH_ADDR,
140         BT_task(
141             AUTONOMOUS_MODE
142         )
143     )
144
145
146 def bt_switch_to_manual():
147     EventBus.post(
148         BLUETOOTH_ADDR,
149         BT_task(
150             MANUAL_MODE
151         )
152     )
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                protocol.py                                 #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-12-17                      #
7  #                                                                            #
8  #                               Rebecca Lindblom                             #
9  #                              Matildha Sjöstedt                             #
10 #                                Patrik Sletmo                               #
11 #                                                                            #
12 ##############################################################################
13
14 """
15 The protocol for the robot consist of various commands, or events, passed along
16 the main bus using a distributed event bus. Each command is identified by its
17 command id and is transmitted before eventual arguments.
18
19 All commands may be sent in any direction but the implementations will probably
20 choose to ignore irrelevant one. For messages originating from the main unit,
21 see outbound.py.
22 """
23
24 # Addresses for the units on the bus. Note that the laser cannot be queried
25 # using the protocol described in bus.py.
26
27
28 BLUETOOTH_ADDR = 0xBEEF
29
30 # Packet addresses
31 PACKET_HEADER = 0
32 PACKET_DATA = 1
33
34 # Request data from the sensor unit
35 CMD_REQUEST_SENSOR_DATA = 1
36 """
37 Issues a request to the sensor unit prompting it to send its most recent sensor
38 data back to the main unit. As the data transfer is asynchronous the sensor
39 unit responds to the request by posting a CMD_RETURN_SENSOR_DATA command on the
40 bus.
41
42 Target: Sensor unit
43
44 Arguments: None
45 """
46
47 # Return sensor data from the sensor unit
48 CMD_RETURN_SENSOR_DATA = 2
49 """
50 Command sent from the sensor unit after a request for its sensor data has been
51 made. As data from various sensors are reported independently it is not certain
52 that the value from sensor A and value from sensor B reflect the world at the
53 same point in time.
54
55 Target: Main unit
56
57 Arguments:
58 ir_left_mm (2 bytes, two's complement)
59     Distance recorded by the left IR sensor in mm, or -1 if the distance is not
60     within the supported range.
61 ir_right_mm (2 bytes, two's complement)
62     Distance recorded by the right IR sensor in mm, or -1 if the distance is
63     not within the supported range.
64 """
65
```

```
66 # Ping a unit
67 CMD_PING = 3
68 """
69 Dummy command which will only trigger the other unit to respond with a PONG
70 command. Preferably used to test a connection without forcing the other party
71 to perform any actual action.
72
73 Target: Any AVR unit
74
75 Arguments: None
76 """
77
78 # Pong a unit
79 CMD_PONG = 4
80 """
81 Reply to a PING command.
82
83 Target: Main unit
84
85 Arguments: None
86 """
87
88 # Set both motor speeds in the control unit
89 CMD_SET_MOTOR_SPEED = 5
90 """
91 Sets the speed of both the left and right motors on the robot. Values can
92 be both positive and negative and the range -100 to 100 is supported, where the
93 value represents a percentage of the max speed. It seems like the different
94 directions have various max speeds, which has to be considered when
95 implementing on-spot-rotation or other actions which assume equal speed forward
96 and backwards.
97
98 Target: Control unit
99
100 Arguments:
101 left_motor_speed (1 byte, positive or negative)
102     Left motor speed in percentage of max speed ranging from -100 to 100.
103 right_motor_speed (1 byte, positive or negative)
104     Right motor speed in percentage of max speed ranging from from -100 to 100.
105 """
106
107 # Set left motor speed only in control unit
108 CMD_SET_LEFT_MOTOR_SPEED = 6
109 """
110 Sets the speed of the left motors on the robot. It is only possible to pass
111 positive values with the command in the range 0 to 100, where the value
112 represent a percentage of the max speed.
113
114 Target: Control unit
115
116 Arguments:
117 speed (1 byte, positive)
118     Left motor speed in percentage of max speed ranging from 0 to 100.
119 """
120
121 # Set right motor speed in the control unit
122 CMD_SET_RIGHT_MOTOR_SPEED = 7
123 """
124 Sets the speed of the right motors on the robot. It is only possible to pass
125 positive values with the command in the range 0 to 100, where the value
126 represent a percentage of the max speed.
127
128 Target: Control unit
129
130 Arguments:
```

```
131 speed (1 byte, positive)
132     Right motor speed in percentage of max speed ranging from 0 to 100.
133 """
134 # Indicates that the robot has started turning
135 """
136 Event called internally within the main unit to indicate that a simple 90
137 degree turn has been initiated.
138
139 Target: Main unit
140
141 Arguments: None
142 """
143 CMD_TURN_STARTED = 8
144
145 # Indicates that the robot has stopped turning
146 """
147 Event called internally within the main unit to indicate that a simple 90
148 degree turn has finished.
149
150 Target: Main unit
151
152 Arguments:
153 is_right_turn (1 byte, boolean)
154     True for right turn, false for left turn.
155 """
156 CMD_TURN_FINISHED = 9
157
158 # -------------------- Bluetooth commands ---------------------
159
160 REQUEST_PI_IP = 10
161 RETURN_PI_IP = 11
162
163 BT_SERVER_RESTART = 12
164
165 BT_SERVER_SHUTDOWN = 13
166
167 BT_REQUEST_SENSOR_DATA = 14
168 BT_RETURN_SENSOR_DATA = 15
169
170 BT_REQUEST_SERVO_DATA = 16
171 BT_RETURN_SERVO_DATA = 17
172
173 BT_REQUEST_MAP_DATA = 18
174 BT_RETURN_MAP_DATA = 19
175
176 BT_DRIVE_FORWARD = 20
177 BT_DRIVE_BACK = 21
178
179 BT_TURN_RIGHT = 22
180 BT_TURN_LEFT = 23
181
182 BT_DRIVE_FORWARD_RIGHT = 24
183 BT_DRIVE_FORWARD_LEFT = 25
184
185 AUTONOMOUS_MODE = 26
186 MANUAL_MODE = 27
187
188 # Indicates that the robot has changed to a new navigator mode
189 CMD_MODE_SET = 29
190 """
191 Command to toggle between the available modes (autonomous and manual) instead
192 of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.
193
194 Target: Main unit
195
```

```
196 Arguments:
197 new_mode (1 byte)
198     Integer representation of the new mode:
199       0 = Manual mode
200       1 = Autonomous mode
201 """
202
203 BT_CLIENT_COMMANDS = [REQUEST_PI_IP, BT_SERVER_RESTART,
204                       BT_SERVER_SHUTDOWN, BT_REQUEST_SENSOR_DATA,
205                       BT_REQUEST_MAP_DATA, BT_REQUEST_SERVO_DATA,
    BT_DRIVE_FORWARD, BT_DRIVE_BACK,
206                       BT_TURN_RIGHT, BT_TURN_LEFT, BT_DRIVE_FORWARD_RIGHT,
    BT_DRIVE_FORWARD_LEFT]
207
208 BT_SERVER_COMMANDS = [RETURN_PI_IP, BT_RETURN_SENSOR_DATA,
209                       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
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                acceldriver.py                              #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-11-20                     #
7  #                                                                            #
8  #                                Anton Dalgren                               #
9  #                                                                            #
10 ##############################################################################
11
12 from time import sleep
13 #from datetime import datetime, timedelta
14
15 from accel import Accel
16 #from driver import Driver
17
18 accel = Accel()
19
20 while (True):
21     data = accel.read_data()
22     print("x-accel: " + str(data))
23     sleep(0.2)
```

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

```

```
63          if (regulation > self.MAX_REGULATION):
64              regulation = self.MAX_REGULATION
65          elif (regulation < -self.MAX_REGULATION):
66              regulation = -self.MAX_REGULATION
67
68          if (regulation > -10):
69              speed_close_wall = self.get_speed(ir_right_mm, ir_right_back_mm) +
    regulation
70          else:
71              speed_close_wall = 10
72
73          if (regulation < 10):
74              speed_far_wall = self.get_speed(ir_right_mm, ir_right_back_mm) -
    regulation
75          else:
76              speed_far_wall = 10
77
78          time_last_regulation = time_now
79
80          return int(speed_close_wall), int(speed_far_wall), regulation
81
82      def get_speed(self, ir_right_mm, ir_right_back_mm):
83          if ir_right_mm == -1 and ir_right_back_mm != -1:
84              return self.STANDARD_SPEED
85          else:
86              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):
        """
```

```
66          Sends data via bluetooth to connected client.
67          Sends data saved in server, unless data is passed.
68          """
69          if not data:
70              self.client_sock.send(self.outgoing_data)
71          else:
72              self.client_sock.send(data)
73
74      def _pop_from_outgoing(self):
75          return bt_task_handler.pop_outgoing()
76
77      def update_incoming(self):
78          """
79          Updates incoming data aimed for the robot by receiving via bluetooth.
80          Saves the new data in itself.
81          Returns True/False whether or not new data was received.
82          """
83          has_new_incoming = False
84          try:
85              self.client_sock.settimeout(0.1)
86              data = self.client_sock.recv(1024).decode('utf-8')
87              if len(data) != 0:  # TODO or None? (using json)
88                  self.incoming_data = data
89                  has_new_incoming = True
90          except bluetooth.btcommon.BluetoothError:
91              pass
92          finally:
93              self.client_sock.settimeout(None)
94          return has_new_incoming
95
96      def update_outgoing(self):
97          """
98          Updates outgoing_data aimed for client by poping from robots queue.
99          Saves the new data in itself.
100         Returns true if data was updated, false otherwise
101         """
102         has_new_outgoing = False
103         task = self._pop_from_outgoing()
104         if type(task) == BT_task and task.cmd_id != 0:
105             self.outgoing_data = str(task.cmd_id) + ", " + str(task.data)
106             has_new_outgoing = True
107         return has_new_outgoing
108
109     def shutdown_server(self):
110         """
111         Shuts down itself by closing server and client sockets.
112         """
113         self.server_sock.shutdown(2)
114         self.server_sock.close()
115         self.client_sock.close()
116         print("Closed connection.")
