Projektrapport- Radiostyrd Robot

LUNDS UNIVERSITET

EITFF11 Digitala Projekt

Alexander Bengtsson, Noah Hansson, Victor Lagerfors

Handledare: Bertil Lindvall & Christoffer Cederberg

Innehållsförteckning

1. Inledning 3
2. Kravspecifikation 3
3. Hårdvara 3
3.1 Robot 3
Processor 3
Radiosändare 3
Två motorer med tillhörande bandhjul 3
H-brygga 3
3.2 Handkontroll 4
Processor 4
Radiosändare 4
Två styrspakar 4
4. Kopplingsschema 4
4.1 Kopplingsschema för roboten 4
4.2 Kopplingsschema för kontrollen 5
5. Mjukvara 5
6. Genomförande 5
7. Resultat 6
8. Diskussion 6

1. Inledning

I detta projekt har designades och byggdes en prototyp av en radiostyrd robot. Prototypen är en robot med bandhjul som styrs genom att kommunicera över radio med en handhållen kontroll.

Intresset för att bygga en radiostyrd kom då gruppens medlemmar uppskattat kursen i datorkommunikation, och även såg goda möjligheter att applicera kunskaper inom CAD för att konstruera roliga skal genom 3d-printning till roboten.

I denna rapport presenteras produkten och dess framtagning samt en redogörelse för dess hårdvara, mjukvara, koppling och programmering. Resultatet presenteras och slutligen avslutas rapporten med en diskussion och reflektion kring arbetet.

2. Kravspecifikation

Kraven på prototypen är som följande:

1. Robotens bandhjul ska kunna styras genom att kommunicera med en kontroll över radio
2. Bandhjulen på roboten ska kunna gå både framlänges och baklänges för att roboten ska vara fullt styrbar
3. Hastigheten på bandhjulen ska kunna varieras
4. Kontrollen ska ej vara fysiskt ihopkopplad med roboten

3. Hårdvara

3.1 Robot

Processor

Robotens processor är en AVR ATmega16. Processorn är en 8-bitars processor med 16 KB flashminne och 1 KB RAM. Processorn har 4 portar och 40 pinnar. Koden är skriven i C och är bränd in i processorn med en JTAG.

Radiosändare

Roboten kommunicerar med kontrollen genom ett DIGI XBee RC-chip. Ett likadant chip sitter på kontrollen och de kommunicerar över samma länk. Detta simulerades genom två sladdar mellan enheternas RX-TX pinnar.

Två motorer med tillhörande bandhjul

Motorerna som användes för att framdriva roboten var två generiska DC-motorer, en på vardera sida av konstruktionen. Dessa är i sin tur kopplade till varsitt band vilket tillåter att driva roboten framåt och för att styra den i sidled.

H-brygga

För att kontrollera motorerna används en H-brygga av typen L298N. H-Bryggan styrs med logiska signaler från mikroprocessorn och matar därefter motorerna med den relativt sett högre strömstyrka som krävs för att driva motorerna. Det är alltså H-bryggan som möjliggör styrning av motorerna. Med hjälp av h-bryggan möjliggörs även ändring av motorernas rotationsriktning, och detta möjliggör även att justera hastigheten på motorerna genom att med hjälp av PWM skicka in pulser för att aktivera och deaktivera motorerna med ytterst korta intervall.

3.2 Handkontroll

Processor

Handkontrollen har likt roboten en AVR ATmega16-processor.

