Zelfrijdende auto

Thomas more | CAMPUS DE NAYER

Project 2

Thomas Eckert – MAxim aelterman

2016

# Inhoud

[1 Inhoud 2](#_Toc451714957)

[2 Voorwoord 4](#_Toc451714958)

[3 Dankwoord 4](#_Toc451714959)

[4 Taakverdeling 4](#_Toc451714960)

[5 Inleiding 5](#_Toc451714961)

[6 Doelstelling 5](#_Toc451714962)

[7 Hardware 6](#_Toc451714963)

[7.1 Blokschema 6](#_Toc451714964)

[7.2 Voeding 7](#_Toc451714965)

[7.3 Microcontroller 7](#_Toc451714966)

[7.4 Auto 7](#_Toc451714967)

[7.5 Afstandssensoren 7](#_Toc451714968)

[7.6 Motor driver auto 8](#_Toc451714969)

[7.7 Bedradingsschema 8](#_Toc451714970)

[8 Software 9](#_Toc451714971)

[8.1 De main-routine 9](#_Toc451714972)

[8.2 Interrupt-routine 10](#_Toc451714973)

[9 Broncode bespreking 11](#_Toc451714974)

[9.1 Includes/declaraties/defines 11](#_Toc451714975)

[9.1.1 Globale variabelen 12](#_Toc451714976)

[9.1.2 Functie declaraties 12](#_Toc451714977)

[9.2 Main 13](#_Toc451714978)

[9.2.1 Poort declaratie 13](#_Toc451714979)

[9.2.2 Interrupt declaratie 13](#_Toc451714980)

[9.2.3 Main zelf 14](#_Toc451714981)

[9.2.4 Hoofd functies – autonoom 16](#_Toc451714982)

[9.2.5 Hoofd functies – manueel 18](#_Toc451714983)

[9.3 Interrupt routines 20](#_Toc451714984)

[9.3.1 Korte uitleg 20](#_Toc451714985)

[9.3.2 HC-SR04 20](#_Toc451714986)

[9.3.3 Code 21](#_Toc451714987)

[9.4 Functies 22](#_Toc451714988)

[9.4.1 Timer functies 22](#_Toc451714989)

[9.4.2 Motor sturing functies 22](#_Toc451714990)

[10 Besluit 24](#_Toc451714991)

[10.1 Doelstelling bereikt? 24](#_Toc451714992)

[10.2 Wat zouden we anders hebben aangepakt? 24](#_Toc451714993)

[11 Resultaat (foto’s) 25](#_Toc451714994)

[12 Bijlagen 27](#_Toc451714995)

[12.1 Code 27](#_Toc451714996)

[12.2 Datasheet – HC-SR04 (Afstand sensoren) 35](#_Toc451714997)

[12.3 Atmega datasheet 38](#_Toc451714998)

[12.3.1 Pinout 38](#_Toc451714999)

[12.3.2 Interrupts 39](#_Toc451715000)

[12.3.3 Timers 44](#_Toc451715002)

# Voorwoord

Wij zijn Maxim en Thomas en wij zitten in het tweede jaar ICT/Elektronica aan Thomas More campus De Nayer. Als opdracht voor het vak Project 2, hebben wij ervoor gekozen om een zelfrijdende auto te maken.

Thomas Eckert - thomas.eckert@student.thomasmore.be

Maxim Aelterman - maxim.aelterman@student.thomasmore.be

# Dankwoord

Graag bedanken wij Wim Dams om ons te helpen tijdens de lessen bij het oplossen van de problemen die we hadden tijdens het maken van het project. Ook dank aan onze medestudenten voor het gebruiken van materiaal en componenten.  
Ook danken wij onze ouders voor de financiering van het project.

# Taakverdeling

Maxim & Thomas:

* Documenteren / Verslag schrijven
* Afstandssensoren inlezen
* Motoren uitsturen
* Testprogramma’s schrijven

Maxim:

* Programma voor zelfrijdende modus
* Maken van website

Thomas:

* Maken van print
* Maken van PowerPoint

# Inleiding

De bedoeling van het vak “project 2” is het bedenken en uitwerken van idee tot realisatie. Op het einde van het eerste jaar kwamen we met het idee om een auto met afstandssensoren te maken; dit leek ons dus het gepaste onderwerp voor project 2.

# Doelstelling

De oorspronkelijke bedoeling was om een auto te maken met 3 modi:

* Automatisch remmen voor obstakels met manuele controle
* Volledig autonome modus
* Leermodus

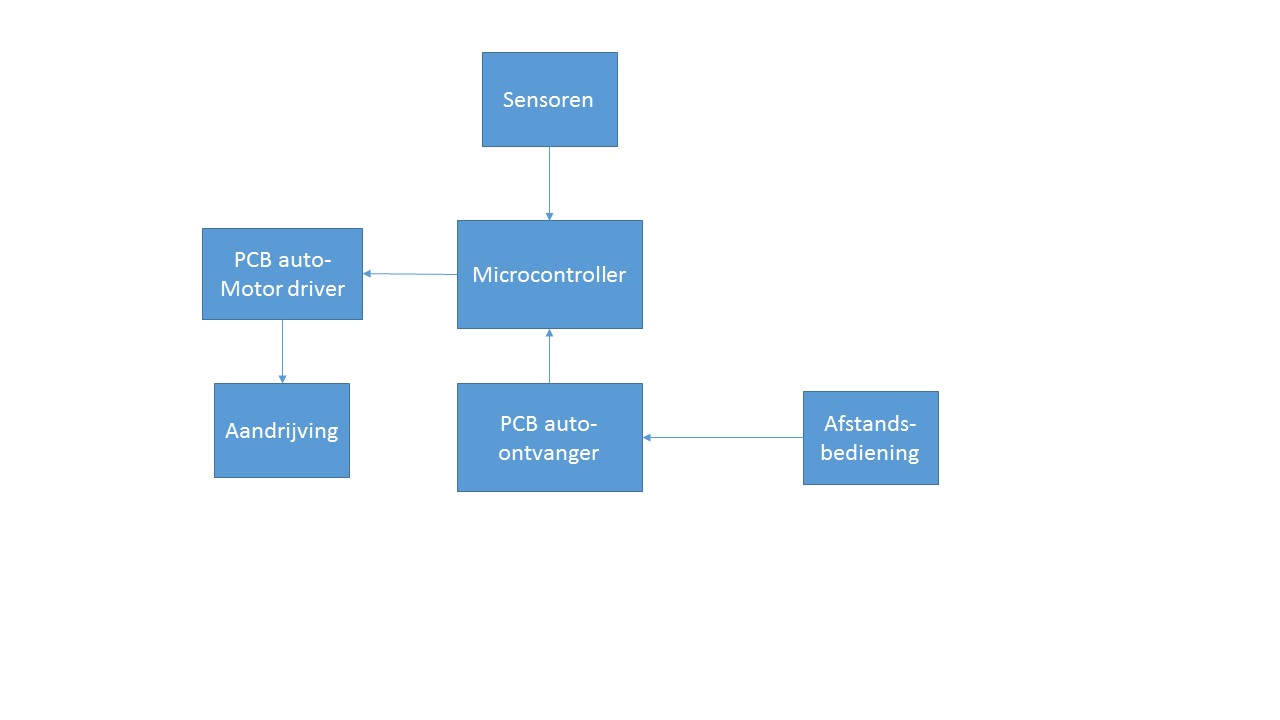
Door tijdsgebrek, en problemen met onze microcontroller, zijn we niet aan de laatste modus toegekomen. Onze kennis van C uit het eerste jaar is goed van pas gekomen, omdat we niet de overstap gemaakt hebben naar de zogenaamde “Cortex” borden, maar bij onze vertrouwde microcontroller van vorig jaar zijn gebleven.  
De reden hiervoor is dat de Atmega328p voldoet aan de vereisten van het project, en we toch volgend jaar les gaan krijgen over de cortex-bordjes.

# Hardware

In dit hoofdstuk vindt u alle info over de hardware die we gebruikt hebben, alsook de aansluitingen.  
Al onze printen zijn met de hand gesoldeerd omdat we nog niet wisten hoe we alles gingen aansluiten voor de deadline van de print-ontwerpen.

De blokschema’s zijn gemaakt in MS Visio.

## Blokschema



Figuur : Hardware blokschema

We lezen de afstandsbediening in via de originele pcb van de auto. Deze signalen nemen we binnen in de microcontroller, die nakijkt of de gekozen richting wel toegestaan is. De microcontroller stuurt dan de motor driver chip van de auto aan.  
De microcontroller wordt gevoed door de batterijen van de auto (4x1.5V).