```

```
1  ##############################################################################
2  #                                                                            #
3  #                              bt_server_cmds.py                             #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-11-28                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                               Matildha Sjöstedt                            #
10 #                                                                            #
11 ##############################################################################
12
13 import os
14
15 """
16 File for miscellaneous functions.
17 Functions can be moved if no need of file otherwise.
18 """
19
20
21 def get_pi_ip():
22     """
23     Returns IP address for Rasberry Pi connected to Eduroam.
24     """
25     s = os.popen('ifconfig wlan0 | grep "inet\ addr" | cut -d: -f2 | cut -d" " -
       f1')
26     pi_ip = s.read()
27     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()

```

```
66      if has_new_incoming:
67          if int(server.incoming_data) == protocol.BT_SERVER_RESTART:
68              return protocol.BT_SERVER_RESTART
69          elif int(server.incoming_data) == protocol.BT_SERVER_SHUTDOWN:
70              return protocol.BT_SERVER_SHUTDOWN
71          else:
72              server.post_to_incoming()
73              return GOT_DATA
74      return NO_DATA
75
76
77  def main():
78      """
79      The main function initializes instance of the server.
80      Runs the main control of data flow in the bluetooth connection.
81      """
82      server = setup_server()
83      exit = NO_DATA
84      while exit != protocol.BT_SERVER_SHUTDOWN:
85          exit = recieve(server)
86          send(server)
87          if exit == protocol.BT_SERVER_RESTART or exit == protocol.
        BT_SERVER_SHUTDOWN:
88              has_sent = False
89              while has_sent:
90                  has_sent = send(server)
91
92              server.shutdown_server()
93              del server
94
95              if exit == protocol.BT_SERVER_RESTART:
96                  server = setup_server()
97                  exit = NO_DATA
98                  # Breaks if exit == SHUTDOWN
99
100
101 main()
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                  bt_task.py                                #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                           Senast modifierad 2016-11-28                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                                                                            #
10 ##############################################################################
11
12 class BT_task:
13     """
14     Class for packing command ID and data when sending
15     commands and corresponding answers over Bluetooth.
16     """
17
18     def __init__(self, cmd_id=0, data=0):
19         self.cmd_id = int(cmd_id)
20         self.data = data
```

```
1  ##############################################################################
2  #                                                                            #
3  #                              bt_task_handler.py                            #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-11-28                     #
7  #                                                                            #
8  #                                Rebecca Lindblom                            #
9  #                               Matildha Sjöstedt                            #
10 #                                                                            #
11 ##############################################################################
12
13 import pickle
14 from bt_task import BT_task
15
16
17 def clean_queue_files():
18     """
19     Creates clean files for queues between server and main unit.
20     """
21     answer_queue = open("bt_answers.txt", "w")
22     answer_queue.seek(0)
23     answer_queue.truncate()
24     command_queue = open("bt_commands.txt", "w")
25     command_queue.seek(0)
26     command_queue.truncate()
27     answer_queue.close()
28     command_queue.close()
29
30
31 def post_outgoing(task):
32     """
33     Called from main unit.
34     Posts BT_task processed by main unit to answer queue.
35     """
36     answer_queue = open("bt_answers.txt", "wb")
37     pickle.dump(task, answer_queue)
38     answer_queue.close()
39
40
41 def pop_incoming():
42     """
43     Called from main unit.
44     Pops and returns next BT_task from commands queue,
45     to be processed by the main unit.
46     """
47     command_queue = open("bt_commands.txt", "rb")
48     task = None
49
50     task_q = []
51     while (True):
52         try:
53             task_i = pickle.load(command_queue)
54             task_q.append(task_i)
55         except EOFError:
56             break
57     if task_q:
58         task = BT_task(task_q[0].cmd_id, task_q[0].data)
59         del task_q[0]
60         command_queue = open("bt_commands.txt", "wb")
61         for task_i in task_q:
62             pickle.dump(task_i, command_queue)
63
64     command_queue.close()
65     return task
```

```
66
67
68  def post_incoming(task):
69      """
70      Called from server.
71      Posts given BT_task to command queue,
72      to be processed by main unit.
73      """
74      command_queue = open("bt_commands.txt", "wb")
75      pickle.dump(task, command_queue)
76      command_queue.close()
77
78
79  def pop_outgoing():
80      """
81      Called from server.
82      Pops next outgoing BT_task from answer queue,
83      tasks already processed by main unit.
84      Returns popped task.
85      """
86      answer_queue = open("bt_answers.txt", "rb")
87      task = None
88
89      task_q = []
90      while (True):
91          try:
92              task_i = pickle.load(answer_queue)
93              task_q.append(task_i)
94          except EOFError:
95              break
96      if task_q:
97          task = BT_task(task_q[0].cmd_id, task_q[0].data)
98          del task_q[0]
99          answer_queue = open("bt_answers.txt", "wb")
100         for task_i in task_q:
101             pickle.dump(task_i, answer_queue)
102
103     answer_queue.close()
104     return task
```

```
1  ################################################################################
2  #                                                                              #
3  #                                    bus.py                                    #
4  #                                                                              #
5  #                                Version 1.0                                   #
6  #                           Senast modifierad 2016-11-16                       #
7  #                                                                              #
8  #                                Patrik Sletmo                                 #
9  #                                                                              #
10 ################################################################################
11
12 """
13 Wrapper for the I2C bus with added functionality to support the packet protocol
14 the robot is using.
15
16 Packet protocol
17 ----
18 I2C supports two functions for interacting on the bus:
19   Read address,
20   Write address
21
22 There functions are in turn addressed to specific slaves and contain the
23 address requested/set and a value when writing data. Each function write or
24 read a single byte.
25
26 As the desired functionality requires data to be sent and received in
27 multi-byte chunks this functionality must be abstracted away using a looser
28 protocol. To accomplish this the read and written addresses are purposely used
29 incorrectly to specify different states of receiving or sending data.
30
31 The protocol use two "addresses":
32   PACKET_HEADER,
33   PACKET_DATA
34
35 Using these two addresses it is possible to send data of the length 255 bytes
36 by first sending the data length and then all bytes in the data array until
37 every byte has been sent.
38
39 PACKET_HEADER:
40 Contains the length of the following data on the range 0 to 255. If there is no
41 data to be sent the value returned is 0.
42
43 PACKET_DATA:
44 Contains the data of the n:th read byte since the PACKET_HEADER. Reading data
45 from a packet which has already been read to the end has undefined behaviour.
46
47 The master-slave problem
48 ----
49 As requests to read and write data can only be made from the master unit there
50 are difficulties actually notifying of new data from a slave unit. The protocol
51 solves this by reading the "addresses" periodically in order to search for
52 pending data. When data read from PACKET_HEADER is not zero there is data
53 available and the program can then read its data.
54 """
55
56 from busprovider import WIRED_BUS
57 from protocol import PACKET_HEADER, PACKET_DATA
58
59
60 class Bus:
61     def __init__(self, interface=1):
62         self.interface = interface
63         self.bus = WIRED_BUS
64
65     def send(self, data, unit_addr):
```

```
66          self._write_packet_start(len(data), unit_addr)
67          self._write_packet_data(data, unit_addr)
68
69      def try_receive(self, unit_addr):
70          size = self._get_pending_packet_size(unit_addr)
71
72          if size == 0:
73              # No pending packet
74              return None
75
76          return self._read_packet_data(size, unit_addr)
77
78      # Internal methods
79
80      def _write_packet_start(self, packet_len, unit_addr):
81          self.bus.write_byte_data(unit_addr, PACKET_HEADER, packet_len)
82
83      def _write_packet_data(self, packet_data, unit_addr):
84          for b in packet_data:
85              self.bus.write_byte_data(unit_addr, PACKET_DATA, b)
86
87      def _read_packet_data(self, packet_len, unit_addr):
88          data = []
89          for i in range(packet_len):
90              data.append(self.bus.read_byte_data(unit_addr, PACKET_DATA))
91
92          return data
93
94      def _get_pending_packet_size(self, addr):
95          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()
```

```
1  ################################################################################
2  #                                                                              #
3  #                            command_processors.py                             #
4  #                                                                              #
5  #                                 Version 1.0                                  #
6  #                           Senast modifierad 2016-12-01                       #
7  #                                                                              #
8  #                                Rebecca Lindblom                              #
9  #                                 Patrik Sletmo                                #
10 #                                                                              #
11 ################################################################################
12
13 """
14 Contains various pre-notify command processors used to parse received data into
15 more usable form than single bytes.