Radiosändare

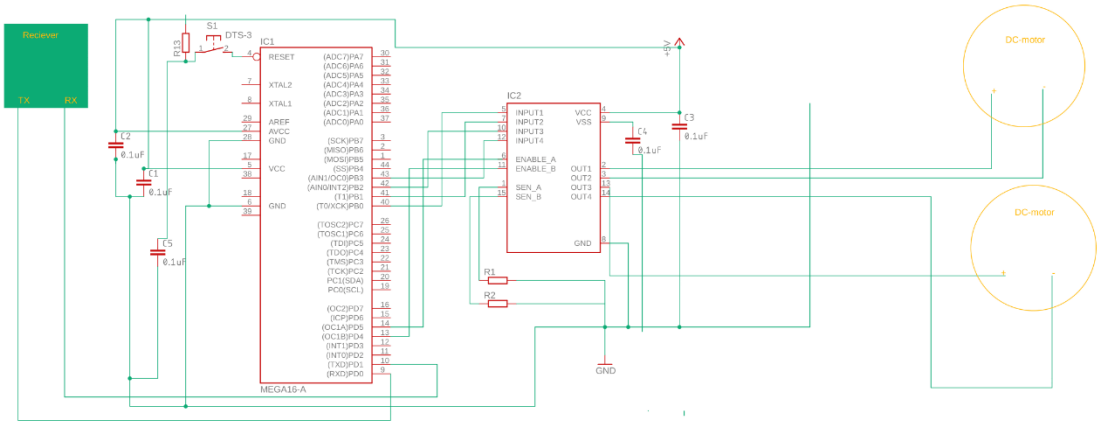
Likt roboten har även handkontrollen ett DIGI XBee RC-chip för att kommunicera trådlöst. Den skickar seriella data.

Två styrspakar

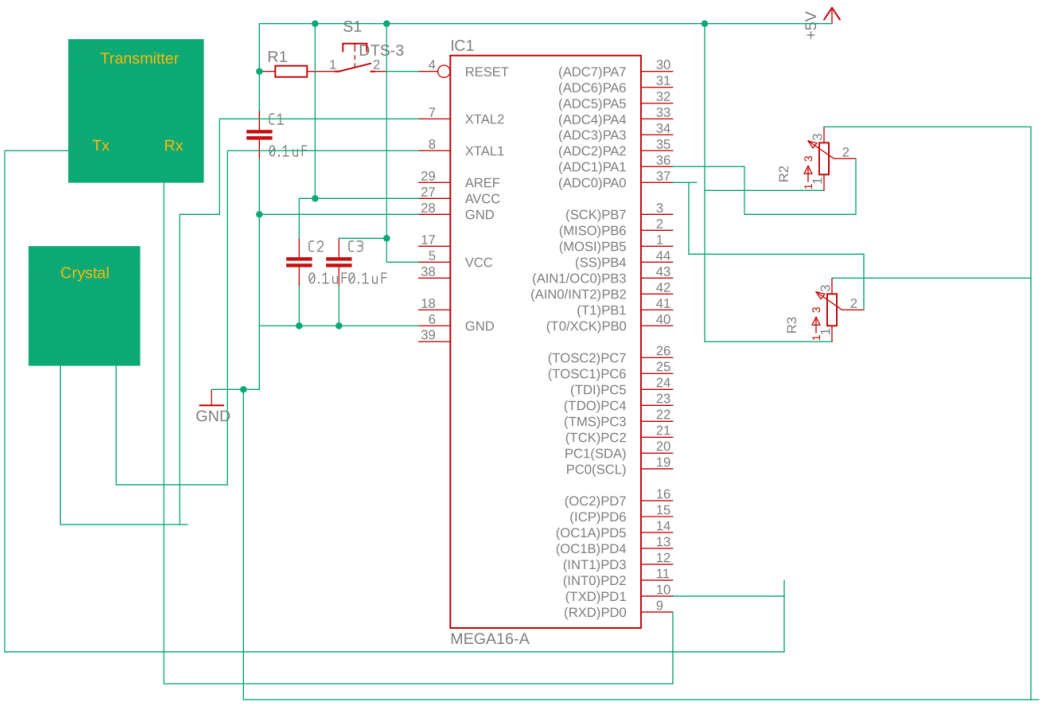
Handkontrollen har två styrspakar som är läser av spakens läge med en potentiometer. Varje styrspak styr respektive motor på roboten.

