

Radiostyrd bil kontrollerad med handrörelser

Självständigt projektarbete inom programmering av enkapseldatorer

Sigge Axelsson, Jacob Signeul

Civilingenjörsprogrammet i elektroteknik

Programmering av enkapseldatorer 10 hp

5 mars 2024



UPPSALA UNIVERSITET

Uppsala university
i samarbete med Uwe Zimmermann

Innehåll

1	Introduktion	1
2	Metod och teori	1
2.1	Hårdvara	1
2.1.1	Enkapseldator ATmega328p	1
2.1.2	USB-TTL pl2303hx	1
2.1.3	Bluetooth HC-05	2
2.1.4	Accelerometer MPU6050	3
2.1.5	Motordrivare SN754410	4
2.2	Mjukvara	4
2.2.1	Medelanden och kodning	4
2.2.2	Motorstyrning	5
2.2.3	Protokoll för tappad anslutning	5
2.2.4	Watchdog timer	6
3	Resultat	6
3.1	Kontroller	6
3.2	Bilen	7
4	Diskussion	8
4.1	MPU6050 i praktiken	8
4.2	I2C timeout	8
4.3	Medelanden och kodning	8
5	Slutsatser	9
A	Appendix kretsscheman	10
A.1	Kretsschema kontroller	10
A.2	Kretsschema bilen	11
B	Appendix Kod	12
B.1	Bil	12
B.2	Kontroller	14

1 Introduktion

Trådlös teknologi är en central del av vår vardag. Genom att använda enkapseldatorer för att trådlöst styra och övervaka olika system har vi skapat en värld full av möjligheter. För att utforska detta närmare ska vi bygga en radiostyrd bil. Den kommer att styras trådlöst från en accelerometer som känner av användarens handrörelser. Informationen från accelerometern skickas sedan trådlöst via bluetooth till bilen, där enkapseldatorer använder den för att styra bilens motorer. Målet med att bygga och utforska denna bil är att vi ska få en djupare förståelse för trådlös kommunikation och styrning.

2 Metod och teori

2.1 Hårdvara

2.1.1 Enkapseldator ATmega328p

För att styra kontrollen och bilen behövs en mikrokontroller på vardera ände, en på kontrollern och en på bilen. För detta projekt användes två ATmega328p. ATmega:n programmerades i *C* genom *VisualStudioCode* med verktyget *PlatformIO*. Det går att programmera ATmega:n med *assembly*, men för detta projekt valde vi att använda *C*. ATmega328p har 28 portar, varav 23 kan användas som input/output (I/O)-portar. För de övriga 5 används 2 för att koppla 0 volts referensspänning (*GND*), 1 för matningsspänning (*V_{CC}*), 1 för analog till digital omvandlarens matningsspänning (*AV_{CC}*), och den sista som referensspänning (*AREF*) för analog till digital omvandlaren. Av de 23 portarna används 2 för UART-kommunikation med bluetooth-chippen, 2 för I²C-kommunikation för accelerometern, 4 för pulse width modulation (PWM), och 8 som I/O-portar.



Figur 1: ATmega328p

2.1.2 USB-TTL pl2303hx

För att på ett effektivt sätt kunna felsöka koden hos en enkapseldator samt för att konfigurera bluetooth-modulen HC-05 så användes en USB-seriell adapter. Detta möjliggör seriell kommunikation mellan en dator och enheter som stödjer UART-kommunikation och kan användas via t.ex. Serial monitor i Arduino IDE. USB-seriell

adapter som användes i detta projekt var av modellen pl2303hx. Det ska noteras att drivrutinen som i Windows automatiskt installeras för pl2303hx inte fungerar, indikerat av att drivrutinen har namnet “pl2303hx phased out since 2012”. För att undkomma detta problem så är det möjligt att installera äldre drivrutiner, i detta projekt användes Prolifics drivrutin för pl2303-moduler med versionen 3.3.3.114 för att få modulen att fungera som tänkt i Windows.



Figur 2: USB-TTL-modul av modellen pl2303hx

2.1.3 Bluetooth HC-05

Kommunikation via Bluetooth utfördes med modulen HC-05, vilken är en modul med möjligheten att både fungera i rollen som ”slave” och ”master”. HC-05-modulen som används i projektet består av en EGBT-045MS Bluetooth-modul integrerad i ett utbrytningskort av modellen zs-040 [1] [2] [3]. EGBT-045MS-modulen sköter Bluetooth-kommunikationen och drivs med en matningsspänning på 3,3V samt använder sig av 3,3V-logik för seriell kommunikation. zs-040-kortet syftet att förenkla användningen av EGBT-045MS-modulen och innehåller en 3,3V-regulator, diod för polaritetsskydd, statusport som indikerar anslutning, UART-portar (RXD och TXD) samt en knapp som gör modulen mottaglig för AT-kommandon.