16 """
17 from protocol import CMD_RETURN_SENSOR_DATA
18 from utils import twos_comp
19
20 COMMAND_PROCESSORS = {}
21
22
23 def process_arguments(message_id, arguments):
24     if message_id in COMMAND_PROCESSORS:
25         return COMMAND_PROCESSORS[message_id](*arguments)
26
27     return arguments
28
29
30 # Sensor data contains some arguments stored in 16-bit two's complement which
31 # must be parsed into it's corresponding python values.
32
33 def process_sensor_data(left_ir_mm_hi, left_ir_mm_lo,
34                         right_ir_mm_hi, right_ir_mm_lo,
35                         right_ir_back_mm_hi, right_ir_back_mm_lo,
36                         left_ir_back_mm_hi, left_ir_back_mm_lo, *args):
37     left_ir_mm = twos_comp((left_ir_mm_hi << 8) | left_ir_mm_lo, 16)
38     right_ir_mm = twos_comp((right_ir_mm_hi << 8) | right_ir_mm_lo, 16)
39     right_ir_back_mm = twos_comp((right_ir_back_mm_hi << 8) |
       right_ir_back_mm_lo, 16)
40     left_ir_back_mm = twos_comp((left_ir_back_mm_hi << 8) | left_ir_back_mm_lo,
       16)
41
42     return [left_ir_mm, right_ir_mm, right_ir_back_mm, left_ir_back_mm] + list(
       args)
43
44
45 COMMAND_PROCESSORS[CMD_RETURN_SENSOR_DATA] = process_sensor_data
```

```
1  ################################################################################
2  #                                                                              #
3  #                                communicator.py                               #
4  #                                                                              #
5  #                                 Version 1.0                                  #
6  #                           Senast modifierad 2016-12-06                       #
7  #                                                                              #
8  #                                Sebastian Callh                               #
9  #                               Matilda Dahlström                              #
10 #                                 Anton Dalgren                                #
11 #                                Rebecca Lindblom                              #
12 #                               Matildha Sjöstedt                              #
13 #                                                                              #
14 ################################################################################
15
16 import outbound
17
18 from utils import get_ip
19
20
21 class Communicator:
22     def __init__(self, ir, laser, gyro, driver, navigator, position):
23         self.ir = ir
24         self.laser = laser
25         self.gyro = gyro
26         self.driver = driver
27         self.navigator = navigator
28         self.position = position
29
30     '''
31     Sends the sensor data as a string of integers on the format
32     "ir_left, ir_left_back, ir_right, ir_right_back, laser, gyro"
33     '''
34     def send_sensor_data(self):
35         outbound.bt_return_sensor_data(str([self.ir.get_ir_left()] +
36                                            [self.ir.get_ir_left_back()] +
37                                            [self.ir.get_ir_right()] +
38                                            [self.ir.get_ir_right_back()] +
39                                            [self.laser.get_data()] +
40                                            [self.gyro.get_data()]))
41
42     '''
43     Sends the sensor data as a string of integers on the format
44     "left_speed, right_speed"
45     '''
46     def send_servo_data(self):
47         outbound.bt_return_servo_data(str([self.driver.get_left_speed()] +
48                                           [self.driver.get_right_speed()]))
49
50     '''
51     Sends the map data as a list with tuples of integers corresponding to corner
        coordinates on the format
52     "[(X1, Y1), (X2, Y2), ... , (Xn, Yn)]"
53     '''
54     def send_map_data(self):
55         map_data = self.position.get_map_data()
56         outbound.bt_return_map_data(str(self.position.get_map_data()))
57
58     def send_ip(self):
59         outbound.bt_return_ip([get_ip()])
60         print("send_ip, done")
61
62     def drive_forward(self):
63         self.navigator.drive_forward()
64         print("Communicator drove forward!")
```

```
65
66      def drive_backward(self):
67          self.navigator.drive_backward()
68
69      def drive_forward_right(self):
70          self.navigator.drive_forward_right()
71
72      def drive_forward_left(self):
73          self.navigator.drive_forward_left()
74
75      def turn_left(self):
76          self.navigator.turn_left()
77
78      def turn_right(self):
79          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
```

```
60          self.tasks = []
61          self.task = Task(None, lambda: True)
62          self.gyro = gyro
63          self.laser = laser
64          self.right_speed = 0
65          self.left_speed = 0
66
67      #Returns true only if we have executed all provided tasks
68      def idle(self):
69          return self.task.done() and not self.tasks
70
71      #Expected to be run every main loop. Checks whether the current task is
72      #done, and if there are more to perform, starts performing them.
73      def update(self):
74          if self.task.done():
75              if self.tasks:
76                  self.task = self.tasks.pop()
77                  print("Next task: " + str(self.task))
78                  self.task.start()
79              else:
80                  #print("STANNA")
81                  self.stop()
82
83      def drive(self, left_speed, right_speed):
84          self.left_speed = left_speed
85          self.right_speed = right_speed
86          set_motor_speed(left_speed, right_speed)
87          # print("Driver drive set motor speed to ", left_speed, right_speed)
88
89      def outer_turn_right(self):
90          print('outer turn right')
91
92          current_degree = math.degrees(math.atan(autocontroller.last_valid_diff /
     165))
93          degree = TURN_DEGREES - current_degree
94
95          # We won't need the value of last_valid_diff any longer, so reset it to
96          # avoid rotating too far or too little in dead ends (not needed here)
97          print('Last valid diff:', autocontroller.last_valid_diff)
98          autocontroller.last_valid_diff = 0
99
100         print('Current degree:', current_degree)
101         print('Turning degree:', degree)
102
103         self.task = Task(None, lambda: True)
104         self.tasks = [DegreeTask(self._turn_right, degree, self.gyro)]
105
106     def outer_turn_left(self):
107         print('outer turn left')
108         self.task = Task(None, lambda: True)
109         self.tasks = [DistanceTask(self._post_turn, POST_TURN_DISTANCE, self.
    laser),
110                       DegreeTask(self._turn_left, TURN_DEGREES, self.gyro),
111                       DistanceTask(self._pre_turn, PRE_TURN_DISTANCE, self.laser
    )]
112
113     def inner_turn_left(self):
114         print('inner turn left')
115         current_degree = math.degrees(math.atan(autocontroller.last_diff / 165))
116
117         # Over turn for dead ends
118         if autocontroller.last_diff == 0:
119             current_degree = 2
120
121         degree = TURN_DEGREES + current_degree
```

```
122
123         # We won't need the value of last_diff any longer, so reset it to avoid
124         # rotating too far or too little in dead ends
125         autocontroller.last_diff = 0
126
127         print('Current degree:', current_degree)
128         print('Turning degree:', degree)
129
130         self.task = Task(None, lambda: True)
131         self.tasks = [DegreeTask(self._turn_left, degree, self.gyro)]
132
133     def inner_turn_right(self):
134         print('inner turn right')
135         self.task = Task(None, lambda: True)
136         self.tasks = [DegreeTask(self._turn_right, TURN_DEGREES, self.gyro)]
137
138     def warmup(self):
139         print('warming up')
140         self.tasks = [TimedTask(lambda: self.drive(0, 0), WARMUP_TIME)]
141
142     def stop(self):
143         #print('stopping')
144         self.drive(0, 0)
145
146     def get_right_speed(self):
147         return self.right_speed
148
149     def get_left_speed(self):
150         return self.left_speed
151
152
153     # Commands intended to be called while remote controlling
154
155     def drive_forward(self):
156         self.task = TimedTask(self._drive_forward, REMOTE_COMMAND_EXECUTE_TIME)
157         self.task.start()
158
159     def drive_backward(self):
160         self.task = TimedTask(self._drive_backward, REMOTE_COMMAND_EXECUTE_TIME)
161         self.task.start()
162
163     def turn_left(self):
164         self.task = TimedTask(self._turn_left, REMOTE_COMMAND_EXECUTE_TIME)
165         self.task.start()
166
167     def turn_right(self):
168         self.task = TimedTask(self._turn_right, REMOTE_COMMAND_EXECUTE_TIME)
169         self.task.start()
170
171     def drive_forward_right(self):
172         self.task = TimedTask(self._drive_forward_right,
    REMOTE_COMMAND_EXECUTE_TIME)
173         self.task.start()
174
175     def drive_forward_left(self):
176         self.task = TimedTask(self._drive_forward_left,
    REMOTE_COMMAND_EXECUTE_TIME)
177         self.task.start()
178
179     # Not intended for public use
180
181     def _turn_left(self):
182         print('turn left')
183         self.drive(-TURN_SPEED, TURN_SPEED)
184
```

```
185     def _turn_right(self):
186         print('turn right')
187         self.drive(TURN_SPEED, -TURN_SPEED)
188 
189     def _drive_forward(self):
190         print('drive forward')
191         self.drive(STANDARD_SPEED, STANDARD_SPEED)
192 
193     def _drive_backward(self):
194         print('drive backward')
195         self.drive(-STANDARD_SPEED, -STANDARD_SPEED)
196 
197     def _drive_forward_right(self):
198         print('drive forward right')
199         self.drive(FAST_SPEED, SLOW_SPEED)
200 
201     def _drive_forward_left(self):
202         print('drive backward left')
203         self.drive(SLOW_SPEED, FAST_SPEED)
204 
205     def _post_turn(self):
206         print('post turn')
207         self.drive(STANDARD_SPEED, STANDARD_SPEED)
208 
209     def _pre_turn(self):
210         print('pre turn')
211         self.drive(STANDARD_SPEED, STANDARD_SPEED)
212 
213 
214 class Task:
215     def __init__(self, task_func, done_func):
216         self.task_func = task_func
217         self.done_func = done_func
218 
219     def start(self):
220         self.task_func()
221 
222     def done(self):
223         return self.done_func()
224 
225 
226 class TimedTask(Task):
227     def __init__(self, task_func, duration):
228         Task.__init__(self, task_func, self.timed_task)
229         self.duration = duration
230         self.stop_time = datetime.now()
231 
232     def start(self):
233         Task.start(self)
234         self.stop_time = datetime.now() + timedelta(milliseconds=self.duration)
235         print("Start timed task")
236 
237     def timed_task(self):
238         #print('time left driving: ' + str(self.stop_time - datetime.now()))
239         return self.stop_time <= datetime.now()
240 
241 
242 class DegreeTask(Task):
243     def __init__(self, task_func, degrees, gyro):
244         Task.__init__(self, task_func, self.degree_task)
245         self.total_degrees = 0
246         self.previous_time = datetime.now()
247         self.degrees = degrees
248         self.gyro = gyro
249 
```

```
250     def start(self):
251         self.total_degrees = 0
252         self.previous_time = datetime.now()
253         Task.start(self)
254 
255 