## Voeding

Aangezien de auto op 6V werkt, hebben we besloten om deze spanning te gebruiken om de microcontroller mee te voeden. De microcontroller kan gevoed worden tussen de 1.8 en 5.5 v. Daarom plaatsen we een diode, om de spanning zo met 0.7V te verminderen, zodat we altijd in het werkbaar gebied zitten.  
De datasheet van de gebruikte batterijen kan u vinden bij de bijlagen.

## Microcontroller

Wij hebben in ons project gekozen voor een ATmega328P omdat dit een veelzijdige microcontroller is.

Daarom wordt hij ook gebruikt in de Arduino bordjes.  
Ook heeft hij veel uitgangen die zowel als input als output kunnen geschakeld worden. Wat onze keuze ook bepaald heeft, is dat hij redelijk gemakkelijk te programmeren is mits aanschaffing van een programmer.   
Ook is dit een redelijk goedkope microcontroller en was dit een van de weinige die op de site van onze componenten te verkrijgen was. En omdat we hier vorig jaar mee gewerkt hebben.

## Auto

De auto is gekozen op basis van aanbod, en beschikbare oppervlakte om wijzigingen aan te brengen. Ook mocht hij niet al te duur zijn, om de kosten wat te drukken.

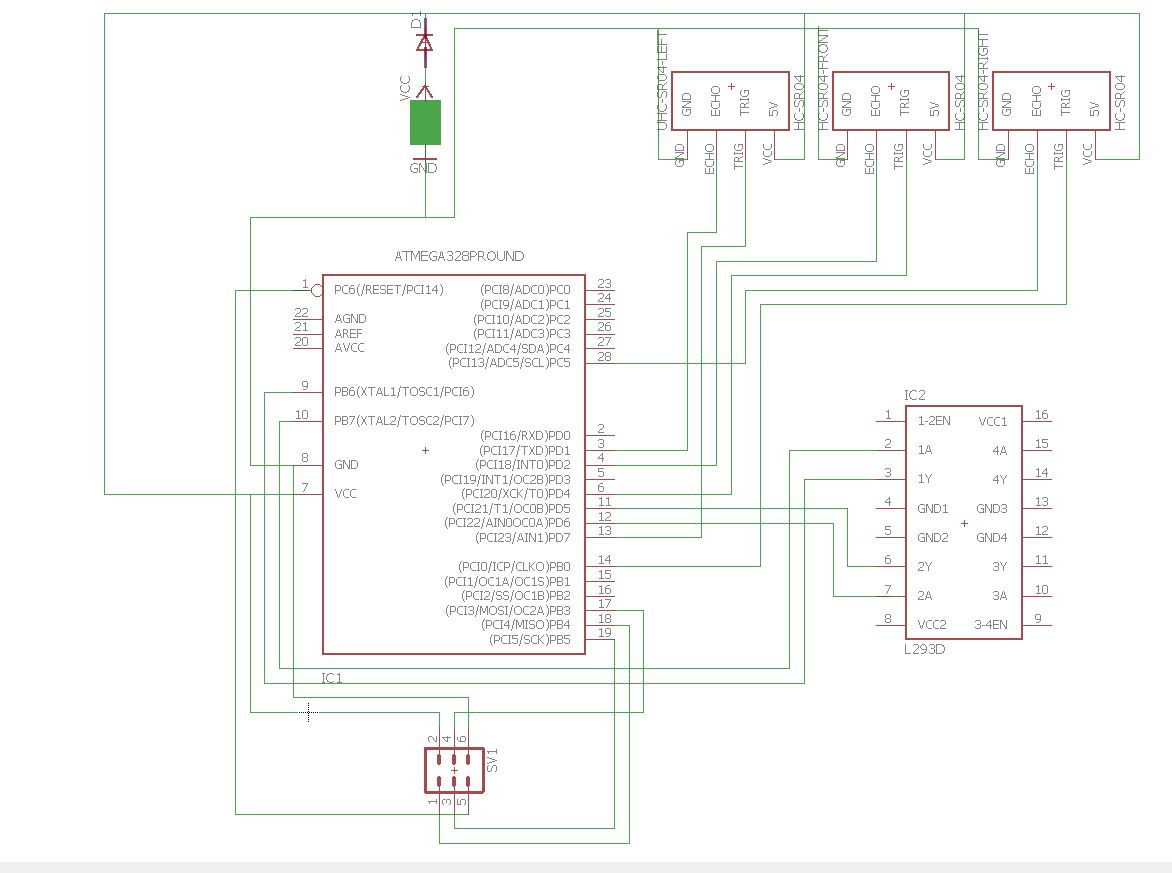
## Afstandssensoren

We beslisten al vroeg dat we wilden werken met ultrasone sensoren en geen infrarood. De HC-SR04 zijn vaak gebruikte sensoren, waar veel info over te vinden is op het internet, en die een goede range hebben. Perfect voor ons project dus. De datasheet hiervan kan u bij de bijlagen bekijken.

## Motor driver auto

We waren in het begin van plan om via een aparte motor driver ic de motoren uit te sturen. Een dergelijke chip is reeds te vinden op de auto zelf, en kan mits wat soldeerwerk hergebruikt worden, om de motoren mee te sturen. Van de chip zelf hebben we spijtig genoeg geen info gevonden op het net. Één enkele site had een pinout van een soortgelijke chip waar we op verder konden gaan, maar voor de zekerheid hebben we alle uitgangen nagemeten met een scoop.

## Bedradingsschema



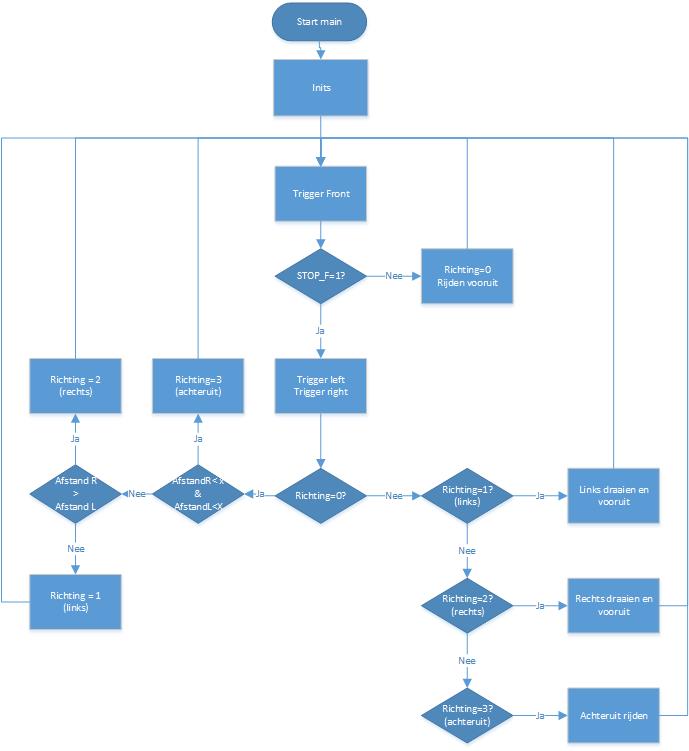
Figuur : Bedradingsschema

# Software

In dit hoofdstuk gaan we de software van ons project bespreken.

De software die we gebruikt hebben om de programma’s te schrijven is Atmel Studio 6. We hebben onze software in C omdat dit ons het gemakkelijkste leek; we hebben namelijk al meer geleerd over C dan over assembler.

## De main-routine



Figuur : main routine autonoom rijden

Figuur 3 is de flowchart van onze main routine. Na een trigger signaal, volgt een echo signaal uit onze sensoren, deze leveren elk een aparte interrupt op, die een andere routine start( zie hieronder). Stop\_f is een globale variabele die in de interrupt wordt gemanipuleerd.

## Interrupt-routine



Figuur : interrupt routine autonoom rijden

In de interrupt-routine, starten en stoppen we verschillende timers (naargelang welke interrupt vector opgeroepen word). De waardes die hieruit komen vormen we om naar een voor ons bekende maat/afstand zodat we hiermee kunnen rekenen/vergelijken. Als de waardes onder een voorafgestelde waarde komen, sturen we een stop signaal, dat in de main gelezen kan worden dmv globale volatile variabelen.