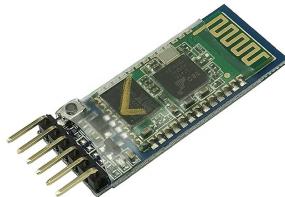
Tabell 1: Relevanta AT-kommandon för konfigurering av slave och master

Slave	Master
AT+UART=9600,0,0	AT+UART=9600,0,0
AT+RMAAD	AT+RMAAD
AT+ROLE=0	AT+ROLE=1
AT+ADDR?	AT+CMODE=0
	AT+BIND=(slave ADDR)

Tabell 2: [1] Förlaring av AT-kommandon i tabell 1

AT-kommandon	Förlaring
AT+UART=*arg,**arg,***arg	*Baud, **stopbit, ***paritetsbit
AT+RMAAD	Rensar tidigare anslutna enheter
AT+ROLE=*arg	*0-slave, *1-master
AT+ADDR?	Svarar med bluetoothmodules adress
AT+BIND=*arg	*fast adress att ansluta till, (xxxx,xx,xxxxxx)
AT+CMODE=*arg	*0-anslut till fast adress, *1-anslut valfri adress

[1] EGBT-045MS använder AT-kommandon för extern konfiguration. För att konfigurera två HC-05-moduler att ansluta till varandra användes kommandon som beskrivs i tabell 1, där en HC-05-modul konfigureras som ”slave” och en som ”master”. Efter att HC-05-modulerna har konfigurerats ansluter de automatiskt till varandra vid uppstart. Förlaringar av AT-kommandon i tabell 1 finns i tabell 2.



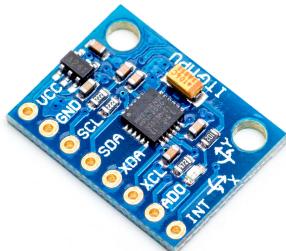
Figur 3: Bluetooth-modul av modellen HC-05 med utbrytningsbräden zs-040

2.1.4 Accelerometer MPU6050

Modulen som används för att mäta vinkeln på handen är en MPU-6050, en accelerometer och gyroskop i ett. I detta projekt används endast accelerometern när vinkeln på handen mäts. MPU-6050 mäter accelerationen genom en massa i sensorn som accelereras och skapar en förändring i kapacitans som mäts. Kapacitansen resulterar i en spänningsförändring som förstärks och filtreras, och sedan omvandlas med en 16-bits ADC. För kommunikation mellan MPU-6050 och ATmega328p används I²C-protokollet. Eftersom ATmega bara har 8-bitars minne läses de högre 8 bitarna först, följt av de lägre 8 bitarna. Detta görs för alla tre axlarna i accelerometern.

Även om MPU-6050 har inbyggd filtrering används ett löpande medelvärde av de åtta senaste mätningarna i koden för stabilare mätvärden. Eftersom sensorn mäter accelerationen ger den inte direkt en vinkel för sensorns lutning, men det kan antas att när en av sensoraxlarna ligger parallellt med tyngdaccelerationen kommer den att mäta 1g acceleration på z-axeln och 0g på x- och y-axlarna. När accelerometern

vinkel 90° mäter x-axeln 1g och z- och y-axeln mäter 0g. Detta används för att mäta hur mycket x- och y-axlarna är lutade och därmed hur snabbt och i vilken riktning man vill styra fordonet. [4] [5]



Figur 4: Accelerometer-modul av modellen MPU6050

2.1.5 Motordrivare SN754410

För att driva motorerna används två SN754410 kvadrupel halv H-bryggor. Dessa konfigureras så att två halva H-bryggor bildar en hel H-brygga. Det ger möjligheten att driva varje motor med en egen H-brygga. För att styra motordrivarna används tre signaler per motor. Två signaler är kopplade till varsin sida av motorn i H-bryggan och bestämmer riktningen som motorn snurrar. Detta uppnås genom att sätta ena sidan till hög och den andra till låg för att driva motorn åt ena hålet, och vice versa för att driva motorn åt andra hålet. Den tredje signalen är kopplad till en av Pulse Width Modulation (PWM) portarna på ATmega och styr hastigheten på motorn. Detta görs genom att ändra pulsbreddsmodulerings duty cycle, vilket ändrar den genomsnittliga tiden som drivkretsen är aktiverad, vilket resulterar i en ändring av medelvärdet av spänningen över motorn. [6]



Figur 5: H-brygga av modellen SN754410

2.2 Mjukvara

2.2.1 Medelanden och kodning

Kontrollern som läser av accelerometerns lutning är programmerad att skicka ut tre typer av meddelanden: *left*, *right* och *direction*. Meddelandena har en längd på 8

bitar, varav de två mest signifikanta bitarna (MSB) används för att koda meddelandets typ och de återstående 6 bitarna fylls med data från accelerometern. Kodningen utförs genom att först fylla ett meddelandes 8 bitar med data från accelerometern, sedan bitshifta meddelandet två steg till höger och slutligen läggs kodbitarna till vid position 0 och 1 (räknat från vänster).