256     def degree_task(self):
257         data = self.gyro.get_data()
258 
259         if data == -1:
260             raise Exception('Error reading gyro')
261 
262         time_delta = (self.previous_time - datetime.now()).total_seconds()
263         delta_degrees = data * time_delta
264         #print("time_delta: " + str(time_delta))
265         #print("delta_degrees: " + str(delta_degrees))
266         self.previous_time = datetime.now()
267         self.total_degrees += delta_degrees
268         #print('total degrees turned :' + str(self.total_degrees))
269 
270         return abs(self.total_degrees) >= self.degrees
271 
272 
273 class DistanceTask(Task):
274     def __init__(self, task_func, distance, laser):
275         Task.__init__(self, task_func, self.distance_task)
276         self.destination = 0
277         self.previous_time = datetime.now()
278         self.distance = distance
279         self.laser = laser
280 
281     def start(self):
282         laser_data = -1
283         while laser_data == -1:
284             laser_data = self.laser.get_data()
285             print("RUN RUN RUN LASER READINGS")
286 
287         #print("Distance: " + str(self.distance))
288         #print("Laser data: " + str(laser_data))
289 
290         self.destination = laser_data - self.distance
291         #print("Destination: " + str(self.destination))
292         self.previous_time = datetime.now()
293         Task.start(self)
294 
295     def distance_task(self):
296         laser_data = -1
297         while laser_data == -1:
298             laser_data = self.laser.get_data()
299 
300         return self.destination >= laser_data
```

```
1  ##############################################################################
2  #                                                                            #
3  #                              emulated_bus.py                               #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-11-17                      #
7  #                                                                            #
8  #                                Patrik Sletmo                               #
9  #                                                                            #
10 ##############################################################################
11
12 import logging
13 from queue import Queue
14
15 from protocol import CMD_REQUEST_SENSOR_DATA, CMD_RETURN_SENSOR_DATA, STYR_ADDR,
       \
16     SENSOR_ADDR, LASER_ADDR, PACKET_HEADER, PACKET_DATA
17
18 log = logging.getLogger(__name__)
19
20
21 class EmulatedBus:
22     def __init__(self):
23         self.slaves = {
24             SENSOR_ADDR: EmulatedSlave(SENSOR_ADDR),
25             STYR_ADDR: EmulatedSlave(STYR_ADDR),
26             LASER_ADDR: EmulatedLaser
27         }
28
29     def write_byte_data(self, address, data_id, data):
30         if address in self.slaves:
31             self.slaves[address].write_byte_data(data_id, data)
32         else:
33             log.error('Trying to send data to nonexistant slave.')
34
35     def read_byte_data(self, address, data_id):
36         if address in self.slaves:
37             return self.slaves[address].read_byte_data(data_id)
38         else:
39             log.error(
40                 'Trying to read data from nonexistant slave (address = {}).'
41                 .format(address)
42             )
43             return None
44
45
46 class EmulatedSlave:
47     def __init__(self, address):
48         self.address = address
49         self.data_queue = Queue()
50
51         # Receiving
52         self.read_packet = None
53         self.bytes_to_read = 0
54         self.current_read_byte = 0
55
56         # Transmitting
57         self.transmitted_packet = None
58         self.current_transmitted_byte = 0
59
60     def write_byte_data(self, event_type, data):
61         if event_type == PACKET_HEADER:
62             if self.read_packet is not None:
63                 log.error(
64                     'Data header incorrectly sent to address {} '
```

```
65                      'which still waits for more bytes'
66                      .format(self.address)
67                  )
68
69              self.read_packet = []
70              self.bytes_to_read = data
71              self.current_read_byte = 0
72          elif event_type == PACKET_DATA:
73              if self.read_packet is None:
74                  log.error(
75                      'Data incorrectly sent to address {} '
76                      'without preceding packet header'
77                      .format(self.address)
78                  )
79
80              self.read_packet.append(data)
81              self.current_read_byte += 1
82
83              if self.current_read_byte >= self.bytes_to_read:
84                  if self.read_packet[0] == CMD_REQUEST_SENSOR_DATA:
85                      self.data_queue.put([CMD_RETURN_SENSOR_DATA, 0, 0, 0, 0])
86
87                  self.read_packet = None
88
89      def read_byte_data(self, data_id):
90          if data_id == PACKET_HEADER:
91              if self.transmitted_packet is not None:
92                  log.error(
93                      'Data header incorrectly requested from address {} '
94                      'which still sends bytes'
95                      .format(self.address)
96                  )
97
98              if self.data_queue.empty():
99                  return 0
100
101             self.transmitted_packet = self.data_queue.get()
102             self.current_transmitted_byte = 0
103
104             return len(self.transmitted_packet)
105         elif data_id == PACKET_DATA:
106             if self.transmitted_packet is None:
107                 log.error(
108                     'Data incorrectly requested from address {} '
109                     'without preceding packet header'
110                     .format(self.address)
111                 )
112
113             data = self.transmitted_packet[self.current_transmitted_byte]
114             self.current_transmitted_byte += 1
115
116             if self.current_transmitted_byte >= len(self.transmitted_packet):
117                 self.transmitted_packet = None
118
119             return data
120
121
122 class EmulatedLaser:
123     def write_byte_data(self, *args):
124         pass
125
126     def read_byte_data(self, *args):
127         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
```

```
################################################################################
#                                                                              #
#                                  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():
```

```
66          EventBus.receive_from_addr(BLUETOOTH_ADDR)
67          EventBus.receive_from_addr(SENSOR_ADDR)
68          EventBus.receive_from_addr(STYR_ADDR)
69
70      @staticmethod
71      def receive_from_addr(unit_addr):
72          for i in range(MAX_READ_COUNT):
73              data = EventBus.pop(unit_addr)
74              if data is None:
75                  break
76
77              if unit_addr == BLUETOOTH_ADDR:
78                  EventBus.notify(data.cmd_id, *data.data)
79              else:
80                  command = Event.parse(data)
81                  command.process()
82
83                  EventBus.notify(command.message_id, *command.arguments)
84
85      @staticmethod
86      def subscribe(command_id, handler):
87          observer = EventBus._get_observer_for_command(command_id)
88          observer.subscribe(handler)
89
90      @staticmethod
91      def notify(command_id, *args):
92          observer = EventBus._get_observer_for_command(command_id)
93          observer.notify(*args)
94
95      @staticmethod
96      def _get_observer_for_command(command_id):
97          if command_id not in EventBus.observers:
98              EventBus.observers[command_id] = Observer()
99
100         return EventBus.observers[command_id]
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                    gyro.py                                 #
4  #                                                                            #
5  #                                  Version 1.0                               #
6  #                            Senast modifierad 2016-11-28                    #
7  #                                                                            #
8  #                                 Sebastian Callh                            #
9  #                                  Anton Dalgren                             #
10 #                                                                            #
11 ##############################################################################
12
13 from time import sleep
14
15 from protocol import GYRO_ADDR
16 from eventbus import EventBus
17 from utils import twos_comp
18
19 GYRO_LOWER_LIMIT = 10
20
21
22 class Gyro:
23     def __init__(self):
24         self.data = 0
25
26
27     def get_data(self):
28         return self.data
29
30     def read_data(self):
31         try:
32             hi = EventBus.bus.bus.read_byte_data(GYRO_ADDR, 0x2d)
33             lo = EventBus.bus.bus.read_byte_data(GYRO_ADDR, 0x2c)
34             data = (hi << 8) | lo
35
36             # Divided by gyro sensitivity 18/256 for 2000 dps
37             two_comp_data = 18 * twos_comp(data, 16) / 256
38
39             # To prevent garbage values while standing still
40             if abs(two_comp_data) <= GYRO_LOWER_LIMIT:
41                 self.data = 0
42
43             self.data = two_comp_data
44
45         except:
46             self.data = -1
47
48     @staticmethod
49     def initialize():
50         # Set the PD flag to 1 to go from power-down mode to normal mode
51         EventBus.bus.bus.write_byte_data(GYRO_ADDR, 0x20, 0x0F)
52         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)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                    ir.py                                   #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                           Senast modifierad 2016-12-01                     #
7  #                                                                            #
8  #                                 Sebastian Callh                            #
9  #                                Matilda Dahlström                           #
10 #                                  Anton Dalgren                             #
11 #                                                                            #
12 ##############################################################################
13
14 from datetime import datetime, timedelta
15 import outbound
16
17 class IR:
18     def __init__(self):
19         self.busy = False
20         self.ir_left = 0
21         self.ir_left_back = 0
22         self.ir_right = 0
23         self.ir_right_back = 0
24         self.last_request = datetime.now()
25         self.request_period = timedelta(milliseconds=1)
26
27     def get_ir_left(self):
28         return self.ir_left
29
30     def get_ir_left_back(self):
31         return self.ir_left_back
32
33     def get_ir_right(self):
34         return self.ir_right
35
36     def get_ir_right_back(self):
37         return self.ir_right_back
38
39     def sensor_data_received(self, ir_left_mm, ir_right_mm, ir_right_back_mm,
       ir_left_back_mm):
40         self.ir_left = ir_left_mm
41         self.ir_right = ir_right_mm
42         self.ir_right_back = ir_right_back_mm
43         self.ir_left_back = ir_left_back_mm
44         self.busy = False
45
46     def request_data(self):
47         if not self.busy and datetime.now() - self.last_request > self.