# Broncode bespreking

## Includes/declaraties/defines

#include <avr/io.h>

#define F\_CPU 8000000

#include <avr/interrupt.h>

#include <util/delay.h>

#define UITGANGV PB6

#define UITGANGA PB7

#define UITGANGL PD5

#define UITGANGR PD6

#define INGANGV PC3

#define INGANGA PC2

#define INGANGL PC1

#define INGANGR PC0

#define ECHO\_F PD2

#define ECHO\_L PD1

#define ECHO\_R PC5

#define TRIGGER\_F PD4

#define TRIGGER\_L PD7

#define TRIGGER\_R PB0

#define AFSTAND 20

Hierboven ziet u alle includes en globale declaraties.

Includen wil zeggen het toevoegen van standaard libraries om zo een werkend programma te verkrijgen.

Avr/io.h heb je nodig om in- en uitgangen te kunnen sturen.

F\_CPU zegt aan het programma op welke frequentie de klok draait, zodat de delayfunctie weet hoelang hij moet wachten.

Avr/delay.h dient om de delay instructies te laten werken.

Avr/interrupt.h zorgt ervoor dat je de functies die met interrupt werken inschakelt.

Hierna volgen de declaraties van de pinnen.

### Globale variabelen

volatile uint8\_t stop\_F = 0; //als stop == 0 dan geen object gedetecteerd

volatile uint8\_t stop\_L = 0;

volatile uint8\_t stop\_R = 0;

volatile uint8\_t afstand\_L = 0;

volatile uint8\_t afstand\_R = 0;

Dit zijn de globale variabelen. Normaal gezien moeten deze zoveel mogelijk vermeden worden, maar wanneer de getallen moeten aangepast worden in de main én in de interrupt, dan kunnen we niet anders dan globale variabelen gebruiken.  
Hier hebben we ook gekozen om onze variabelen “volatile” te maken, zodat deze in sneller geheugen terecht komen. De kleinste optie voor een variabele in onze ontwikkelomgeving en microcontroller is 8 bit. Spijtig genoeg zetten we meestal slechts 1 bit. Hierdoor gebruiken we meer plaats dan het getal eigenlijk nodig heeft.

### Functie declaraties

void autonoom();

void manueel();

void timer1Start();

void timer1Stop();

void timer0Start();

void timer0Stop();

void timer2Start();

void timer2Stop();

void vooruit(*uint8\_t*);

void achteruit(*uint8\_t*, *uint8\_t*);

void links();

void rechts();

void centreer();

Dit zijn de functiedeclaraties. Deze moeten gebeuren zodat het programma en de functie weet wat voor soort getal hij meekrijgt, en wat hij eventueel moet terugsturen.

In dit geval zijn deze allemaal void, dit wil zeggen dat deze geen returnwaarde bevatten en ook geen waardes meekrijgen. Dit omdat zij enkel een aantal registers wijzigen om timers te starten/stoppen, of motoren aan sturen.

De eerste twee functies worden vanuit de main opgeroepen: dit zijn onze twee hoofd functies.

De timerStart en timerStop functies starten en stoppen de timers, zoals de naam al doet vermoeden.

Vooruit en achteruit sturen de aandrijvingsmotoren aan, links, rechts en centreer passen de oriëntatie van de wielen aan.

## Main

### Poort declaratie

DDRB |= (1<<UITGANGV);

DDRB |= (1<<UITGANGA);

DDRD |= (1<<UITGANGL);

DDRD |= (1<<UITGANGR);

DDRC &= ~(1<<INGANGV);

DDRC &= ~(1<<INGANGA);

DDRC &= ~(1<<INGANGL);

DDRC &= ~(1<<INGANGR);

DDRD &= ~(1<<ECHO\_F);

DDRD &= ~(1<<ECHO\_L);

DDRC &= ~(1<<ECHO\_R);

DDRD |= (1<<TRIGGER\_F);

DDRD |= (1<<TRIGGER\_L);

DDRB |= (1<<TRIGGER\_R);

Hier zeggen we welke poort gebruikt wordt als ingang/uitgang. Door het register op “1” te zetten door middel van een “OF” poort wordt dit een uitgang, een “0” is dan voor een ingang.

### Interrupt declaratie

sei();

EICRA |= (1<<ISC01)| (1<<ISC00);

EIMSK |= (1<<INT0);

PCICR |= (1<<PCIE2);

PCICR |= (1<<PCIE1);

PCMSK2 |=(1<<PCINT17);

PCMSK1 |=(1<<PCINT13);

TCNT0 = 0;

TCNT1 = 0;

TCNT2 = 0;

OCR1A = 38000;

TCCR1B |= (1<<WGM12);

Sei is de verkorte versie om in het interrupt register de globale interrupts te enablen.  
In het EICRA register zetten we 2 bits. ISC01 en ISC00. Dit is om een interrupt te enablen op de rising edge van het echosignaal van de voorste afstandssensor.

In het EIMSK register zetten we 1 bit om de external interrupt van pin0 actief te maken, zodat deze werkt op falling/rising edge.

PCICR is het register voor de pin change interrupts. Hier zetten we pci2 voor interrupts 23-16 en pci1 voor interrupt 14-8

PCMSK2/1 is dan om deze interrupts van verschillende pinnen te maskeren tot alleen de interrupts die je in je project nodig hebt. We hebben er slechts 1 van elk nodig.

Hierna zetten we alle timer registers op 0, zodat we zeker weten dat ze leeg zijn voor we beginnen.

OCR1A is de waarde waarmee timer1 zal vergelijken en een compare match zal oproepen. Dit is na 38ms (dus niets gedetecteerd); hier doen we in principe niets mee.

### Main zelf

int main(void)

{

// poort declaratie(IO) //

DDRB |= (1<<UITGANGV);

DDRB |= (1<<UITGANGA);

DDRD |= (1<<UITGANGL);

DDRD |= (1<<UITGANGR);

DDRC &= ~(1<<INGANGV);

DDRC &= ~(1<<INGANGA);

DDRC &= ~(1<<INGANGL);

DDRC &= ~(1<<INGANGR);

DDRD &= ~(1<<ECHO\_F);

DDRD &= ~(1<<ECHO\_L);

DDRC &= ~(1<<ECHO\_R);

DDRD &= ~(1<<MODE\_SEL1);

DDRD &= ~(1<<MODE\_SEL2);

DDRD |= (1<<TRIGGER\_F);

DDRD |= (1<<TRIGGER\_L);

DDRB |= (1<<TRIGGER\_R);

//interrupt initialiseren

sei(); //global interrupts enable (SREG interrupts 1)

EICRA |= (1<<ISC01)| (1<<ISC00); //rising edge interrupt controle (ISC00 op 0 en isc 01 op 1 voor falling)

EIMSK |= (1<<INT0); //enablen van external interrupt 0

//PCICR voor enablen van pin change interrupt[23-16] enz.

//PCMSK voor enablen van apparte interrupts?

//PCIFR bevat de interrupt flags

PCICR |= (1<<PCIE2);

PCICR |= (1<<PCIE1); //enablen van pin change interrupt 23tot 6 en 14 tot 8

PCMSK2 |=(1<<PCINT17); //enablen van pin change interrupt 17

PCMSK1 |=(1<<PCINT13); //enablen van pin change interrupt 13

TCNT0 = 0; //timer register op 0 zetten

TCNT1 = 0; //TCCR0B voor prescaler settings(CS02 op 1 voor 256 prescaler=>max range = 1.4m)

TCNT2 = 0; //TCCR2B ook op 256 prescaler => timer puls om de (8000000/256)==>1/31250==> om de 32 microseconden==> 32\*255 = max lengte van object ==>/58==>1.4m

//TCCR2B CS22=> 1 CS21 => 1 CS20==> 0

OCR1A = 38000;

TCCR1B |= (1<<WGM12); //WGM12= ctc mode , compraen met ocr1A ...CS11 is 8 bit prescaler => counter om de 1us

//TCNT1 = 0; //timer register op 0 zetten

while(1)

{

if(PIND & (1<<MODE\_SEL1))

{

autonoom();

}

if(PIND & (1<<MODE\_SEL2))

{

manueel();

}

*\_delay\_ms*(1000);

}

}

Dit is onze main zelf, de main bestaat voornamelijk uit poortdeclaraties en initialisaties van timers en interrupts. Binnen de oneindige loop wordt er gecheckt welke mode de gebruiker heeft geselecteerd. Zolang dat er geen mode geselecteerd is, wordt de modus om de seconde gecheckt.

