```c
/*
 * GccApplication1.c
 *
 * Created: 2019-04-09 11:27:55
 * Author : vi6708la-s
 */

#include "glob_header.h"
#include "timer0.h"
#include <stdio.h>
#include <stdlib.h>

#define  ADC_START()    ADCSRA |= (1 << ADSC);

#define BAUD 9600
#define BAUDRATE 51
#define MOTORAFORWARD 1
#define MOTORABACKWARDS 0
#define MOTORBFORWARD 3
#define MOTORBBACKWARDS 2
#define MOTORAENABLE 4
#define MOTORBENABLE 5

volatile int16_t calibration_values[5];

volatile int16_t sensor_values[5];

volatile uint8_t rx_header;
volatile uint8_t rx_high;
volatile uint8_t rx_low;

volatile uint8_t rx_data;
volatile uint8_t header;
volatile uint16_t left_data;
volatile uint16_t right_data;

volatile uint16_t pulse_a;


volatile uint8_t packets_to_recieve;

volatile uint8_t timer_cnt;
volatile uint8_t request_data;
volatile uint8_t receive_complete;

volatile uint8_t rx_data_1;
volatile uint8_t rx_data_2;
volatile uint8_t rx_data_3;
volatile uint8_t rx_data_4;
volatile uint8_t rx_data_5;
volatile uint8_t rx_data_6;

volatile uint8_t current_sensor;

volatile uint16_t current_adc;

```

```
57  volatile double current_offset = 0;
58
59  volatile uint8_t initalization;
60
61  volatile uint8_t calibrated = 0;
62
63  volatile uint8_t initial_calibration_read = 0;
64
65  volatile uint8_t mode = 1; // 1 för linjeföljare, 0 för radiostyrd
66
67  uint8_t right_h;
68  uint8_t right_l;
69  uint8_t left_h;
70  uint8_t left_l;
71  uint8_t trailer;
72  uint16_t right;
73  uint16_t left;
74
75  //Motor A (Right)
76  void initMotorA(){
77      DDRD |= (1<<MOTORAENABLE);
78
79      DDRB |= (1<<MOTORABACKWARDS);
80      DDRB |= (1<<MOTORAFORWARD);
81
82  }
83
84  void forwardMotorA(){
85      PORTB |=(1<<MOTORAFORWARD);
86      PORTB &=~(1<<MOTORABACKWARDS);
87  }
88
89  void reverseMotorA(){
90      PORTB |=(1<<MOTORABACKWARDS);
91      PORTB &=~(1<<MOTORAFORWARD);
92  }
93
94  void breakMotorA(){
95
96  }
97
98  //Motor B (Left)
99  void initMotorB(){
100     DDRD |= (1<<MOTORBENABLE);
101
102     DDRB |= (1<<MOTORBBACKWARDS);
103     DDRB |= (1<<MOTORBFORWARD);
104
105 }
106
107 void forwardMotorB(){
108     PORTB &=~(1<<MOTORBBACKWARDS);
109     PORTB |= (1<<MOTORBFORWARD);
110
111 }
112
```

```
113 void reverseMotorB(){
114     PORTB &=~(1<<MOTORBFORWARD);
115     PORTB |= (1<<MOTORBBACKWARDS);
116 
117 
118 }
119 
120 void breakMotorB(){
121     PORTB &=~(1<<MOTORBFORWARD);
122 }
123 
124 void spinClockwise(){
125     forwardMotorA();
126     reverseMotorB();
127 }
128 
129 void spinCounterClockwise(){
130     reverseMotorA();
131     forwardMotorB();
132 }
133 
134 void stop(){
135     breakMotorA();
136     breakMotorB();
137 }
138 
139 void PWM_init()
140 {
141     /*set fast PWM mode with non-inverted output*/
142     TCCR1A |= (1<<WGM11) | (1<<COM1A1) | (1<<COM1B1);
143     TCCR1B |= (1<<WGM13) | (1<<WGM12) | (1<<CS10);
144     //TIMSK |= (1<<OCIE1A);
145 
146 }
147 
148 void ADC_init(){
149     ADMUX |= (1 << REFS0);
150     ADCSRA |= (1 << ADEN) | (1 << ADIE) | (1 << ADPS0) | (1 << ADPS1) | (1 << ↵
        ADPS2);
151 }
152 
153 void timeout_timer_init(){
154         TCCR0 |=  (1<<CS02) | (1<<CS00);
155         TCNT0 = 0;
156 }
157 
158 void set_PWM_period(uint16_t period){
159     ICR1 = period;
160 }
161 
162 void set_PWM_pulse(uint16_t pulseA, uint16_t pulseB){
163 
164     right_data = pulseA;
165     left_data = pulseB;
166 
167     if (right_data > 1023) right_data = 511;
```

```
168     if (left_data > 1023) left_data = 511;
169     uint16_t reverse_flip_mask = 0x1ff;
170
171     if(left_data < 500){
172         reverseMotorA();
173         left_data ^= reverse_flip_mask;
174         left_data = (int) left_data/8;
175         left_data = left_data + 447;
176         OCR1A = left_data;
177         } else if(left_data >= 520) {
178         forwardMotorA();
179         left_data &= ~(1 << 9);
180         left_data = (int) left_data/8;
181         left_data = left_data + 447;
182         OCR1A = left_data;
183         } else if (left_data > 500 && left_data < 520) {
184         OCR1A = 0;
185     }
186
187     if(right_data >= 520){
188         forwardMotorB();
189         right_data &= ~(1 << 9);
190         right_data = (int) right_data / 8;
191         right_data = right_data + 447;
192         OCR1B = right_data;
193
194         } else if(right_data < 500){
195         reverseMotorB();
196         right_data ^= reverse_flip_mask;
197         right_data = (int) right_data / 8;
198         right_data = right_data + 447;
199         OCR1B = right_data;
200         } else if (right_data > 500 && right_data < 520){
201         OCR1B = 0;