       request_period:
48             self.busy = True
49             self.last_request = datetime.now()
50             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')
```

```
66              self.debug_file.flush()
67
68      def reset(self):
69          print('resetting laser (but actually not)')
70
71      @staticmethod
72      def initialize():
73          # Was bus.write_byte_data, but that method was renamed/removed
74          EventBus.bus.bus.write_byte_data(LASER_ADDR, 0x00, 0x00)  # Resets FPGA
    registers
75          sleep(1)
76          EventBus.bus.bus.write_byte_data(LASER_ADDR, 0x11, 0xff)  # sets laser
    to read forever
77          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)
```

```
131         self._send(DISPLAY_CLEAR)
132
133
134     def reset_cursor(self):
135         GPIO.output(RS, 0)
136         self._send(RESET_CURSOR)
137
138
139     def send(self, data):
140         self.clear()
141         self.reset_cursor()
142
143         #ugly ugly
144         self._send(self.bit_pattern(' '))
145         self._send(self.bit_pattern(' '))
146         self._send(self.bit_pattern(' '))
147
148         GPIO.output(RS, 1)    #Select data register
149
150         for d in data.replace('\n', ''):
151             self._send(self.bit_pattern(d))
152
153
154     def bit_pattern(self, char):
155         return BIT_PATTERN[char.upper()]
156
157
158     def _send(self, data):
159         GPIO.output(E, 0)     #Make sure E is initially low
160
161         #Put the data on the pins
162         for d, p in zip(data, PINS):
163             GPIO.output(p, d)
164
165         #Write the data
166         GPIO.output(E, 1)
167         sleep_us(1)       #Larger than recommended wait
168         GPIO.output(E, 0)
169         sleep_us(1)       #Larger than recommended wait
170
171
172     def cleanup(self):
173         GPIO.cleanup()
174
175
176 #Sleep in ms
177 def sleep_ms(t):
178     sleep(t / 1000)
179
180
181 #Sleep in us
182 def sleep_us(t):
183     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()
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                  main.py                                   #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                           Senast modifierad 2016-12-18                     #
7  #                                                                            #
8  #                                Sebastian Callh                             #
9  #                               Matilda Dahlström                            #
10 #                                 Anton Dalgren                              #
11 #                                Rebecca Lindblom                            #
12 #                               Matildha Sjöstedt                            #
13 #                                 Patrik Sletmo                              #
14 #                                                                            #
15 ##############################################################################
16
17 """
18 Main program code - where all the magic happens
19 """
20
21 from datetime import datetime, timedelta
22
23 from navigator import Navigator
24 from driver import Driver
25 from laser import Laser
26 from gyro import Gyro
27 from ir import IR
28 from communicator import Communicator
29
30 from eventbus import EventBus
31 from position import Position
32
33 from protocol import *
34 from safety import Safety
35
36 ir = IR()
37 laser = Laser()
38 gyro = Gyro()
39 driver = Driver(gyro, laser)
40 navigator = Navigator(Navigator.MANUAL, ir, driver, laser)
41 position = Position(laser, ir, navigator)
42 communicator = Communicator(ir, laser, gyro, driver, navigator, position)
43
44
45 def setup():
46     Safety.setup_terminal_abort()
47     EventBus.subscribe(BT_REQUEST_SENSOR_DATA, communicator.send_sensor_data)
48     EventBus.subscribe(BT_REQUEST_SERVO_DATA, communicator.send_servo_data)
49     EventBus.subscribe(BT_REQUEST_MAP_DATA, communicator.send_map_data)
50     EventBus.subscribe(BT_DRIVE_FORWARD, communicator.drive_forward)
51     EventBus.subscribe(BT_DRIVE_BACK, communicator.drive_backward)
52     EventBus.subscribe(BT_TURN_RIGHT, communicator.turn_right)
53     EventBus.subscribe(BT_TURN_LEFT, communicator.turn_left)
54     EventBus.subscribe(BT_DRIVE_FORWARD_RIGHT, communicator.drive_forward_right)
55     EventBus.subscribe(BT_DRIVE_FORWARD_LEFT, communicator.drive_forward_left)
56     EventBus.subscribe(AUTONOMOUS_MODE, (lambda: navigator.set_mode(Navigator.
       AUTONOMOUS)))
57     EventBus.subscribe(MANUAL_MODE, (lambda: navigator.set_mode(Navigator.MANUAL
       )))
58     EventBus.subscribe(CMD_TOGGLE_MODE, navigator.toggle_mode)
59     EventBus.subscribe(REQUEST_PI_IP, communicator.send_ip)
60     EventBus.subscribe(CMD_RETURN_SENSOR_DATA, ir.sensor_data_received)
61     Laser.initialize()
62     Gyro.initialize()
63
```

```
64
65 def main():
66     setup()
67     while True:
68         laser.read_data()
69         gyro.read_data()
70         ir.request_data()
71
72         EventBus.receive()
73         position.update()
74         navigator.navigate()
75
76 Safety.run_safely(main)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                navigator.py                                #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-12-18                      #
7  #                                                                            #
8  #                               Sebastian Callh                              #
9  #                              Matilda Dahlström                             #
10 #                                Anton Dalgren                               #
11 #                              Matildha Sjöstedt                             #
12 #                                Patrik Sletmo                               #
13 #                                                                            #
14 ##############################################################################
15
16 """
17 State machine for navigational states
18 """
19 from datetime import datetime, timedelta
20
21 from autocontroller import AutoController
22 from eventbus import EventBus
23 from outbound import notify_mode_changed
24 from protocol import CMD_TURN_STARTED, CMD_TURN_FINISHED
25
26 TURN_DIRECTION_RIGHT = True
27 TURN_DIRECTION_LEFT = False
28 UPDATE_FREQUENCY = 50
29
30
31 class State:
32     def run(self, data):
33         assert 0, "run not implemented"
34
35
36 class AutoControl(State):
37     auto_controller = AutoController()
38
39     def is_at_right_turn(self, data):
40         return data['ir'].get_ir_right() == -1 and \
41                data['ir'].get_ir_right_back() == -1 and \
42                Navigator.right_turn_enabled
43
44     def run(self, data):
45         if self.is_at_right_turn(data):
46             data['driver'].outer_turn_right()
47             print('NAVIGATOR: outer turn right')
48             return Turn(TURN_DIRECTION_RIGHT)
49
50         laser_data = data['laser'].get_data()
51
52         if not Navigator.right_turn_enabled:
53             Navigator.right_turn_enabled = (data['ir'].get_ir_right() != -1 and
    data['ir'].get_ir_right_back() != -1)
54
55         # Inner turn
56         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)):
57             Navigator.force_left_turn = False
58             if data['side'] == Navigator.RIGHT_SIDE:
59                 data['driver'].inner_turn_left()
60                 print('NAVIGATOR: inner turn left')
61                 return Turn(TURN_DIRECTION_LEFT)
62
```

```
63          right_speed, left_speed, regulation = AutoControl.auto_controller.