4. Kopplingsschema

4.1 Kopplingsschema för roboten



4.2 Kopplingsschema för kontrollen



5. Mjukvara

När roboten startas initieras roboten och ställer in alla portar. Motorerna styrs genom PWM där PWM-intensiteten styr motorernas hastighet. Kommunikation sker med en hastighet på 9600 baud på AVR:ens RX- och TX-portar för seriella data. Kontrollens main-loop skickar kontinuerligt byte för att kommunicera med bilen. Datan består av en header-byte, två byte för vänster hjul, två byte för höger hjul och en trailer. När bilen tar emot ett paket genereras ett interrupt där den kontinuerligt sparar alla paket, och adderar paketet längst bak i en fifo-kö. Matchar strukturen på inkomna data med strukturen för en komplett försändelse ställer bilen in vad PWM-paramterarna ska vara, annars så kastar den datan som är äldst i kön och fortsätter att kasta var gång ett nytt paket kommer in tills komplett data finns. Om signalen är låg, det vill säga att styrspaken dras bakåt, inverteras datan i paketet. Detta för att signalen från spaken i sitt utgångsläge på 512 ska ge en PWM-puls på 0, och att maxlägena på spaken, 1023 och 0, ska vara maxvärdet på PWM-pulsen 512. Om spaken är i ett läge mindre än 512 inverteras motorerna och PWM-signalen ökar proportionerligt mot spakens minskning.

Mjukvaran är programmerad i språket C i Atmel Studio 7.0. För att flasha programmet till Atmega16 användes JTAG-utrustning för att kommunicera på portarna avsedda för JTAG-kommunikation.

6. Genomförande

Genomförandet utfördes i ett antal etapper.

I det första steget planerades arbetet. Projektidén togs fram och diskuterades med handledare. Efter ett antal mindre modifieringar av projektidén godkändes denna. Därefter togs ett blockdiagram av konstruktionen fram, utifrån denna gjordes ett fullständigt kopplingsschema som godkändes av handledaren. För att designa kopplingsschemat användes mjukvaran Eagle.

Efter godkänt kopplingsschema började roboten och kontrollen konstrueras enligt ritning. En kopplingsplatta monterades på ett 3d-printat chassi med integrerad hjulupphängning och motorinfästning. Elektronikutrustningen i form av motstånd, kondensatorer, H-brygga m.m. monterades och sammankopplades på kopplingsplattan. Sammankopplingen gjordes genom trådbindning och lödning då det inte gick att vira. Samma process och metoder användes vid monteringen av styrkontrollen. Därefter påbörjades mjukvaruutvecklingen. Ett kommunikationsprotokoll skrevs, och regleringen av motorhastigheten gjordes med hjälp av pulsviddsmodulering. Då gruppen inte hade tillgång till RC-chip förrän mot slutet av projektet fick detta ersättas av två trådar mellan enheternas RX-TX pinnar.

7. Resultat

Till sist hade en fungerande prototyp av en radiostyrd robot byggts och programmerats, denna uppfyllde kravspecifikationen. Roboten kan styra båda de individuella bandhjulen fram och tillbaka oberoende av varandra för att köra framåt och bakåt, samt att svänga genom asymmetrisk hastighet på bandhjulen. Både roboten och kontrollen är helt trådlösa och har strömförsörjning genom batterier med ström på över 6v+ som stegas ner till 5v på grund av känsliga komponenter.