De eigenlijke main zit in autonoom() en manueel(), we hebben deze in functies opgedeeld zodat de main veel overzichtelijker wordt.

### Hoofd functies – autonoom

void autonoom()

{

volatile *uint8\_t* stoppen\_auto = 0;

while(PIND & (1<<MODE\_SEL1))

{

//trigger signaal sturen

PORTD &= ~(1<<TRIGGER\_F);

PORTD |= (1<<TRIGGER\_F);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_F);

*\_delay\_ms*(60);

//eind trigger signaal

//trigger signaal LEFT sturen

PORTD &= ~(1<<TRIGGER\_L);

PORTD |= (1<<TRIGGER\_L);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_L);

//eind trigger signaal LEFT

*\_delay\_ms*(60);

//trigger signaal RIGHT sturen

PORTB &= ~(1<<TRIGGER\_R);

PORTB |= (1<<TRIGGER\_R);

*\_delay\_us*(10);

PORTB &= ~(1<<TRIGGER\_R);

//eind trigger RIGHT signaal

if(stop\_F == 0)

{

//vooruit rijden

stoppen\_auto = 0;

centreer();

vooruit(1);

}

else if(stop\_L==1)

{

rechts();

vooruit(1);

}

else if(stop\_R==1)

{

links();

vooruit(1);

}

else

{

if(stoppen\_auto == 0)

{

achteruit(1, 1); //remmen, functie 1

stoppen\_auto = 1;

}

//trigger signaal LEFT sturen

PORTD &= ~(1<<TRIGGER\_L);

PORTD |= (1<<TRIGGER\_L);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_L);

//eind trigger signaal LEFT

*\_delay\_ms*(60);

//trigger signaal RIGHT sturen

PORTB &= ~(1<<TRIGGER\_R);

PORTB |= (1<<TRIGGER\_R);

*\_delay\_us*(10);

PORTB &= ~(1<<TRIGGER\_R);

//eind trigger RIGHT signaal

*\_delay\_ms*(60); //wachten op echo signalen

if((afstand\_R >= afstand\_L) && (stop\_R == 0))

{

rechts();

vooruit(1);

}

else if(stop\_L == 0)

{

links();

vooruit(1);

}

else

{

centreer();

achteruit(0,1);

}

}

}

}

Dit is onze functie voor het autonoom rijden. De bedoeling van deze functie is dat de auto vooruit rijdt tot deze een object in de weg detecteert. Dan zal hij de richting kiezen waar hij het langste onafgebroken zou kunnen vooraleer terug te moeten veranderen van richting .

Zolang dat de overeenkomende mode select pin hoog blijft, zal het programma blijven lopen. Dankzij de vanzelfsprekende functie namen en de commentaar is deze functie gemakkelijk te begrijpen en heeft weinig uitleg nodig.

### Hoofd functies – manueel

void manueel()

{

while(PIND & (1<<MODE\_SEL2)) //mode pin == 1

{

//trigger signaal sturen

PORTD &= ~(1<<TRIGGER\_F);

PORTD |= (1<<TRIGGER\_F);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_F);

*\_delay\_ms*(30);

//eind trigger signaal

if((PINC & (1<<INGANGV)) && (stop\_F==0)) //gebruiker drukt vooruit en niets gedetecteerd

{

vooruit(2);

stoppen = 0;

if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

}

else if(PINC & (1<<INGANGA))

{

achteruit(0, 2); //achteruit, 2de functie

if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

}

else if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

if(stop\_F==1)

{

if(stoppen == 0)

{

if(stopdebounce < 2)

{

stopdebounce++;

}

else

{

stoppen = 1;

centreer();

achteruit(1, 2); //remmen, 2de functie

stopdebounce = 0;

}

}

}

else stopdebounce = 0;

}

stoppen = 0;

stop\_F = 0;

}

Dit is onze functie voor het manueel rijden, met assistentie van de microcontroller. De bedoeling van deze functie is dat de gebruiker gewoon kan rijden met de afstandsbediening. Indien de auto te dicht bij een obstakel komt zal de microcontroller de controle overnemen en remmen. Zolang het obstakel in de weg ligt, kan de bestuurder niet vooruit rijden.

Ook hier zal de functie blijven lopen totdat de mode select pin laag wordt.

## Interrupt routines

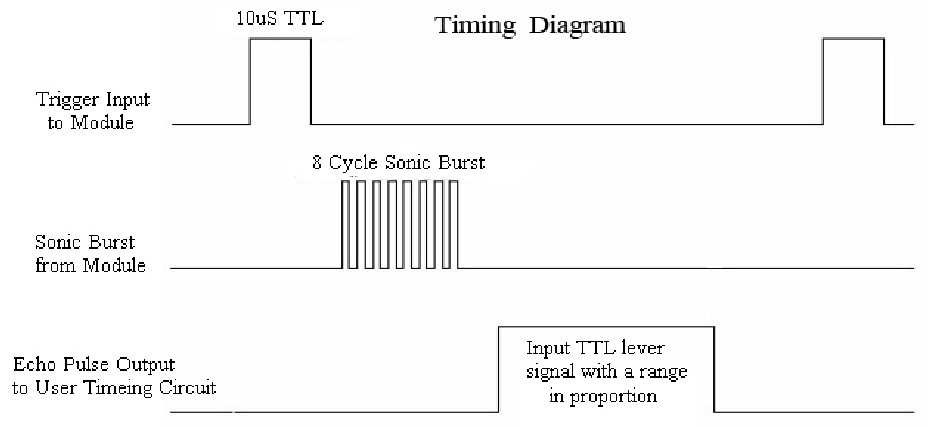
### Korte uitleg

We maken gebruik van 3 verschillende routines. Elke sensor, kan een aparte interrupt genereren, die op hun beurt een verschillende timer starten. De manier van werken is bij alle 3 hetzelfde, behalve dat we 2 8-Bit timers gebruiken en 1 16 bit timer. Zo hebben we onze prescalers een beetje moeten aanpassen, zodat we toch een voldoende groot bereik hadden, en moeten we daardoor ook met een andere waarde delen.

### HC-SR04

Een woordje uitleg bij de werking van onze afstandssensoren.  
om een echo signaal te verkrijgen, verwacht de HC-SR04 een trigger signaal van 10us. Wanneer hij dit signaal stuurt, zullen er 8 ultrasone pulsen worden uitgestuurd. De lengte van het echo signaal is afhankelijk van de afstand van een al dan niet gedetecteerd object.

Om dit signaal te meten maken wij gebruik van timers. Met een simpele wiskundige bewerking ( en een goed gekozen prescaler) kunnen we snel de lengte van dit signaal omzetten in een afstand, die door ons geïnterpreteerd kan worden.



### Code

ISR(INT0\_vect)

{

static uint8\_t status1 = 1; //als status==1 dan is de pin hoog. en omgekeerd

static uint16\_t timerwaarde1 =0;

if(status1==1)

{

timer1Start();

status1 = 0;

}

else

{

timer1Stop();

status1 = 1;

timerwaarde1 = TCNT1 / 58;

if(timerwaarde1 <= 20)

{

stop\_F = 1;

}

else

{

stop\_F = 0;

}

}

}

Deze routine wordt opgeroepen wanneer external interrupt 0 een verandering van edge ziet. Dit is dus voor onze voorste sensor.   
de eerste keer dat we hierin gaan, zetten we de status op 1. Status is in principe de indicator voor rising/falling edge. static wilt zeggen, dat hij alleen de eerste keer de status op 1 zal zetten.

Wanneer de status dus 1 is(rising edge van het echo-signaal) dan starten we de timer, en zetten we de status op 0 voor de volgende keer.

Wanneer de falling edge wordt gedetecteerd(einde van het echo-signaal) dan stoppen we de timer, halen we de waarde uit het register van de timer, delen we dit door 58 (zoals vermeld in de datasheet om naar centimeters om te zetten).

Wanneer de afstand kleiner is dan 20 cm, zetten we stop\_F op 1. Zo weet de main, dat er een object gedetecteerd is.

## Functies

### Timer functies

void timer1Start()

{

TCNT1 = 0;

EICRA &= ~(1<<ISC00); //volgende keer op faling edge interrupten

TCCR1B |= (1<<CS11); //prescaler(8) zetten = `klok starten

}

void timer1Stop()

{

EICRA |=(1<<ISC00); //volgende keer op rising edge interrupten

TCCR1B &= ~(1<<CS11);

}

We maken gebruik van 2 functies om elke timer afzonderlijk te stoppen en te starten. We bespreken enkel diegene van timer 1, omdat de andere quasi dezelfde zijn.