    auto_control(data['ir'].get_ir_right(),
64
                data['ir'].get_ir_right_back(),
65
                data['side'])
66          data['driver'].drive(left_speed, right_speed)
67
68          return self
69
70
71  class Warmup(State):
72      def run(self, data):
73          if data['driver'].idle():
74              print('NAVIGATOR: changin to auto control')
75              return AutoControl()
76          else:
77              return self
78
79
80  class Turn(State):
81      def __init__(self, is_right_turn):
82          self.is_right_turn = is_right_turn
83
84      def run(self, data):
85          if Navigator.right_turn_enabled:
86              Navigator.right_turn_enabled = False
87
88          if data['driver'].idle():
89              print('NAVIGATOR: changing to auto control')
90              return Stabilize(self.is_right_turn)
91          else:
92              return self
93
94  class Stabilize(State):
95      def __init__(self, is_right_turn):
96          self.is_right_turn = is_right_turn
97          self.angle_threshold = 2
98          self.speed_scaling = 26
99
100     def run(self, data):
101         if self.is_right_turn or Navigator.force_left_turn:
102             data['laser'].reset()
103             return AutoControl()
104         else:
105             ir_front = data['ir'].get_ir_right()
106             ir_back = data['ir'].get_ir_right_back()
107             diff = ir_front - ir_back
108             print("Stabilize diff:", diff)
109             if abs(diff) < self.angle_threshold:
110                 data['laser'].reset()
111                 return AutoControl()
112
113             turn_speed = int(abs(diff)/diff*self.speed_scaling)
114             data['driver'].drive(turn_speed, -turn_speed)
115             return self
116
117 ###### NAVIGATOR CLASS #######
118 class Navigator:
119     LEFT_SIDE = 0
120     RIGHT_SIDE = 1
121
122     MANUAL = 0
123     AUTONOMOUS = 1
124
```

```
125     DISCONTINUITY_DIST = 25.0  # mm
126     FACING_WALL_DIST = 200  # mm
127 
128     right_turn_enabled = True
129     force_left_turn = False
130 
131     def __init__(self, mode, ir, driver, laser):
132         self.data = {
133             'driver': driver,
134             'laser': laser,
135             'ir': ir,
136             'side': Navigator.RIGHT_SIDE,
137         }
138         self.mode = mode
139         self.state = Warmup()
140         self.last_updated_time = datetime.now()
141 
142         # Stand still waiting for sensors
143         self.data['driver'].warmup()
144 
145     # Runs the state. The states run method returns the next state
146     def navigate(self):
147         self.data['driver'].update()
148         if self.mode == Navigator.AUTONOMOUS:
149             next_state = self.state.run(self.data)
150 
151             curr_type = type(self.state)
152             next_type = type(next_state)
153 
154             if curr_type is not Turn and next_type is Turn:
155                 EventBus.notify(CMD_TURN_STARTED)
156 
157             if curr_type is Stabilize and next_type is not Stabilize:
158                 EventBus.notify(CMD_TURN_FINISHED, self.state.is_right_turn)
159 
160             self.state = next_state
161 
162     def drive_forward(self):
163         if self.mode == Navigator.MANUAL:
164             self.data['driver'].drive_forward()
165 
166     def drive_backward(self):
167         if self.mode == Navigator.MANUAL:
168             self.data['driver'].drive_backward()
169 
170     def drive_forward_right(self):
171         if self.mode == Navigator.MANUAL:
172             self.data['driver'].drive_forward_right()
173 
174     def drive_forward_left(self):
175         if self.mode == Navigator.MANUAL:
176             self.data['driver'].drive_forward_left()
177 
178     def turn_left(self):
179         if self.mode == Navigator.MANUAL:
180             self.data['driver'].turn_left()
181 
182     def turn_right(self):
183         if self.mode == Navigator.MANUAL:
184             self.data['driver'].turn_right()
185 
186     def set_mode(self, mode):
187         self.mode = mode
188         notify_mode_changed(mode)
189 
```

```
190     def toggle_mode(self):
191         if self.mode == Navigator.MANUAL:
192             self.data['laser'].reset()
193             self.set_mode(Navigator.AUTONOMOUS)
194         else:
195             self.set_mode(Navigator.MANUAL)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                observer.py                                 #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-11-12                      #
7  #                                                                            #
8  #                               Patrik Sletmo                                #
9  #                                                                            #
10 ##############################################################################
11
12 """
13 Simple implementation of the observer pattern.
14
15 See https://en.wikipedia.org/wiki/Observer_pattern for more information.
16 """
17
18
19 class Observer:
20     def __init__(self):
21         self.subscribers = []
22
23     def subscribe(self, func):
24         if func not in self.subscribers:
25             self.subscribers.append(func)
26
27     def notify(self, *args, **kwargs):
28         for subscriber in self.subscribers:
29             subscriber(*args, **kwargs)
```

```
1  ################################################################################
2  #                                                                              #
3  #                                  outbound.py                                 #
4  #                                                                              #
5  #                                  Version 1.0                                 #
6  #                            Senast modifierad 2016-12-12                      #
7  #                                                                              #
8  #                                 Sebastian Callh                              #
9  #                                  Anton Dalgren                               #
10 #                                 Rebecca Lindblom                             #
11 #                                  Patrik Sletmo                               #
12 #                                                                              #
13 ################################################################################
14
15 """
16 This file contains functions for interacting with the two different AVR units.
17 All functions defined here are outbound which mean they go from the main unit
18 to one of the AVR units.
19
20 The message passing function is implemented as a distributed event bus which
21 in its distributed nature depends on asynchronous functionality. This means
22 that messages sent are not executed immediately and there is no guarantee that
23 the sent command is actually executed on the receiving unit.
24
25 For more information see eventbus.py.
26 """
27
28 from event import Event
29 from eventbus import EventBus
30 from protocol import *
31 from bt_task import BT_task
32
33
34 # NOTE: Function comments are purposely left out from this file in favor of the
35 # complete definitions of every found command in proctol.py.
36
37 def request_sensor_data():
38     EventBus.post(
39         SENSOR_ADDR,
40         Event(
41             message_id=CMD_REQUEST_SENSOR_DATA
42         )
43     )
44
45
46 def set_motor_speed(left_speed, right_speed=None):
47     if right_speed is None:
48         right_speed = left_speed
49
50     EventBus.post(
51         STYR_ADDR,
52         Event(
53             message_id=CMD_SET_MOTOR_SPEED,
54             arguments=[
55                 left_speed,
56                 right_speed
57             ]
58         )
59     )
60
61
62 def set_left_motor_speed(speed):
63     EventBus.post(
64         STYR_ADDR,
65         Event(
```

```
66              message_id=CMD_SET_LEFT_MOTOR_SPEED,
67              arguments=[
68                  speed
69              ]
70          )
71      )
72
73
74  def set_right_motor_speed(speed):
75      EventBus.post(
76          STYR_ADDR,
77          Event(
78              message_id=CMD_SET_RIGHT_MOTOR_SPEED,
79              arguments=[
80                  speed
81              ]
82          )
83      )
84
85
86  def bt_return_ip(ip):
87      EventBus.post(
88          BLUETOOTH_ADDR,
89          BT_task(
90              RETURN_PI_IP, ip
91          )
92      )
93
94
95  def bt_return_sensor_data(data):
96      EventBus.post(
97          BLUETOOTH_ADDR,
98          BT_task(
99              BT_RETURN_SENSOR_DATA, data
100         )
101     )
102
103
104 def bt_return_servo_data(data):
105     EventBus.post(
106         BLUETOOTH_ADDR,
107         BT_task(
108             BT_RETURN_SERVO_DATA, data
109         )
110     )
111
112
113 def bt_return_map_data(data):
114     EventBus.post(
115         BLUETOOTH_ADDR,
116         BT_task(
117             BT_RETURN_MAP_DATA, data
118         )
119     )
120
121
122 def notify_mode_changed(new_mode):
123     EventBus.post(
124         BLUETOOTH_ADDR,
125         BT_task(
126             CMD_MODE_SET, new_mode
127         )
128     )
```

```
##############################################################################
#                                                                            #
#                                 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)
```

```
66              if self.mapping_state == MAPPING_STATE_FOLLOWING_OUTER_WALL:
67                  self.looking_for_kitchen_sections()
68                  self.remove_invalid_kitchen_sections()
69              elif self.mapping_state == MAPPING_STATE_RETURNING_TO_ISLAND:
70                  self.check_if_returned_to_island()
71              elif self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND:
72                  self.current_section.finish()
73                  temporary_x, temporary_y = self.transform_map_data(self.
    current_section, self.current_x, self.current_y, estimate=True, offset=0.1)
74                  if temporary_x == self.kitchen_start_x and temporary_y == self.