8. Diskussion

Efter att ha genomfört detta projekt från idé till färdig prototyp kan det med säkerhet sägas att det varit lärorikt, frustrerande, utmanande och väldigt roligt. Då gruppen innan projektet hade väldigt begränsade kunskaper inom integrerad utveckling behövde gruppen leta reda på information om C-programmering, hårdvara och mycket mer. På vägen möttes många svårigheter. Att få kommunikationsprotokollet att fungera var speciellt svårt då detta behövde skrivas från grunden. Att skicka åtta bitar över seriell kommunikation är inga konstigheter, då man kan skicka ett sådant paket direkt. AD-omvandlade värden från styrspakarna användes, vilket skapar problem att skicka dessa som seriell data då dessa värden är på tio bitar. Detta innebar att varje AD-omvandling behövde delas upp i två paket, vilket då skapade behov av felhantering. Då protokollet i början fungerade genom att roboten frågade kontrollen om data kunde överföringen ibland stoppas då bitfel uppstod, eftersom roboten då inte fick tillbaka komplett data och därmed inte bad om ett nytt paket. Andra konstiga fel som detta ledde till vara att höger och vänster spak råkade byta vilken motor det styrde, förmodligen då också på grund av bitfel vilket ledde till förskjutning av vilka paket som lades in i PWM-kontroll-metoden. Ett mer gediget kommunikationsprotokoll hade varit av nytta för detta projekt, men är svårt att skapa.

Ett annat problem som uppstod var när DC-motorerna genererade signaler i form av brus som fick robotens motorer att låsa sig och sluta lyssna på kontrollen. Detta löstes genom flitig användning av kondensatorer. Felsökningsprocessen för att komma fram till denna slutsats har dock varit lång och omständig, då trots att samma kod använts på en konstruktion som har varit korrekt kopplat har resultaten under utvecklingsprocessen varierat. Mycket diskussion har förts med handledare Christoffer kring problemet och trots detta har felsökningen tagit enorma tidsresurser. Såhär i efterhand hade en mer systematisk approach till felsökningen gynnat vårt projekt. Trots att tester aldrig utfördes på H-bryggan och dess funktionsduglighet så är det sannerligen där problemet har legat. Gruppen hade kunnat konstatera detta enklare genom att kolla vad PWM-signalerna från ATmega16 hade varit med hjälp av ett oscilloskop, och därmed lagt resurser på att felsöka de delar som är trasiga eller behov av byte. För att undvika liknande problem ifall detta projekt skulle göras igen hade det varit av värde att funktionstesta påmonterade komponenter bättre, helst innan de monteras då möjligt.

Under nästan hela arbetets gång fanns visionen att kunna få roboten att följa linjer i golvet med hjälp av en ljussensor monterad på robotens undersida. Detta visade sig vara mycket svårare än vad som tidigare antagits och på grund av tidsbrist efter att ha fått radiostyrningen att fungera valdes denna funktion bort. En lärdom som dragits av projektet är att genomförande av projekt ofta tar betydligt längre tid än vad som förväntas.

```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_ */
```

```c
/*
 * adc_debug.c
 *
 * Created: 2019-05-08 14:40:42
 * Author : vi6708la-s
 */

#include "glob_header.h"
#include "adc.h"
#include "uart.h"
#include "timer0.h"

#define TRUE    1
#define FALSE   0

#define RS 5
#define RW 4
#define lcd_enable 3

volatile uint16_t adc_res;
volatile uint8_t adc_low;
volatile uint8_t adc_high;
volatile uint8_t rx_data;

volatile uint8_t left_l;
volatile uint8_t left_h;
volatile uint8_t right_l;
volatile uint8_t right_h;

volatile uint8_t left_adc_checksum;
volatile uint8_t right_adc_checksum;

volatile char current_adc_read;
volatile uint8_t transmit_data;

volatile uint8_t timer_cnt;
volatile uint8_t request_data;

void timeout_timer_init(){
    TCCR0 |= (1<<CS02) | (1<<CS00);
    TCNT0 = 0;
}


void transmit_stick(void){

    if((right_l == 255) || (right_l == 254)) right_l = 253;
    if((left_l == 255)  || (left_l == 254)) left_l = 253;

    uart_transmit_byte(0xFE);
    uart_transmit_byte(right_h);
    uart_transmit_byte(right_l);
    uart_transmit_byte(left_h);
    uart_transmit_byte(left_l);
    uart_transmit_byte(0xFF);