TCNT1 terug op 0 zetten voor de volgende keer (zodat de timer niet verder telt maar terug vanop 0 start)

Timers in AVR worden gestart door een prescaler te zetten, en gestopt door diezelfde prescaler weer op 0 te zetten.

### Motor sturing functies

void links()

{

PORTD &=~(1<<UITGANGR);

PORTD |=(1<<UITGANGL); //wielen naar links draaien

}

void rechts()

{

PORTD &=~(1<<UITGANGL);

PORTD |=(1<<UITGANGR); //wielen naar rechts draaien -> blijven in deze positie tot de weg voor de auto vrij is

}

void centreer()

{

PORTD &=~(1<<UITGANGL); //wielen centreren

PORTD &=~(1<<UITGANGR);

}

Hierboven ziet u de functies die we hebben gebruikt om onze wielen juist te positioneren. Een “|=“ betekent dat we deze poort hoog zetten, “&=~” betekent dat we deze poort laag zetten.

void vooruit(*uint8\_t* modus)

{

PORTB |=(1<<UITGANGV); //gas geven

if(modus == 1) *\_delay\_ms*(50); //niet te snel rijden (auto niet kapot rijden tijdens tests!)

else *\_delay\_ms*(20);

PORTB &=~(1<<UITGANGV);

}

void achteruit(*uint8\_t* remmen, *uint8\_t* modus) //remmen, modus

{

if (remmen == 1)

{

PORTB |=(1<<UITGANGA); //achteruit rijden

if(modus == 1) *\_delay\_ms*(300);

else *\_delay\_ms*(250);

PORTB &=~(1<<UITGANGA);

}

else

{

PORTB |=(1<<UITGANGA);

if(modus == 1) *\_delay\_ms*(10); //achteruit rijden

else *\_delay\_ms*(20);

PORTB &=~(1<<UITGANGA);

}

}

Dit zijn onze functies die de aandrijving van de auto voor hun rekening nemen. Deze zijn iets complexer omdat we de motoren niet op volle kracht willen aansturen, en voor de verschillende modes een andere snelheid hebben gekozen. De snelheid wordt voornamelijk bepaald door de tijd dat we de motoren laten draaien per programmacyclus.

Achteruit krijgt nog een extra argument mee dat bepaald of de auto moet remmen of achteruit moet rijden. Remmen moet met één commando kunnen gebeuren en daarom dus langer aan gestuurd worden dan gewoonweg achteruit rijden.

# Besluit

## Doelstelling bereikt?

Deels. We hebben een werkend prototype van een zelfrijdende auto, maar niet met alle functionaliteiten die we aan het begin van het project wilden implementeren. Het was immers ons plan om nog een derde functie te maken: De auto een traject leren door manueel te besturen, dit zou hij dan later kunnen reproduceren.

Dit komt omdat we enkele lessen problemen gehad hebben met onze microcontroller. Na elke debug-sessie kregen we deze niet meer geprogrammeerd, en probeerden we dit de rest van de les op te lossen. Doch kwam het er telkens op neer dat we bij een klasgenoot moesten aanbellen voor een nieuwe chip.

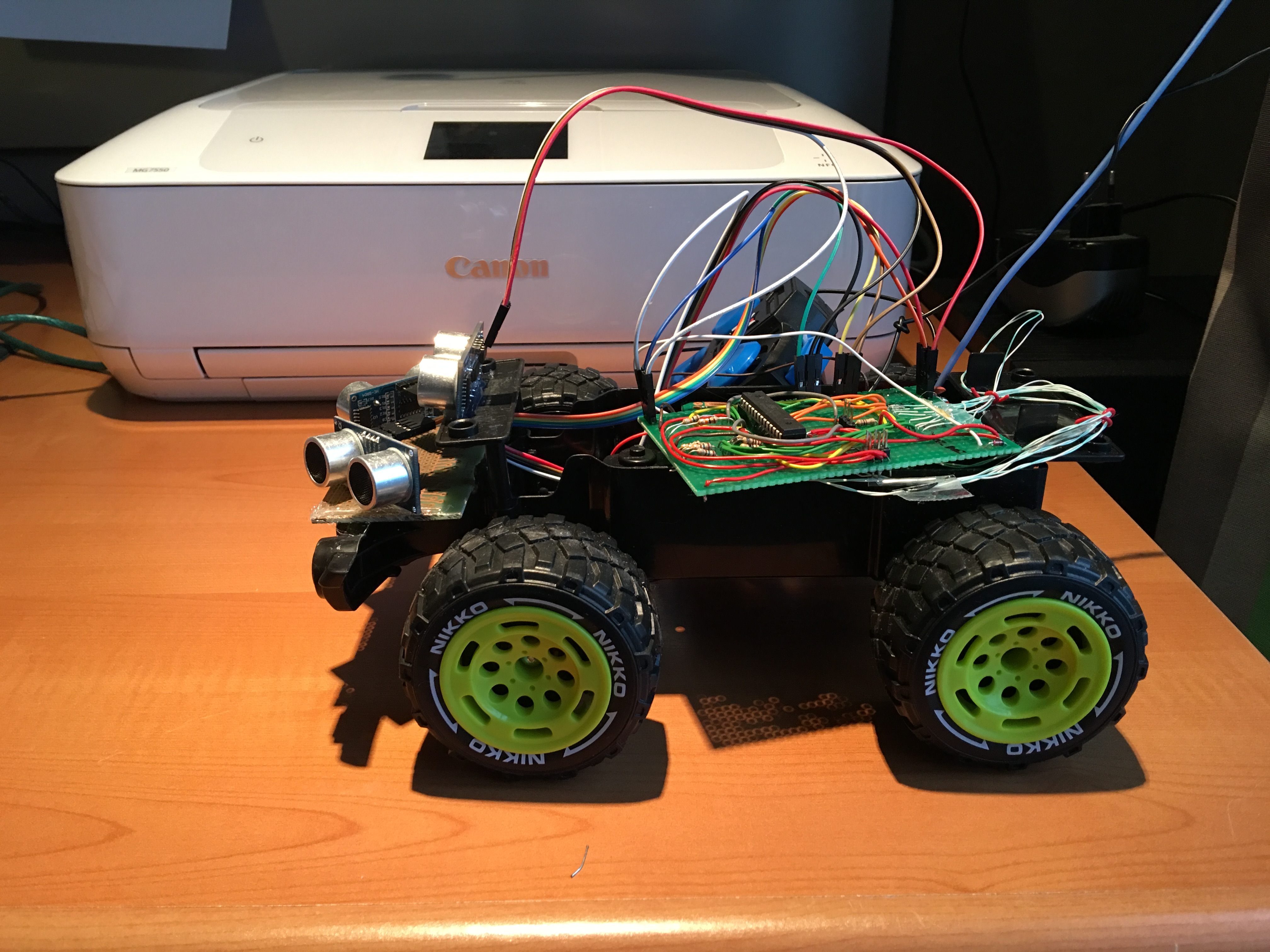
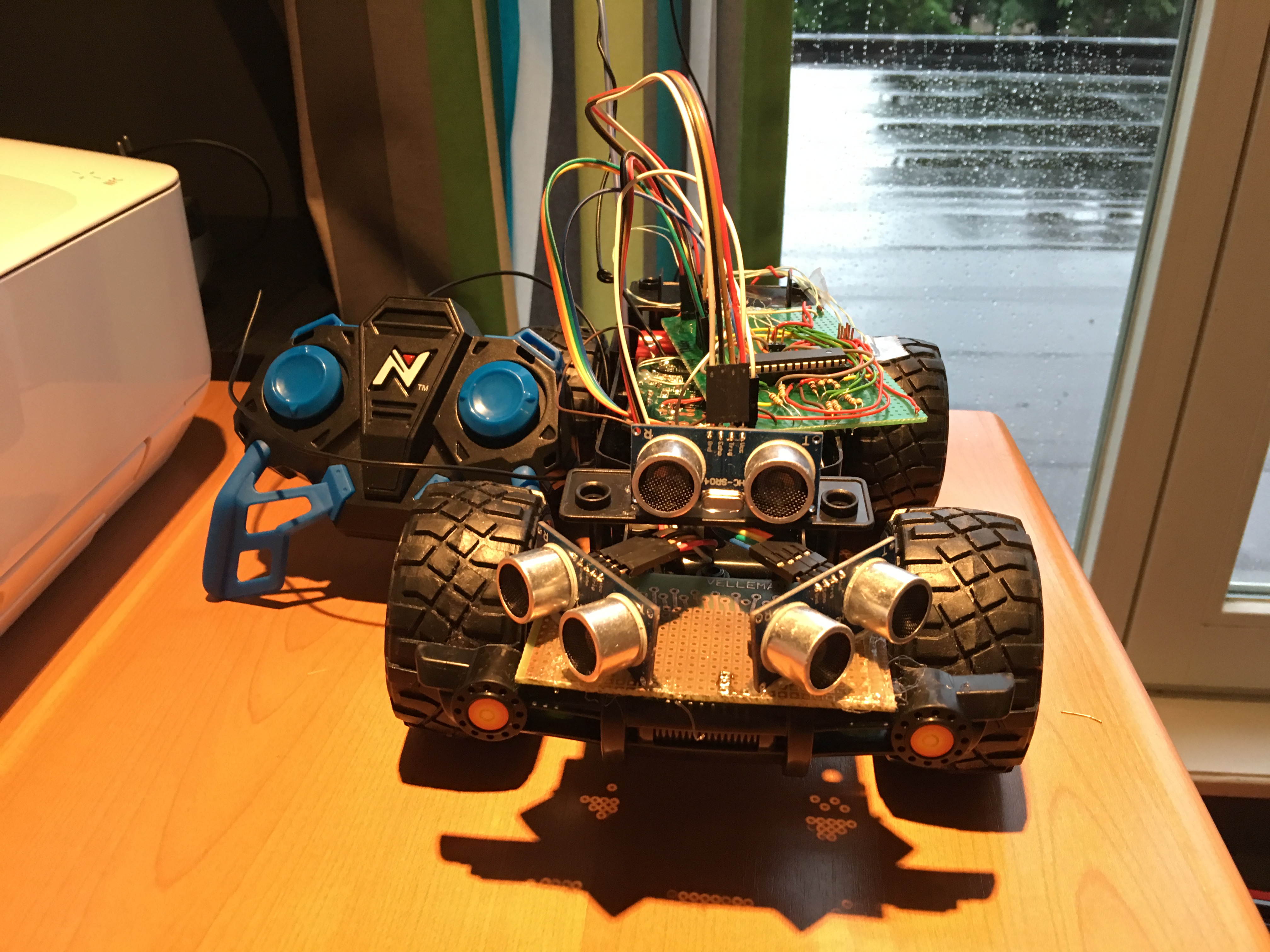
En misschien nog de belangrijkste les voor komende projecten: Als je iets van China moet laten komen is het goedkoop, maar de levertermijn is lang. Dit is nadelig als je snel een component nodig hebt. De kostprijs is dus omgekeerd evenredig met de levertermijn.