    kitchen_start_y \
75                          and self.kitchen_num_mapped > 3:
76                      self.map_island_data()
77                      self.map_remaining_data()
78
79                      print(self.map_data)
80                      print(self.island_data)
81
82                      Navigator.force_left_turn = True
83
84                      # Last island section is not properly stored because of a
85                      # turn so we must store it here
86                      self.store_ongoing_section()
87
88                      self.mapping_state = MAPPING_STATE_RETURNING_TO_GARAGE
89
90      def looking_for_kitchen_sections(self):
91          if self.has_close_kitchen_left():
92              self.add_kitchen_mapping(1, 0.5)
93
94          if self.has_far_kitchen_left():
95              self.add_kitchen_mapping(2, 0.25)
96
97      def has_close_kitchen_left(self):
98          match = self.ir.get_ir_left() > 10
99          if match:
100             self.num_ok_close += 1
101         else:
102             self.num_ok_close = 0
103
104         if self.num_ok_close > 7:
105             self.num_ok_close = 0
106             return True
107         else:
108             return False
109
110     def has_far_kitchen_left(self):
111         match = 250 < self.ir.get_ir_left_back() < 650
112         if match:
113             self.num_ok_far += 1
114         else:
115             self.num_ok_far = 0
116
117         if self.num_ok_far > 7:
118             self.num_ok_far = 0
119             return True
120         else:
121             return False
122
123     def add_kitchen_mapping(self, displacement, offset):
124         distance = self.current_section.estimate_block_distance(offset)
125         cur_x, cur_y = self.transform_partial_map_data(distance, self.
    current_section.direction, self.current_x, self.current_y)
126         key = self.get_left_block_coordinates(cur_x, cur_y, displacement)
127
```

```
128         if key not in self.kitchen_mapping and key not in self.invalid_kitchens:
129             print('Adding new mapping (' + str(key[0]) + ', ' + str(key[1]) + ')
     -> (' + str(cur_x) + ', ' + str(cur_y) + ')')
130             self.kitchen_mapping[key] = (cur_x, cur_y)
131 
132     def remove_invalid_kitchen_sections(self):
133         temporary_x, temporary_y = self.transform_map_data(self.current_section,
134                                                            self.current_x,
135                                                            self.current_y)
136         key = self.get_right_block_coordinates(temporary_x, temporary_y, 1)
137         popped = self.kitchen_mapping.pop(key, None)
138         if popped is not None:
139             print('Removed kitchen mapping for block (' + str(key[0]) + ', ' +
    str(key[1]) + ')')
140 
141         if key not in self.invalid_kitchens:
142             print('Forbidding kitchen block at ' + str(key))
143             self.invalid_kitchens.append(key)
144 
145     def check_if_returned_to_island(self):
146         coordinates = self.transform_map_data(self.current_section,
147                                               self.current_x,
148                                               self.current_y,
149                                               estimate=True)
150 
151         if coordinates in self.kitchen_mapping.values():
152             print('Starting to map island!')
153 
154             self.mapping_state = MAPPING_STATE_FOLLOWING_ISLAND
155             self.kitchen_num_mapped = 0
156             self.num_kitchen_turns = 0
157             key1 = self.transform_partial_map_data(1, self.current_section.
    direction, coordinates[0], coordinates[1])
158             key2 = self.transform_partial_map_data(2, self.current_section.
    direction, coordinates[0], coordinates[1])
159             if key1 not in self.kitchen_mapping and key2 not in self.
    kitchen_mapping:
160                 Navigator.force_left_turn = True
161             else:
162                 Navigator.right_turn_enabled = False
163                 self.num_kitchen_turns = 1
164 
165     def map_island_data(self):
166         if len(self.island_data) == 0:
167             print('ERROR: No island to map!')
168             return
169 
170         initial_island_data = list(self.island_data)
171         self.island_data = []
172         for pos in initial_island_data:
173             self.go_deeper(pos[0], pos[1], self.island_data, self.map_data)
174 
175     def map_remaining_data(self):
176         avoid_data = list(self.edges_data) + list(self.island_data)
177         initial_map_data = list(self.island_data)
178         self.map_data = []
179         for pos in initial_map_data:
180             self.go_deeper(pos[0], pos[1], self.map_data, avoid_data,
    skip_avoid_test=True)
181 
182         self.map_data += self.edges_data
183 
184     def go_deeper(self, x, y, lst, avoid_lst, skip_avoid_test=False):
185         pos = (x, y)
186         if pos not in avoid_lst or skip_avoid_test:
```

```
187             if pos not in lst:
188                 if not skip_avoid_test:
189                    lst.append(pos)
190 
191                 try:
192                     self.go_deeper(x + 1, y, lst, avoid_lst)
193                     self.go_deeper(x - 1, y, lst, avoid_lst)
194                     self.go_deeper(x, y + 1, lst, avoid_lst)
195                     self.go_deeper(x, y - 1, lst, avoid_lst)
196                 except:
197                     traceback.print_exc()
198                     print('Your island or map is too big')
199 
200     def save_current_section(self):
201         if self.mapping_state == MAPPING_STATE_FOLLOWING_OUTER_WALL:
202             self.process_finished_section()
203 
204             print('---- SECTION SAVED ----')
205             print('  direction: ' + str(self.current_section.direction))
206             print('  distance: ' + str(self.current_section.block_distance))
207             print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
208             print('  kitchen mappings: \n' + self.get_kitchen_debug_data())
209             print('-----------------------')
210         elif self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND:
211             self.process_finished_section()
212 
213             print('---- ISLAND ROUNDED ----')
214             print('  direction: ' + str(self.current_section.direction))
215             print('  distance: ' + str(self.current_section.block_distance))
216             print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
217             print('-----------------------')
218 
219             self.kitchen_num_mapped += 1
220             self.num_kitchen_turns += 1
221         else:
222             self.process_finished_section(store_data=False)
223 
224             print('---- SECTION TRAVELLED ----')
225             print('  direction: ' + str(self.current_section.direction))
226             print('  distance: ' + str(self.current_section.block_distance))
227             print('  coordinates: ' + str(self.current_x) + ", " + str(self.
    current_y))
228             print('-----------------------')
229 
230     def get_kitchen_debug_data(self):
231         data = []
232         for k, v in self.kitchen_mapping.items():
233             data.append('    ' + str(k) + ' -> ' + str(v))
234 
235         return '\n'.join(data)
236 
237     def begin_next_section(self, is_right_turn):
238         if is_right_turn:
239             self.current_section = self.current_section.for_right_turn()
240         else:
241             self.current_section = self.current_section.for_left_turn()
242 
243     def on_turning_started(self):
244         if self.mapping_state == MAPPING_STATE_RETURNING_TO_GARAGE:
245             if self.has_returned_to_start(include_direction=False):
246                 self.state = STATE_FINISHED
247                 self.navigator.set_mode(Navigator.MANUAL)
248                 print(self.map_data)
```

```
249                 print(self.island_data)
250         else:
251             self.state = STATE_WAITING
252 
253         self.save_current_section()
254 
255     def on_turning_finished(self, is_right_turn):
256         self.begin_next_section(is_right_turn)
257         if self.has_returned_to_start() or self.mapping_state ==
    MAPPING_STATE_RETURNING_TO_ISLAND:
258             self.mapping_state = MAPPING_STATE_RETURNING_TO_ISLAND
259             self.island_data = list(self.kitchen_mapping.keys())
260             self.edges_data = list(self.map_data)
261 
262         if self.mapping_state == MAPPING_STATE_FOLLOWING_ISLAND and self.
    num_kitchen_turns == 2:
263             self.kitchen_start_x = self.current_x
264             self.kitchen_start_y = self.current_y
265 
266         self.state = STATE_MEASURING
267 
268     def has_returned_to_start(self, include_direction=True):
269 
270         if self.current_x == 0 and self.current_y == 0 and len(self.map_data) >
    0:
271             return not include_direction or self.current_section.direction ==
    NORTH
272         else:
273             return False
274 
275     def transform_map_data(self, section, current_x, current_y, estimate=False,
    offset=None):
276         if estimate:
277             if offset is None:
278                 distance = section.estimate_block_distance()
279             else:
280                 distance = section.estimate_block_distance(offset)
281         else:
282             section.finish()
283             distance = section.block_distance
284 
285         return self.transform_partial_map_data(distance, section.direction,
    current_x, current_y)
286 
287     def transform_partial_map_data(self, distance, direction, current_x,
    current_y):
288         if direction == NORTH:
289             current_y += distance
290         elif direction == EAST:
291             current_x += distance
292         elif direction == SOUTH:
293             current_y -= distance
294         elif direction == WEST:
295             current_x -= distance
296         return current_x, current_y
297 
298     def store_ongoing_section(self):
299         self.current_section.finish()
300 
301         distance = self.current_section.block_distance
302         direction = self.current_section.direction
303 
304         for i in range(1, distance + 1):
305             pos_x, pos_y = self.transform_partial_map_data(i, direction, self.