Bilens mikrokontroller tar emot datapaketet från kontrollen och baserat på de två mest signifikanta bitarna avgörs vilken typ av meddelande som har skickats, enligt tabell 3. För meddelandena *right* och *left* avkodas meddelandena genom att bitshifta 2 steg till vänster. För *direction* sparas de tre minst signifikanta bitarna och resterande bitar kastas.

bit 0-1 (MSB)	meddelandetyp
01	right
10	left
11	direction

Tabell 3: Tilldelad variabel baserat på 2 MSB i mottaget meddelande

2.2.2 Motorstyrning

Riktningen på bilen styrs med *direction*, vilket anger vilka portar som är höga och vilka som är låga för motorernas drivarkretsar, se rad 84 - 99 i appendix B.1. *left* och *right* har ett värde mellan 0 och 255 och styr OCR-värden till PWM-portarna. Ett högre värde innebär att duty cycle på PWM-porten är högre. Detta uppnås genom att använda de interna 8-bits klockorna *TIMER0* och *TIMER2*. Båda timerarna är konfigurerade att ha respektive port hög tills att klockan uppnår OCR-värdet, då sätts porten till låg. Varje timer har två OCR-portar och varje port styr en motor.

Initialt användes en hög frekvens på PWM-signaler, men detta ledde till att motorerna inte kunde drivas korrekt då induktionen från motorerna påverkades av de snabba ändringarna i strömmen. Detta lösades genom att använda klockorna med en *prescaler* så att de räknade 1024 gånger långsammare än vanligt. Den ursprungliga frekvensen för PWM-signalen var 31250 Hz, men med prescalern blev detta 30.5 Hz.

2.2.3 Protokoll för tappad anslutning

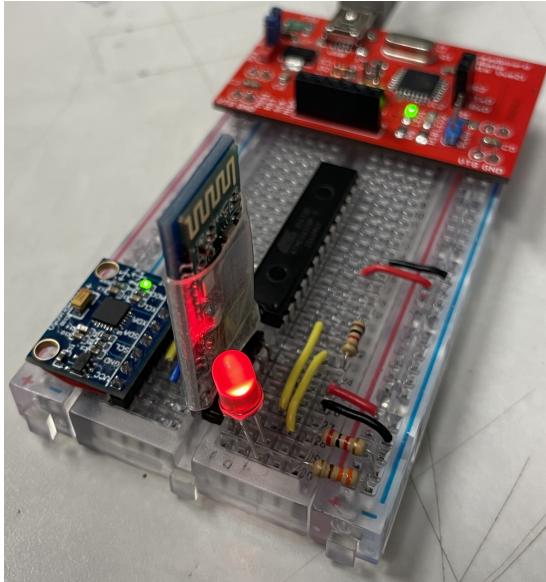
Bilen är programmerad så att signalerna till motorerna förblir desamma tills dess att bilens mikrokontroller tar emot ett meddelande som instruerar en förändring. Detta kan leda till ett scenario där anslutningen till kontrollern tappas och bilen fortsätter att köra. För att förhindra att detta inträffar stannar bilens motorer om inget nytt meddelande har mottagits på 500 ms, se rad 57 - 67 i appendix B.1.

2.2.4 Watchdog timer

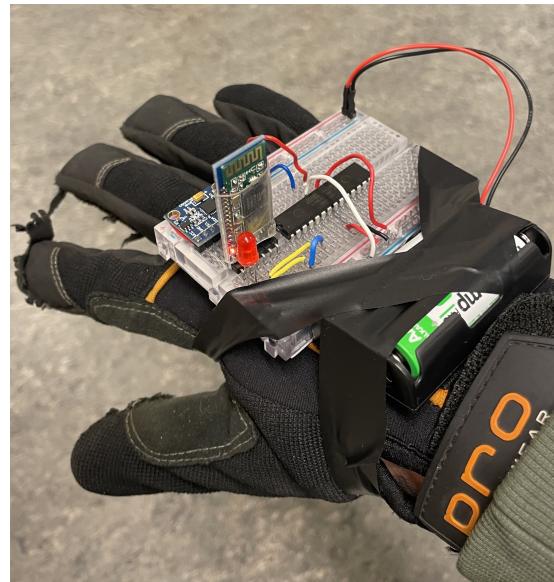
För att förhindra att kontrollen tappar kontakt med bilen under längre perioder på grund av att mikrokontrollern stöter på stopp i koden har en watchdog-timer implementerats i kontrollerns kod. Watchdog-timern fungerar genom att den tickar upp en klocka och när den når ett satt värde startar den om mikrokontrollen. För att förhindra att watchdog-timern startar om kontrollen återställer man räknaren i huvudloopen på programmet. Implementationen av watchdog timern kan ses i rad 40 - 47 samt rad 58 i appendix [B.2](#)

3 Resultat

3.1 Kontroller



(a) Kontrollerkretsen monterad på en prototypbräda med en röd LED som indikerar anslutning till bilen