## Wat zouden we anders hebben aangepakt?

Meer taakverdeling, we hebben vaak aan “Extreme programming” gedaan tijdens de lessen en thuis. Hierdoor hebben we onze tijd niet efficiënt benut.

# Resultaat (foto’s)

Hieronder een enkele foto’s van het uiteindelijke resultaat van ons project.



# Bijlagen

## Code

// Inlcludes/Defines //

#include <avr/io.h>

#define *F\_CPU* 8000000

#include <avr/interrupt.h>

#include <util/delay.h>

#define UITGANGV PB6

#define UITGANGA PB7

#define UITGANGL PD5

#define UITGANGR PD6

#define INGANGV PC3

#define INGANGA PC2

#define INGANGL PC1

#define INGANGR PC0

#define ECHO\_F PD2

#define ECHO\_L PD1

#define ECHO\_R PC5

#define TRIGGER\_F PD4

#define TRIGGER\_L PD7

#define TRIGGER\_R PB0

#define MODE\_SEL1 PD0

#define MODE\_SEL2 PD3

#define AFSTAND 20

// Aanmaken snel geheugen om stop te schrijven //

volatile *uint8\_t* stop\_F = 0; //als stop == 0 dan geen object gedetecteerd

volatile *uint8\_t* stop\_L = 0;

volatile *uint8\_t* stop\_R = 0;

volatile *uint8\_t* afstand\_L = 0;

volatile *uint8\_t* afstand\_R = 0;

volatile *uint8\_t* stoppen = 0;

volatile *uint8\_t* stopdebounce = 0;

// Functie declaraties //

void autonoom();

void manueel();

void timer1Start();

void timer1Stop();

void timer0Start();

void timer0Stop();

void timer2Start();

void timer2Stop();

void vooruit(*uint8\_t*);

void achteruit(*uint8\_t*, *uint8\_t*);

void links();

void rechts();

void centreer();

// ###################### MAIN ######################

int main(void)

{

// poort declaratie(IO) //

DDRB |= (1<<UITGANGV);

DDRB |= (1<<UITGANGA);

DDRD |= (1<<UITGANGL);

DDRD |= (1<<UITGANGR);

DDRC &= ~(1<<INGANGV);

DDRC &= ~(1<<INGANGA);

DDRC &= ~(1<<INGANGL);

DDRC &= ~(1<<INGANGR);

DDRD &= ~(1<<ECHO\_F);

DDRD &= ~(1<<ECHO\_L);

DDRC &= ~(1<<ECHO\_R);

DDRD &= ~(1<<MODE\_SEL1);

DDRD &= ~(1<<MODE\_SEL2);

DDRD |= (1<<TRIGGER\_F);

DDRD |= (1<<TRIGGER\_L);

DDRB |= (1<<TRIGGER\_R);

//interrupt initialiseren

sei(); //global interrupts enable (SREG interrupts 1)

EICRA |= (1<<ISC01)| (1<<ISC00); //rising edge interrupt controle (ISC00 op 0 en isc 01 op 1 voor falling)

EIMSK |= (1<<INT0); //enablen van external interrupt 0

//PCICR voor enablen van pin change interrupt[23-16] enz.

//PCMSK voor enablen van apparte interrupts?

//PCIFR bevat de interrupt flags

PCICR |= (1<<PCIE2);

PCICR |= (1<<PCIE1); //enablen van pin change interrupt 23tot 6 en 14 tot 8

PCMSK2 |=(1<<PCINT17); //enablen van pin change interrupt 17

PCMSK1 |=(1<<PCINT13); //enablen van pin change interrupt 13

TCNT0 = 0; //timer register op 0 zetten

TCNT1 = 0; //TCCR0B voor prescaler settings(CS02 op 1 voor 256 prescaler=>max range = 1.4m)

TCNT2 = 0; //TCCR2B ook op 256 prescaler => timer puls om de (8000000/256)==>1/31250==> om de 32 microseconden==> 32\*255 = max lengte van object ==>/58==>1.4m

//TCCR2B CS22=> 1 CS21 => 1 CS20==> 0

OCR1A = 38000;

TCCR1B |= (1<<WGM12); //WGM12= ctc mode , compraen met ocr1A ...CS11 is 8 bit prescaler => counter om de 1us

//TCNT1 = 0; //timer register op 0 zetten

while(1)

{

if(PIND & (1<<MODE\_SEL1))

{

autonoom();

}

if(PIND & (1<<MODE\_SEL2))

{

manueel();

}

*\_delay\_ms*(1000);

}

}

// ###################### AUTONOME FUNCTIE ######################

void autonoom()

{

volatile *uint8\_t* stoppen\_auto = 0;

while(PIND & (1<<MODE\_SEL1))

{

//trigger signaal sturen

PORTD &= ~(1<<TRIGGER\_F);

PORTD |= (1<<TRIGGER\_F);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_F);

*\_delay\_ms*(60);

//eind trigger signaal

//trigger signaal LEFT sturen

PORTD &= ~(1<<TRIGGER\_L);

PORTD |= (1<<TRIGGER\_L);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_L);

//eind trigger signaal LEFT

*\_delay\_ms*(60);

//trigger signaal RIGHT sturen

PORTB &= ~(1<<TRIGGER\_R);

PORTB |= (1<<TRIGGER\_R);

*\_delay\_us*(10);

PORTB &= ~(1<<TRIGGER\_R);

//eind trigger RIGHT signaal

if(stop\_F == 0)

{

//vooruit rijden

stoppen\_auto = 0;

centreer();

vooruit(1);

}

else if(stop\_L==1)

{

rechts();

vooruit(1);

}

else if(stop\_R==1)

{

links();

vooruit(1);

}

else

{

if(stoppen\_auto == 0)