    current_x, self.current_y)
```

```
306             self.map_data.append((pos_x, pos_y))
307
308     def process_finished_section(self, store_data=True):
309         self.current_section.finish(debug_limits=True)
310
311         if store_data:
312             distance = self.current_section.block_distance
313             direction = self.current_section.direction
314
315             for i in range(1, distance + 1):
316                 coordinates = self.transform_partial_map_data(i, direction, self
    .current_x, self.current_y)
317                 self.map_data.append(coordinates)
318                 self.kitchen_mapping.pop(coordinates, None)
319                 if coordinates not in self.invalid_kitchens:
320                     self.invalid_kitchens.append(coordinates)
321
322             self.saved_sections.append(self.current_section)
323
324         # Could be optimized to use last pos_x, pos_y instead
325         self.current_x, self.current_y = self.transform_map_data(self.
    current_section, self.current_x, self.current_y)
326
327     '''
328     Returns a list of tuples with coordinates of grids visited
329     '''
330     def get_map_data(self):
331         return [self.map_data]
332
333     '''
334     Returns the robots last know x,y coordinates
335     '''
336     def get_current_position(self):
337         return self.current_x, self.current_y
338
339     def get_left_block_coordinates(self, x, y, displacement):
340         if self.current_section.direction == NORTH:
341             x -= displacement
342         elif self.current_section.direction == EAST:
343             y += displacement
344         elif self.current_section.direction == SOUTH:
345             x += displacement
346         elif self.current_section.direction == WEST:
347             y -= displacement
348
349         return x, y
350
351     def get_right_block_coordinates(self, x, y, displacement):
352         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)
```

```
66      Distance recorded by the left IR sensor in mm, or -1 if the distance is not
67      within the supported range.
68  ir_right_mm (2 bytes, two's complement)
69      Distance recorded by the right IR sensor in mm, or -1 if the distance is
70      not within the supported range.
71  """
72
73  # Ping a unit
74  CMD_PING = 3
75  """
76  Dummy command which will only trigger the other unit to respond with a PONG
77  command. Preferably used to test a connection without forcing the other party
78  to perform any actual action.
79
80  Target: Any AVR unit
81
82  Arguments: None
83  """
84
85  # Pong a unit
86  CMD_PONG = 4
87  """
88  Reply to a PING command.
89
90  Target: Main unit
91
92  Arguments: None
93  """
94
95  # Set both motor speeds in the control unit
96  CMD_SET_MOTOR_SPEED = 5
97  """
98  Sets the speed of both the left and right motors on the robot. Values can
99  be both positive and negative and the range -100 to 100 is supported, where the
100 value represents a percentage of the max speed. It seems like the different
101 directions have various max speeds, which has to be considered when
102 implementing on-spot-rotation or other actions which assume equal speed forward
103 and backwards.
104
105 Target: Control unit
106
107 Arguments:
108 left_motor_speed (1 byte, positive or negative)
109     Left motor speed in percentage of max speed ranging from -100 to 100.
110 right_motor_speed (1 byte, positive or negative)
111     Right motor speed in percentage of max speed ranging from from -100 to 100.
112 """
113
114 # Set left motor speed only in control unit
115 CMD_SET_LEFT_MOTOR_SPEED = 6
116 """
117 Sets the speed of the left motors on the robot. It is only possible to pass
118 positive values with the command in the range 0 to 100, where the value
119 represent a percentage of the max speed.
120
121 Target: Control unit
122
123 Arguments:
124 speed (1 byte, positive)
125     Left motor speed in percentage of max speed ranging from 0 to 100.
126 """
127
128 # Set right motor speed in the control unit
129 CMD_SET_RIGHT_MOTOR_SPEED = 7
130 """
```

```
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

```

```
196 BT_REQUEST_SENSOR_DATA = 14
197 BT_RETURN_SENSOR_DATA = 15
198
199 BT_REQUEST_SERVO_DATA = 16
200 BT_RETURN_SERVO_DATA = 17
201
202 BT_REQUEST_MAP_DATA = 18
203 BT_RETURN_MAP_DATA = 19
204
205 BT_DRIVE_FORWARD = 20
206 BT_DRIVE_BACK = 21
207
208 BT_TURN_RIGHT = 22
209 BT_TURN_LEFT = 23
210
211 BT_DRIVE_FORWARD_RIGHT = 24
212 BT_DRIVE_FORWARD_LEFT = 25
213
214 AUTONOMOUS_MODE = 26
215 MANUAL_MODE = 27
216
217 # Toggle between autonomous and manual mode
218 CMD_TOGGLE_MODE = 28
219 """
220 Command to toggle between the available modes (autonomous and manual) instead
221 of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.
222 Target: Main unit
223 Arguments: None
224 """
225
226 # Indicates that the robot has changed to a new navigator mode
227 CMD_MODE_SET = 29
228 """
229 Command to toggle between the available modes (autonomous and manual) instead
230 of explicitly switching to one using AUTONOMOUS_MODE or MANUAL_MODE.
231
232 Target: Main unit
233
234 Arguments:
235 new_mode (1 byte)
236     Integer representation of the new mode:
237       0 = Manual mode
238       1 = Autonomous mode
239 """
240
241 BT_CLIENT_COMMANDS = [REQUEST_PI_IP, BT_SERVER_RESTART,
242                       BT_SERVER_SHUTDOWN, BT_REQUEST_SENSOR_DATA,
243                       BT_REQUEST_MAP_DATA, BT_REQUEST_SERVO_DATA,
244                       BT_DRIVE_FORWARD, BT_DRIVE_BACK,
245                       BT_TURN_RIGHT, BT_TURN_LEFT, BT_DRIVE_FORWARD_RIGHT,
    BT_DRIVE_FORWARD_LEFT]
246
247 BT_SERVER_COMMANDS = [REQUEST_PI_IP, BT_RETURN_SENSOR_DATA,
248                       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)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                  section.py                                #
4  #                                                                            #
5  #                                 Version 1.0                                #
6  #                           Senast modifierad 2016-12-15                     #
7  #                                                                            #
8  #                                Anton Dalgren                               #
9  #                                Patrik Sletmo                               #
10 #                                                                            #
11 ##############################################################################
12
13 from datetime import datetime
14 from math import floor
15
16
17 NORTH = 0
18 EAST = 1
19 SOUTH = 2
20 WEST = 3
21
22 BLOCK_LENGTH_MM = 400
23
24
25 class Section:
26     def __init__(self, direction):
27         self.direction = direction
28         self.measurements = []
29         self.block_distance = None
30
31     def add_distance_sample(self, distance):
32         if 50 < distance < 6000:
33             self.measurements.append((distance, datetime.now()))
34
35     def finish(self, debug_limits=False):
36         # If we're somehow detecting the next turn for a dead end corner before
37         # any measurements have been received our normal algorithm won't work.
38         # Handle this by explicitly setting the distance to zero.
39         if len(self.measurements) == 0:
40             self.block_distance = 0
41             return
42
43         # Takes the difference between max measurement and min measurement and
       divide by block length.
44         self.block_distance = round((self.get_max(debug_limits) - self.get_min(
       debug_limits)) / BLOCK_LENGTH_MM)
45
46     def estimate_block_distance(self, offset=0.25):
47         if len(self.measurements) == 0:
48             return 0
49
50         # Takes the difference between max measurement and min measurement and
       divide by block length.
51         return floor((self.get_max() - self.get_min()) / BLOCK_LENGTH_MM +
       offset)
52
53     def for_right_turn(self):
54         return Section((self.direction + 1) % 4)
55
56     def for_left_turn(self):
57         return Section((self.direction - 1) % 4)
58
59     def get_max(self, debug_limits=False):
60         if len(self.measurements) == 0:
61             return 0
```

```
62
63          value = max(self.measurements, key=lambda x: x[0])[0]
64          if debug_limits:
65              print('Max', value)
66
67          return value
68
69      def get_min(self, debug_limits=False):
70          if len(self.measurements) == 0:
71              return 0
72
73          value = min(self.measurements, key=lambda x: x[0])[0]
74          if debug_limits:
75              print('Min', value)
76
77          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
```

```
65          laser.read_data()
66          laser_distance = laser.get_data()
67          print("Laser distance: " + str(laser_distance))
68          request_sensor_data()
69
70
71  def main():
72      global busy, last_request
73
74      setup()
75
76      while True:
77          EventBus.receive()
78          request_data()
79
80  Safety.run_safely(main)
```

```
1  ##############################################################################
2  #                                                                            #
3  #                                  utils.py                                  #
4  #                                                                            #
5  #                                Version 1.0                                 #
6  #                          Senast modifierad 2016-11-30                      #
7  #                                                                            #
8  #                               Sebastian Callh                              #
9  #                               Rebecca Lindblom                             #
10 #                                Patrik Sletmo                               #
11 #                                                                            #
12 ##############################################################################
13
14 """
15 Various functions which does not belong in any other file.
16
17 NOTE: This file might have to be split if the list of functions grow too large.
18 """
19 import os
20
21
22 # Returns the two's complementary value in it's python representation
23 def twos_comp(val, bits):
24     if (val & (1 << (bits - 1))) != 0:
25         val -= 1 << bits
26
27     return val
28
29
30 def get_ip():
31     s = os.popen('ifconfig wlan0 | grep "inet\ addr" | cut -d: -f2 | cut -d" " -
       f1')
32     pi_ip = s.read()
33     return pi_ip
```