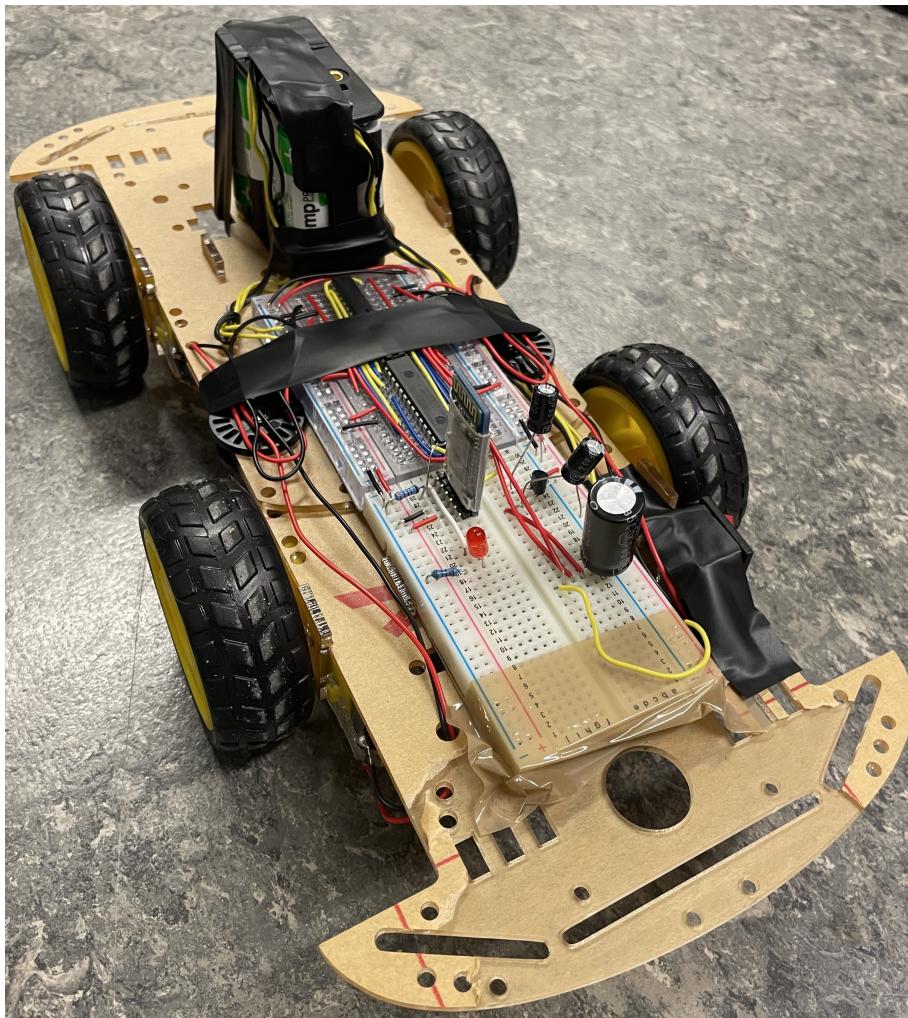


(b) Prototypbräda monterat på en handske

Figur 6: Prototyp av kontroller

I figur [6a](#) visas en bild av den färdiga prototypen av kontrollern. Accelerometern är ansluten till mikrokontrollern via I^2C och skickar data som beskriver kontrollerns lutning. Mikrokontrollern bearbetar och kodar sedan datan innan den skickar den via UART till bluetooth-modulen. Eftersom bluetooth modulen använder 3,3V logik så måste en spänningsdelare användas för att sänka mikrokontrollens UART spänning som använder 5V logik. Mikrokontrollen, accelerometer och bluetooth-modulen är kopplade enligt kretsschemat i appendix [A.1](#), figur [8](#).

3.2 Bilen



Figur 7: Bilen med kretsen kopplad på prototypbrädor driven av 8 stycken AA-batterier, monterat på ett plast-chassi

I figur 7 visas den färdiga prototypen av bilens drivlina. Bilen drivs med 12 volt från 8 stycken AA-batterier kopplade direkt till H-bryggorna och en spänningsregulator. Spänningsregulatorn förser en spänning på 5V till mikrokontrollern, bluetooth-modulen och H-bryggorna. Bluetooth-modulen ansluter till kontrollern och väntar på att ta emot meddelanden. Mottagna meddelanden behandlas sedan av mikrokontrollern som skickar ut 5V PWM-signaler till H-bryggorna. H-bryggorna matar sedan ut en 12V PWM-signal till motorerna. För att motverka störningar från H-bryggorna till mikrokontrollern är en 1mF avkopplingskondensator ansluten mellan H-bryggornas 12V matningsspänning och jord. Mikrokontrollen, de två motordrivarna och en bluetooth-modul kopplades ihop enligt kretsschemat i appendix A.2, figur 9.