202         }
203
204
205 }
206
207 void USART_Init(void){
208     //USAR T Baud Rate Register (UBRR)
209     // USART set to Asynchronous Normal Mode
210     UBRRH = (BAUDRATE>>8);// shift the register right by 8 bits
211     UBRRL = BAUDRATE;// set baud rate
212     UCSRB|= (1<<TXEN)|(1<<RXEN)|(1<<RXCIE);// enable receiver and transmitter
213     UCSRC|= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);
214 }
215
216 void USART_Transmit(volatile uint8_t tx){
217
218     while(!(UCSRA&(1<<UDRE)));
219     UDR=tx;
220
221 }
222
223
```

```
224 uint8_t USART_Receive() {
225
226     /* Wait for data to be received */
227     while ( !(UCSRA & (1<<RXC)));
228     /* Get and return received data from buffer */
229     return UDR;
230
231 }
232
233 void Calculate_Offset(){
234     cli();
235     uint8_t k = 1;
236     uint8_t max = sensor_values[k];
237     uint8_t current_compare;
238
239     for (uint8_t i = 1; i < 4; i++)
240     {
241
242         if (sensor_values[i] > max)
243         {
244             max = sensor_values[i];
245             k = i;
246         }
247     }
248
249     if(calibrated == 1){
250
251         if (k == 1){
252             current_offset = -1;
253         } else if (k == 2){
254             current_offset = 0;
255         } else if (k == 3){
256             current_offset = 1;
257         }
258     }
259
260     sei();
261 }
262
263 void Line_Following_PWM(){
264     if(current_offset > 0.2){
265             OCR1B = 480 + 30 * current_offset;
266             OCR1A = 470;
267     } else if (current_offset < -0.2){
268             OCR1A = 480 - 30 * current_offset;
269             OCR1B = 470;
270     } else {
271         OCR1A = 490;
272         OCR1B = 490;
273     }
274 }
275
276
277
278
279
```

```
280 int main(void) {
281     /* Replace with your application code */
282     initMotorA();
283     initMotorB();
284 
285 
286     set_PWM_period(511);
287     calibrated = 0;
288     current_sensor = 0;
289     mode = 1;
290 
291     PWM_init();
292     USART_Init();
293     ADC_init();
294     ADC_START();
295 
296     packets_to_recieve = 2;
297 
298     sei();
299     //request_packets(1);
300     right_data = 511;
301     left_data = 511;
302 
303     DDRC |= (1 << PC5) | (1 << PC4);
304     PORTC |= (1 << PC5) | (1 << PC4);
305 
306     while (1) {
307 
308 
309 
310 
311 
312             if ((rx_data_6 == 0xFE && rx_data_1 == 0xFF)) {
313 
314             right = (rx_data_3 << 8) | (rx_data_2);
315             left  = (rx_data_5 << 8) | (rx_data_4);
316             set_PWM_pulse(right, left);
317             }
318 
319         sei();
320     }
321 
322 
323 }
324 
325 
326 ISR(USART_RXC_vect){
327 
328     rx_data_6 = rx_data_5;
329     rx_data_5 = rx_data_4;
330     rx_data_4 = rx_data_3;
331     rx_data_3 = rx_data_2;
332     rx_data_2 = rx_data_1;
333     rx_data_1 = UDR;
334 
335     if ((rx_data_6 == 0xFE && rx_data_1 == 0xFF)) {
```

```
336
337         cli();
338     }
339
340 }
341
342 ISR(ADC_vect){
343
344     current_adc = ADC;
345     if(initalization == 1){
346     if(calibrated == 1){
347         sensor_values[current_sensor] = calibration_values[current_sensor] -
              current_adc;
348     } else if (calibrated == 0){
349         calibration_values[current_sensor] = current_adc;
350     }
351
352     if(current_sensor == 4){
353         current_sensor = 0;
354         calibrated = 1;
355     } else {
356         current_sensor++;
357     }
358
359     } else {
360         current_adc = ADC;
361         initalization = 1;
362
363     }
364     ADMUX = (ADMUX & 0xF8)|current_sensor;
365     ADC_START();
366 }
367
```

```
1  /*
2   * glob_header.h
3   *
4   * Created: 2019-05-13 14:40:04
5   *  Author: vi6708la-s
6   */
7
8
9  #ifndef GLOB_HEADER_H_
10 #define GLOB_HEADER_H_
11
12
13 #define F_CPU 8000000UL
14 #include <avr/io.h>
15 #include <avr/interrupt.h>
16 #include <util/delay.h>
17
18 #define TRUE    1
19 #define FALSE   0
20
21 #endif /* GLOB_HEADER_H_ */
```

```c
/*
 * timer0.c
 *
 * Created: 2019-05-13 14:38:35
 *  Author: vi6708la-s
 */

#include "timer0.h"


void timer0_ovf_init() {

    TCCR0 |= (1 << CS01) | (1 << CS00);
    TIMSK |= (1 << TOIE0);
    TCNT0  = 0;

}

void timer0_disable() {

    TCCR0 &= ~((1 << CS01) | (1 << CS00));

}

void timer0_enable() {

    TCCR0 |= (1 << CS01) | (1 << CS00);

}
```

```c
/*
 * timer0.h
 *
 * Created: 2019-05-13 14:38:21
 *  Author: vi6708la-s
 */


#ifndef TIMER0_H_
#define TIMER0_H_

#include "glob_header.h"
void timer0_ovf_init();
void timer0_disable();
void timer0_enable();

#endif /* TIMER0_H_ */
```