```

```
57
58  }
59
60  void init_display(){
61      DDRD |= (1 << )
62  }
63
64  void setRSHigh()
65  {
66      PORTD = PORTD | 0b10000000;
67  }
68
69
70  int main(void)
71  {
72      DDRD |= (1 << PD4);
73      adc_init();
74      sei();
75      uart_init();
76      timer0_ovf_init();
77      ADC_START();
78
79      current_adc_read = 0;
80
81
82      while (1)
83      {
84          if (transmit_data == TRUE) {
85
86              transmit_data = FALSE;
87              transmit_stick();
88              timer0_enable();
89              adc_enable();
90              sei();
91
92          }
93
94
95      }
96  }
97
98
99
100 ISR(ADC_vect) {
101
102     if(!(ADMUX & (1 << MUX0))){
103         left_l= ADCL;
104         left_h = ADCH;
105         ADMUX &= ~(1 << MUX0);
106         ADMUX |=  (1 << MUX0);
107     } else if ((ADMUX & (1 << MUX0))){
108         right_l = ADCL;
109         right_h = ADCH;
110         ADMUX &= ~(1 << MUX0);
111     }
112
```

```
113
114
115
116     ADC_START();
117
118 }
119
120 ISR(USART_RXC_vect){
121
122     rx_data = UDR;
123
124 }
125
126 ISR(TIMER0_OVF_vect) {
127
128
129     timer_cnt++;
130     if(timer_cnt == 200) {
131
132         transmit_data = TRUE;
133         timer0_disable();
134         adc_disable();
135         timer_cnt = 0;
136         cli();
137
138     }
139
140
141 }
```

```c
/*
 * glob_header.h
 *
 * Created: 2019-05-13 14:40:04
 *  Author: vi6708la-s
 */


#ifndef GLOB_HEADER_H_
#define GLOB_HEADER_H_


#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define TRUE    1
#define FALSE   0

#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);

}
```

```
1  /*
2   * timer0.h
3   *
4   * Created: 2019-05-13 14:38:21
5   *  Author: vi6708la-s
6   */
7
8
9  #ifndef TIMER0_H_
10 #define TIMER0_H_
11
12 #include "glob_header.h"
13 void timer0_ovf_init();
14 void timer0_disable();
15 void timer0_enable();
16
17 #endif /* TIMER0_H_ */
```

```c
/*
 * adc.c
 *
 * Created: 2019-05-08 14:40:53
 *  Author: vi6708la-s
 */

#include "adc.h"

void adc_init() {

    ADMUX |= (1 << REFS0);
    ADCSRA |= (1 << ADEN) | (1 << ADIE) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2);


}

void adc_disable() {

    ADCSRA &= ~(1 << ADEN);

}

void adc_enable() {

    ADCSRA |= (1 << ADEN);
    ADC_START();

}
```

```c
/*
 * adc.h
 *
 * Created: 2019-05-08 14:41:04
 *  Author: vi6708la-s
 */


#ifndef ADC_H_
#define ADC_H_

#include <avr/io.h>

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

void adc_init();
void adc_disable();

void adc_enable();

#endif /* ADC_H_ */
```

```c
/*
 * uart.c
 *
 * Created: 2019-05-07 14:30:37
 *  Author: vi6708la-s
 */

#include "uart.h"


void uart_init(){

    UCSRB |= (1<<RXCIE) | (1<<RXEN) | (1<<TXEN);
    UBRRL =  51;

    //UCSRC |= (1<< USBS);

}

void uart_transmit_byte(volatile uint8_t tx_data) {

        /* Wait for empty transmit buffer */
        while ( !( UCSRA & (1<<UDRE)));
        /* Put data into buffer, sends the data */
        UDR = tx_data;

}
```

```c
/*
 * uart.h
 *
 * Created: 2019-05-07 14:30:27
 *  Author: vi6708la-s
 */


#ifndef UART_H_
#define UART_H_

#include <avr/io.h>

void uart_init();
void uart_transmit_byte(uint8_t tx_data);

#endif /* UART_H_ */
```