4 Diskussion

4.1 MPU6050 i praktiken

Slutresultatet blev en fungerande bil som styrs genom att rotera accelerometern. När kontrollen roteras så att x-axeln på accelerometern mäter $1g$ av acceleration skickas ett värde på 252 till bilen. Att det inte är 255 beror på att när meddelandet bitshiftas efter att det mottagits, förlorar de två lägsta bitarnas värden och båda blir satta som 0. Skillnaden mellan 255 och 252 är inte märkbar och åtgärdades inte.

MPU6050 läser av accelerationen och skickar det som ett 16-bits värde. Detta skalas ned med 64, vilket ger ett värde mellan 0 och 1024. Detta värde är 10-bitars och kommer vara för stort för att användas. Accelerometern är inställd på att maximalt mäta $\pm 2g$, men vi använder bara $\pm 1g$ som maxacceleration. Därför är värdena mellan 0 och 512. I praktiken gav detta oss ett värde mellan 0 och 256, vilket var praktiskt och är en anledning till varför det är bra att undersöka vilka signaler man faktiskt tar emot och skickar.

4.2 I2C timeout

Ett problem som stötte på under projektet var med I2C-kommunikationen mellan MPU6050 och ATmega328p. Ibland kunde klockpulsens (*SCL*) fastna på hög och koden slutades köra. Interna timers var fortfarande igång, vilket tydde på att koden hade fastnat i en loop någonstans. En anledning kan vara att om I2C-kommunikationen missar en bit i dataströmmen kan mikrokontrollen tro att den inte fått ett svar medan MPU6050 har skickat det och väntar på att få skicka ny data. Detta kan lösas genom att man väntar på svar från MPU:n endast en viss tid innan man avslutar sändningen och påbörjar en ny. Eftersom vi använde färdiga bibliotek för I2C-protokollen användes watchdog-timern som starta om ATmegans istället.

4.3 Meddelanden och kodning

För att avgöra vilken typ av meddelande som mottagits från bluetooth-modulen använde vi en kodning beskriven i avsnitt 2.2.1. Nackdelen med detta val av kodning är att upplösningen från våra sensorer minskar från 8 bitar, vilket motsvarar 256 olika positioner, till 6 bitar, vilket motsvarar 64 positioner. 64 positioner bedömdes som tillräckligt bra upplösning för att styra bilen. Vid risk för dålig mottagning eller kraftigt brus skulle man dock kunna tänka sig att ett bitfel i den kodade delen av ett meddelandet skulle kunna få stora konsekvenser för styrningen av bilen, då meddelanden skulle blandas ihop.

En alternativ metod som experimenterades med var att skicka okodade meddelanden i en känd sekvens, t.ex. *left*, *right*, *direction*, följt av ett slut-/stop-meddelande för att hålla koll på var mikrokontrollen befinner sig i meddelandesekvensen. Denna

metod skulle ge oss en högre upplösning på 8 bitar, men med risken att systemet hamnar i osynk om ett meddelande missas. Detta skulle få följdten att felaktiga meddelanden sedan används av mikrokontrollern när den ska styra motorerna.