{

achteruit(1, 1); //remmen, functie 1

stoppen\_auto = 1;

}

//trigger signaal LEFT sturen

PORTD &= ~(1<<TRIGGER\_L);

PORTD |= (1<<TRIGGER\_L);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_L);

//eind trigger signaal LEFT

*\_delay\_ms*(60);

//trigger signaal RIGHT sturen

PORTB &= ~(1<<TRIGGER\_R);

PORTB |= (1<<TRIGGER\_R);

*\_delay\_us*(10);

PORTB &= ~(1<<TRIGGER\_R);

//eind trigger RIGHT signaal

*\_delay\_ms*(60); //wachten op echo signalen

if((afstand\_R >= afstand\_L) && (stop\_R == 0))

{

rechts();

vooruit(1);

}

else if(stop\_L == 0)

{

links();

vooruit(1);

}

else

{

centreer();

achteruit(0, 1);

achteruit(0,1);

achteruit(0,1);

}

}

}

}

// ###################### MANUELE FUNCTIE ######################

void manueel()

{

while(PIND & (1<<MODE\_SEL2))

{

//trigger signaal sturen

PORTD &= ~(1<<TRIGGER\_F);

PORTD |= (1<<TRIGGER\_F);

*\_delay\_us*(10);

PORTD &= ~(1<<TRIGGER\_F);

*\_delay\_ms*(30);

//eind trigger signaal

if((PINC & (1<<INGANGV)) && (stop\_F==0))

{

vooruit(2);

stoppen = 0;

if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

}

else if(PINC & (1<<INGANGA))

{

achteruit(0, 2); //achteruit, 2de functie

if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

}

else if(PINC & (1<<INGANGL))

{

links();

}

else if(PINC & (1<<INGANGR))

{

rechts();

}

else

{

centreer();

}

if(stop\_F==1)

{

if(stoppen == 0)

{

if(stopdebounce < 2)

{

stopdebounce++;

}

else

{

stoppen = 1;

centreer();

achteruit(1, 2); //remmen, 2de functie

stopdebounce = 0;

}

}

}

else stopdebounce = 0;

}

stoppen = 0;

stop\_F = 0;

}

// ###################### INTERRUPTS ######################

ISR(INT0\_vect)

{

static *uint8\_t* status1 = 1; //als status==1 dan is de pin hoog. en omgekeerd

static *uint16\_t* timerwaarde1 =0;

if(status1==1)

{

timer1Start();

status1 = 0;

}

else

{

timer1Stop();

status1 = 1;

timerwaarde1 = TCNT1 / 58;

if(timerwaarde1 <= 100)

{

stop\_F = 1;

}

else

{

stop\_F = 0;

}

}

}

ISR(PCINT1\_vect)

{

static *uint8\_t* status0 = 1; //als status==1 dan is de pin hoog. en omgekeerd

static *uint16\_t* timerwaarde0 =0;

if(status0==1)

{

timer0Start();

status0 = 0;

}

else

{

timer0Stop();

status0 = 1;

//timerwaarde1 = TCNT0 / 1856; //1 timer waarde om de 32 us. dus delen door 32 om 1 us te hebben, dan delen door 58 om naar cm te gaan

timerwaarde0 = TCNT0 ;

afstand\_R = timerwaarde0;

if(timerwaarde0 <= 30)

{

stop\_R = 1;

}

else

{

stop\_R = 0;

}

}

}

ISR(PCINT2\_vect)

{

static *uint8\_t* status2 = 1; //als status==1 dan is de pin hoog. en omgekeerd

static *uint16\_t* timerwaarde2 =0;

if(status2==1)

{

timer2Start();

status2 = 0;

}

else

{

timer2Stop();

status2 = 1;

timerwaarde2 = TCNT2;

afstand\_L = timerwaarde2;

if(timerwaarde2 <= 30)

{

stop\_L = 1;

}

else

{

stop\_L = 0;

}

}

}

ISR(TIMER1\_COMPA\_vect)

{

//niets gemeten

}

void timer1Start()

{

TCNT1 = 0;

EICRA &= ~(1<<ISC00); //volgende keer op faling edge interrupten

TCCR1B |= (1<<CS11);//prescaler(8) zetten = `klok starten

}

void timer1Stop()

{

EICRA |=(1<<ISC00); //volgende keer op rising edge interrupten

TCCR1B &= ~(1<<CS11);

}

void timer0Start()

{

TCNT0 = 0;

TCCR0B |= (1<<CS02);

}

void timer0Stop()

{

TCCR0B &= ~(1<<CS02);

}

void timer2Start()

{

TCNT2 = 0;

TCCR2B |= (1<<CS22);

TCCR2B |= (1<<CS21);

}

void timer2Stop()

{

TCCR2B &= ~(1<<CS22);

TCCR2B &= ~(1<<CS21);

}

// ###################### MOTORSTURING ######################

void vooruit(*uint8\_t* modus)

{

PORTB |=(1<<UITGANGV); //gas geven

if(modus == 1) *\_delay\_ms*(50); //niet te snel rijden (auto niet kapot rijden tijdens tests!)

else *\_delay\_ms*(20);

PORTB &=~(1<<UITGANGV);

}

void achteruit(*uint8\_t* remmen, *uint8\_t* modus) //remmen, modus

{

if (remmen == 1)

{

PORTB |=(1<<UITGANGA); //achteruit rijden

if(modus == 1) *\_delay\_ms*(300);

else *\_delay\_ms*(250);

PORTB &=~(1<<UITGANGA);

}

else

{

PORTB |=(1<<UITGANGA);

if(modus == 1) *\_delay\_ms*(10); //achteruit rijden

else *\_delay\_ms*(20);

PORTB &=~(1<<UITGANGA);

}

}

void links()

{

PORTD &=~(1<<UITGANGR);

PORTD |=(1<<UITGANGL); //wielen naar links draaien

}

void rechts()

{

PORTD &=~(1<<UITGANGL);

PORTD |=(1<<UITGANGR); //wielen naar rechts draaien -> blijven in deze positie tot de weg voor de auto vrij is

}

void centreer()

{

PORTD &=~(1<<UITGANGL); //wielen centreren

PORTD &=~(1<<UITGANGR);

}

## Datasheet – HC-SR04 (Afstand sensoren)



Tech Support:

services@elecfreaks.com

**Ultrasonic Ranging Module HC - SR04**

 **Product features:**

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

1. Using IO trigger for at least 10us high level signal,
2. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
3. IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time×velocity of sound (340M/S) / 2,

 **Wire connecting direct as following:**

5V Supply

Trigger Pulse Input

Echo Pulse Output

0V Ground

**Electric Parameter**

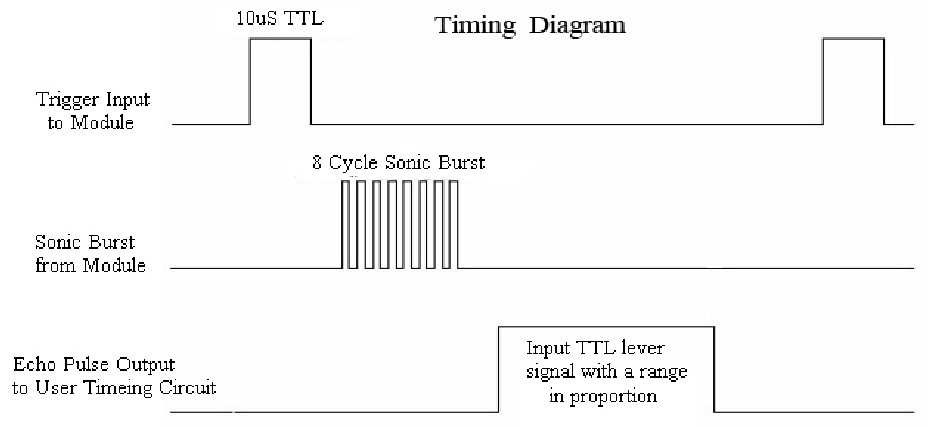
|  |  |
| --- | --- |
| **Working Voltage** | **DC 5 V** |
| **Working Current** | **15mA** |
| **Working Frequency** | **40Hz** |
| **Max Range** | **4m** |
| **Min Range** | **2cm** |
| **MeasuringAngle** | **15 degree** |
| **Trigger Input Signal** | **10uS TTL pulse** |
| **Echo Output Signal** | **Input TTL lever signal and the range in proportion** |
| **Dimension** | **45\*20\*15mm** |



**Vcc Trig Echo GND**

**Timing diagram**

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time \* velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



 **Attention:**

 The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.