5 Slutsatser

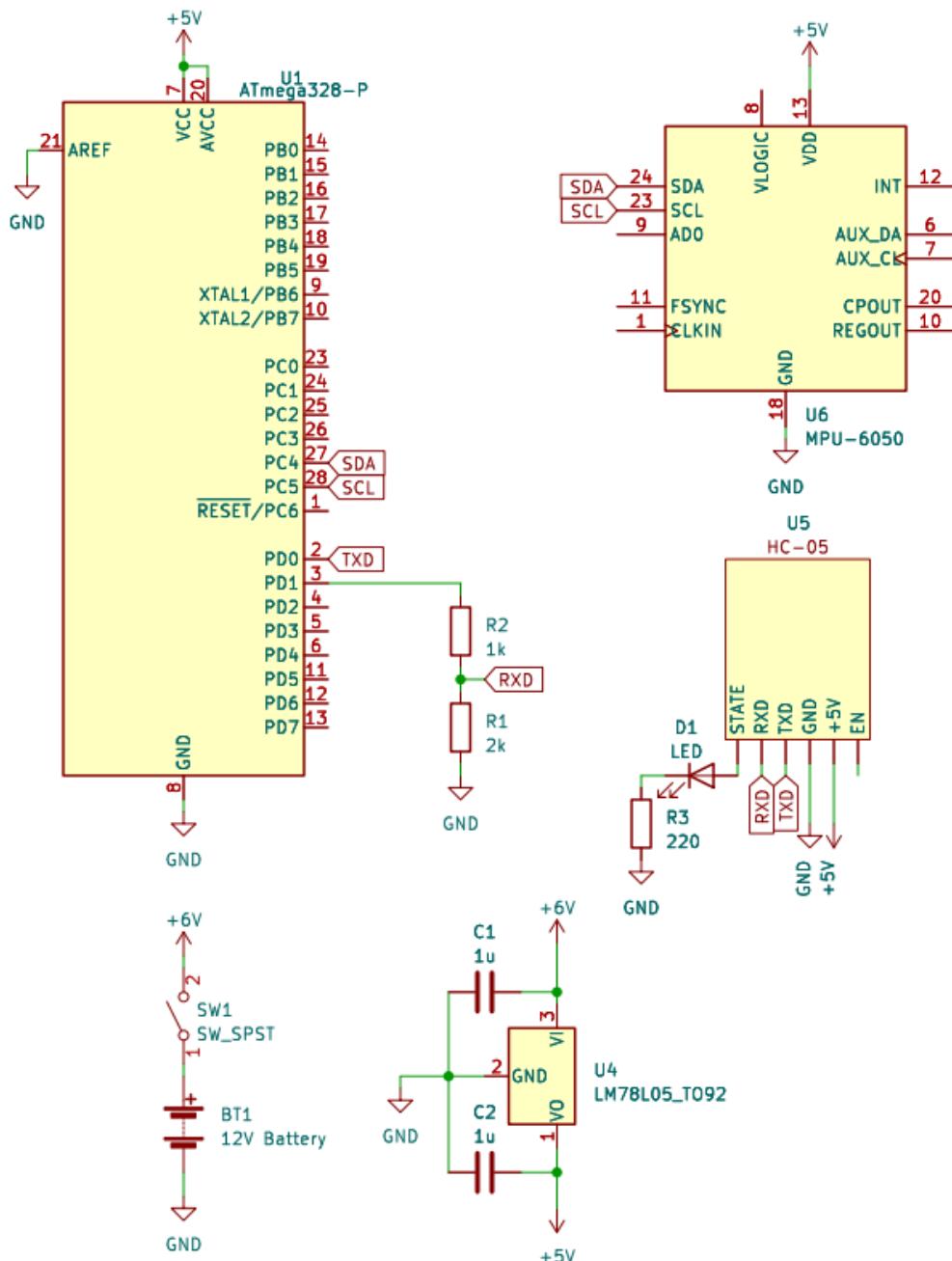
Projektet resulterade i en fungerande radiostyrd bil, som styrs trådlöst genom en accelerometer. Bilen reagerade korrekt på användarens handrörelser, vilket testades och verifierades. Genom att integrera olika komponenter såsom mikrokontroller, Bluetooth-modul, accelerometer och motordrivare, skapades en stabil och effektiv lösning för trådlös styrning av fordon.

Referenser

- [1] *EGBT-045MS and EGBT-046S Bluetooth Modules Hardware Manual & AT Commands Reference Manual Rev. 1r0.* e-Gizmo Mechatronix Central. URL: <https://content.instructables.com/FQ1/UVZ/HXA9PUVQ/FQ1CUVZHXA9PUVQ.pdf>.
- [2] Martyn Currey. *HC-05 and HC-06 zs-040 Bluetooth modules - First Look.* Personal Blog. Okt. 2014. URL: <https://www.martyncurrey.com/hc-05-and-hc-06-zs-040-bluetooth-modules-first-look/>.
- [3] Martyn Currey. *Arduino with HC-05 (ZS-040) Bluetooth module - AT MODE.* Personal Blog. Okt. 2014. URL: <https://www.martyncurrey.com/arduino-with-hc-05-bluetooth-module-at-mode/>.
- [4] Inven Sense Inc. *MPU-6000 and MPU-6050 Product Specification Revision 3.4.* URL: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>.
- [5] Inven Sense Inc. *MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.0.* URL: <https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/RM-MPU-6000A.pdf>.
- [6] Texas Instrument. *SN754410 Quadruple Half-H Drive.* URL: <https://www.ti.com/lit/ds/symlink/sn754410.pdf>.

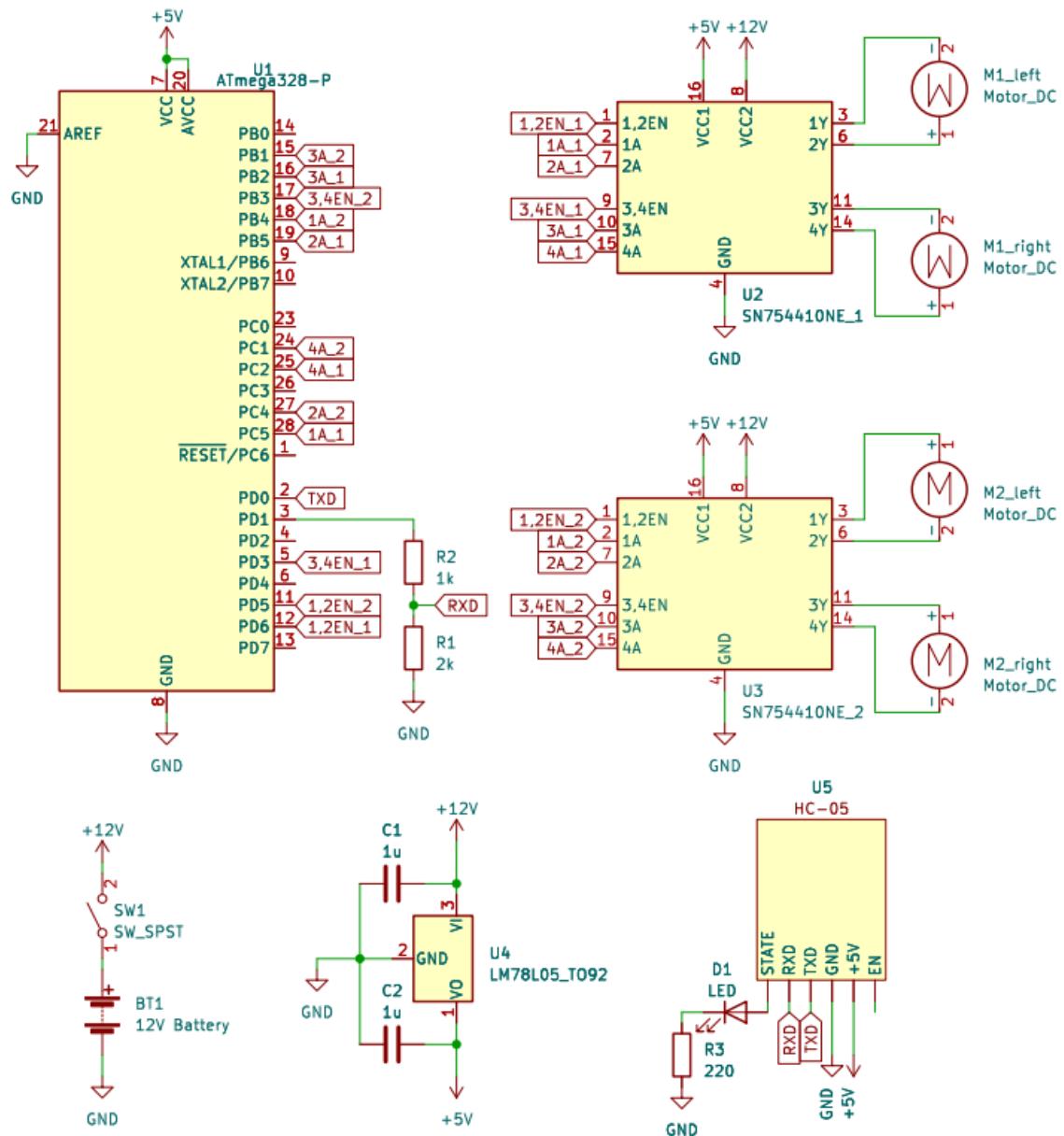
A Appendix kretsscheman

A.1 Kretsschema kontroller



Figur 8: Kressschema styenhet

A.2 Kretsschema bilen



Figur 9: Kresschema motorstyrning

B Appendix Kod

B.1 Bil

```
1 #include <avr/io.h>
2 #include <stdio.h>
3 #include <util/delay.h>
4 #include <avr/interrupt.h>
5 #include <stdlib.h>
6 #include "i2cmaster.h"
7 #include "twimaster.c"
8 #include <mpu.inc>
9 #include "uart.c"
10 #include "uart.h"

11
12 #define AVG 8
13 #define UART_BAUD_RATE 9600
14
15 volatile float x_angle=0,y_angle=0;
16 volatile int32_t accelX=0,accelY=0,accelZ=0;
17 volatile int16_t Ax=0, Ay=0, Az=0;
18 volatile uint8_t k =0;
19
20 void init(void){
21     //PB 1,2,4,5 - fram t ports
22     //PC 1,2,4,5 - Bak t ports, Xor med forward f r att altid ha en
23     //high
24     //PD 3,5,6 & PB 3 - PWM portar till Enable pin p drivarkretsen
25     DDRB = 0b00111110;
26     DDRD = 0b01101000;
27     DDRC = 0b00110110;
28
29     // setting up TIMER0
30     TCCROA = (1 << COM0A1) | (0 << COM0A0) | (1 << COM0B1) | (0 <<
31     COM0B0)
32         | (1 << WGM01) | (1 << WGM00 );
33     TCCROB = (0 << WGM02 ) | (1 << CS02) | (0 << CS01) | (1 << CS00 )
34     ;
35
36     TCCR2A = (1 << COM2A1) | (0 << COM2A0) | (1 << COM2B1) | (0 <<
37     COM2B0)
38         | (1 << WGM21) | (1 << WGM20 );
39     TCCR2B = (0 << WGM22 ) | (1 << CS22) | (0 << CS21) | (1 << CS20 )
40     ;
41
42     uart_init( UART_BAUD_SELECT(UART_BAUD_RATE ,F_CPU) );
43     sei();
44 }
45
46 int main(void){
47     init();
48     OCROA = 0;
```

```

44 OCROB = 0;
45 OCR2A = 0;
46 OCR2B = 0;
47 uint8_t forward = 0b00110110;
48 uint8_t motor_right = 0b00110000;
49 uint8_t motor_left = 0b00000110;
50 uint8_t right = 0;
51 uint8_t left = 0;
52 uint8_t direction = 0;
53 uint16_t timer = 0;
54 unsigned int c;
55
56 while(1){
57     c = uart_getc();
58
59     if (c & UART_NO_DATA){ // no data available from UART (lower
60         byte: received byte from ringbuffer,
61         // higher byte: last receive status. 0m
62         du inte fattar holla p Peter Fleury:s UART dokummentation!!!)
63         if (timer >= 500){
64             right = 0;
65             left = 0;
66             direction = 0;
67         }
68         else{
69             timer = 0;
70             switch(c & 0b11000000){
71                 case 0b01000000:
72                     right = c<<2;
73                     break;
74                 case 0b10000000:
75                     left = c<<2;
76                     break;
77                 case 0b11000000:
78                     direction = c & 0b00000111;
79                     break;
80             }
81         }
82     }
83
84     if (direction & 0x01){
85         PORTB = forward;
86         PORTC = 0x00;
87     }
88     else if (direction & 0x02){
89         PORTB = motor_right;
90         PORTC = forward^motor_right;
91     }
92     else if(direction & 0x04){
93         PORTB = motor_left;