 When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

**www.Elecfreaks.com**

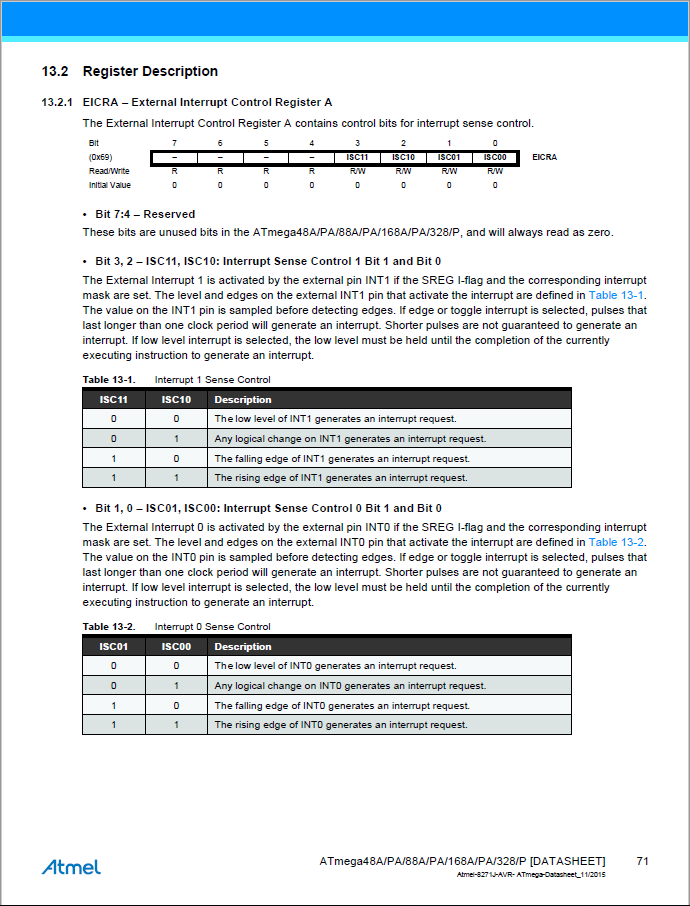
## Atmega datasheet

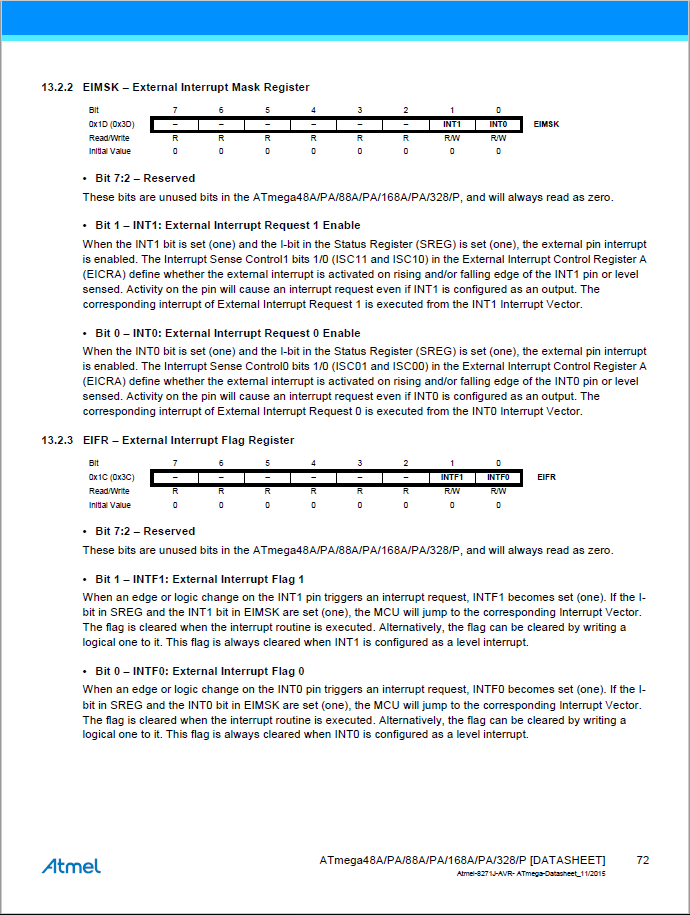
### Pinout

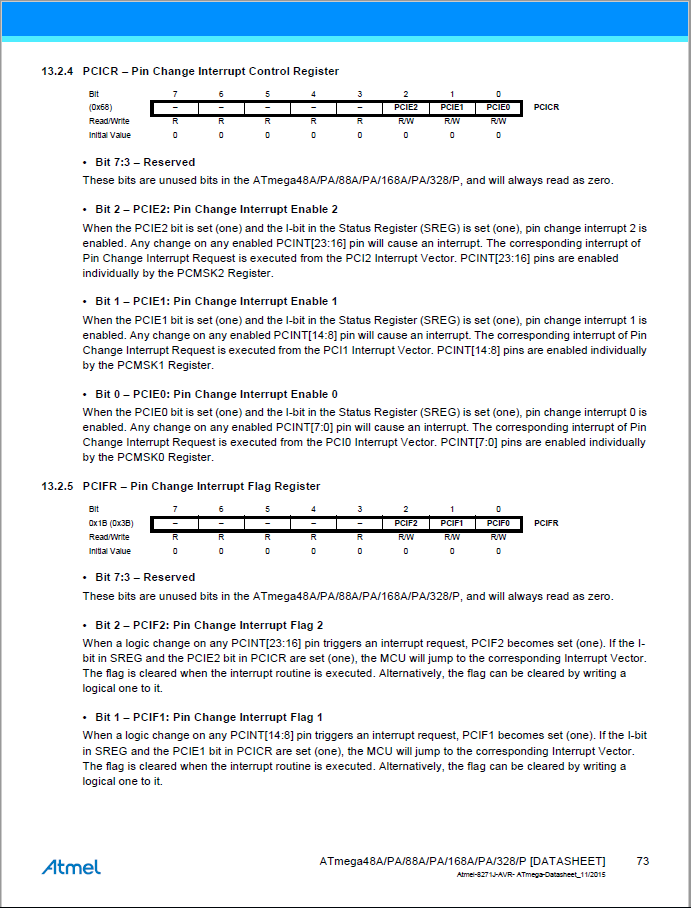


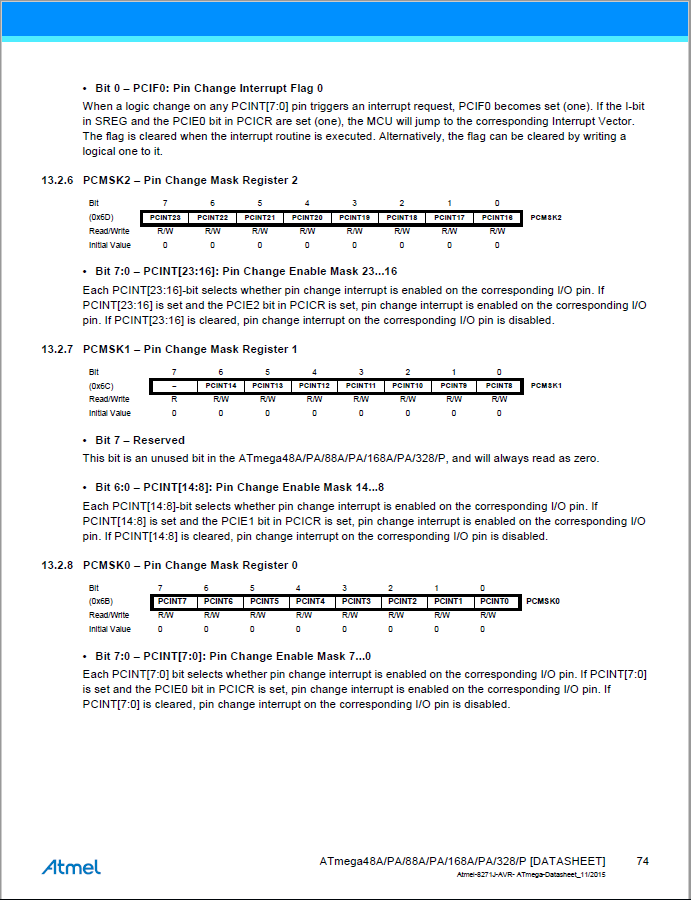
### Interrupts

### C:\Users\Maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\interrupt2.png









### Timers