```

```

94     PORTC = forward^motor_left;
95 }
96 else{
97     PORTB = 0x00;
98     PORTC = forward;
99 }
100
101 OCROA = right;
102 OCROB = right;
103
104 OCR2A = left;
105 OCR2B = left;
106
107 }
108 }
```

B.2 Kontroller

```

1 #include <avr/io.h>
2 #include <avr/interrupt.h>
3 #include <util/delay.h>
4 #include <uart.h>
5 #include <i2cmaster.h>
6 #include <twimaster.c>
7 #include <mpu.inc>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <avr/interrupt.h>
11 #include <avr/wdt.h>
12
13 #define UART_BAUD_RATE 9600
14 #define AVG 8
15
16 volatile int32_t accelX=0, accelY=0, accelZ=0;
17 volatile int16_t Ax=0, Ay=0, right = 0, left = 0;
18
19 void get_data(void){
20     accelX = (MPU6050_readSensor16(MPU6050_ACCEL_XOUT_L,
21                                     MPU6050_ACCEL_XOUT_H) + (AVG-1)*accelX)/AVG;
22     accelY = (MPU6050_readSensor16(MPU6050_ACCEL_YOUT_L,
23                                     MPU6050_ACCEL_YOUT_H) + (AVG-1)*accelY)/AVG;
24     accelZ = (MPU6050_readSensor16(MPU6050_ACCEL_ZOUT_L,
25                                     MPU6050_ACCEL_ZOUT_H) + (AVG-1)*accelZ)/AVG;
26 }
27
28 void scaled_data(){
29     Ax = accelX/64;
30     Ay = accelY/64;
31     right = (Ax + Ay);
32     left = (Ax - Ay);
33 }
```

```

32
33 void init()
34 {
35 // DDRB = 0x00000001; // LED
36 uart_init( UART_BAUD_SELECT(UART_BAUD_RATE ,F_CPU) );
37 i2c_init();
38 MPU6050_writeSensor(MPU6050_PWR_MGMT_1 , 0);
39
40 cli();
41 //reset watchdog
42 wdt_reset();
43 //set up WDT interrupt
44 WDTCSR = (1<<WDCE) | (1<<WDE) |(0<<WDIE) | (0<<WDP3)
45 | (0<<WDP2) | (1<<WDP1) | (1<<WDP0);
46 //Enable global interrupts
47 sei();
48 }
49
50 int main(void)
51 {
52 uint8_t dir = 0, tempL = 0, tempR = 0;
53
54 init();
55
56 for(;;)
57 {
58 wdt_reset();
59 get_data();
60 scaled_data();
61
62 // Left, right
63 if (abs(right)>255){
64 right = 255*((right > 0) - (right < 0));
65 }
66 if (abs(left)>255){
67 left = 255*((left > 0) - (left < 0));
68 }
69
70 // Direction
71 if (right>=0 && left>=0){
72 dir = 1;
73 }
74 else if (right>=0 && left < 0){
75 dir = 2;
76 }
77 else if(right<0 && left>=0){
78 dir = 4;
79 }
80 else{
81 dir = 8;
82 }

```

```
84     tempL = abs(left);
85     tempR = abs(right);
86
87     // Code the messages
88     tempL = (tempL >> 2) | 0b10000000;
89     tempR = (tempR >> 2) | 0b01000000;
90     dir = dir | 0b11000000;
91
92     // Send through UART
93     uart_putc(tempR);
94     uart_putc(tempL);
95     uart_putc(dir);
96     uart_putc('\r');
97
98     _delay_ms(5);
99 }
100
101 return 0;
102 }
```