Zelfrijdende auto

Thomas more | CAMPUS DE NAYER

Project 2

Thomas Eckert – MAxim aelterman

2016

# Inhoud

[1 Inhoud 2](#_Toc451348376)

[2 Voorwoord 3](#_Toc451348377)

[3 Dankwoord 3](#_Toc451348378)

[4 Taakverdeling 3](#_Toc451348379)

[5 Inleiding 4](#_Toc451348380)

[6 Doelstelling 4](#_Toc451348381)

[7 Hardware 5](#_Toc451348382)

[7.1 Blokschema 5](#_Toc451348383)

[7.2 Voeding 6](#_Toc451348384)

[7.3 Microcontroller 6](#_Toc451348385)

[7.4 Auto 6](#_Toc451348386)

[7.5 Afstandssensoren 6](#_Toc451348387)

[7.6 Motor driver auto 7](#_Toc451348388)

# 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 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 bij de problemen die we hadden tijdens de lessen. Ook onze medestudenten voor het gebruiken van materiaal/componenten.  
Ook danken wij onze ouders om het grootste deel van ons project te financieren.

# Taakverdeling

Maxim & Thomas:

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

Maxim:

* Programma voor zelfrijdende modus

Thomas:

* Maken van print

# 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

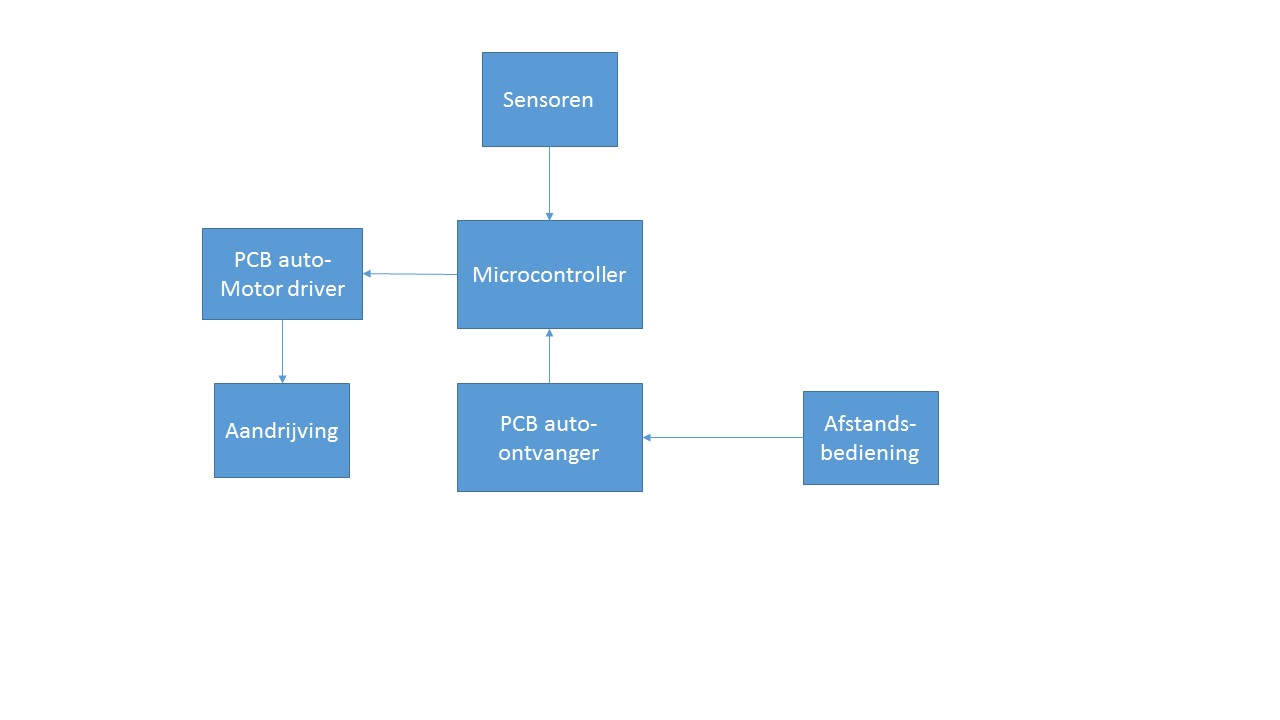
Door tijdsgebrek, en problemen met onze microcontroller, zijn we niet aan de laatste modus toegekomen. Onze kennis van C uit het eerste jaar is goed van pas gekomen, omdat we niet de overstap gemaakt hebben naar de zogenaamde “Cortex” borden, maar bij onze vertrouwde microcontroller van vorig jaar zijn 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 1: 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 bijstond 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.

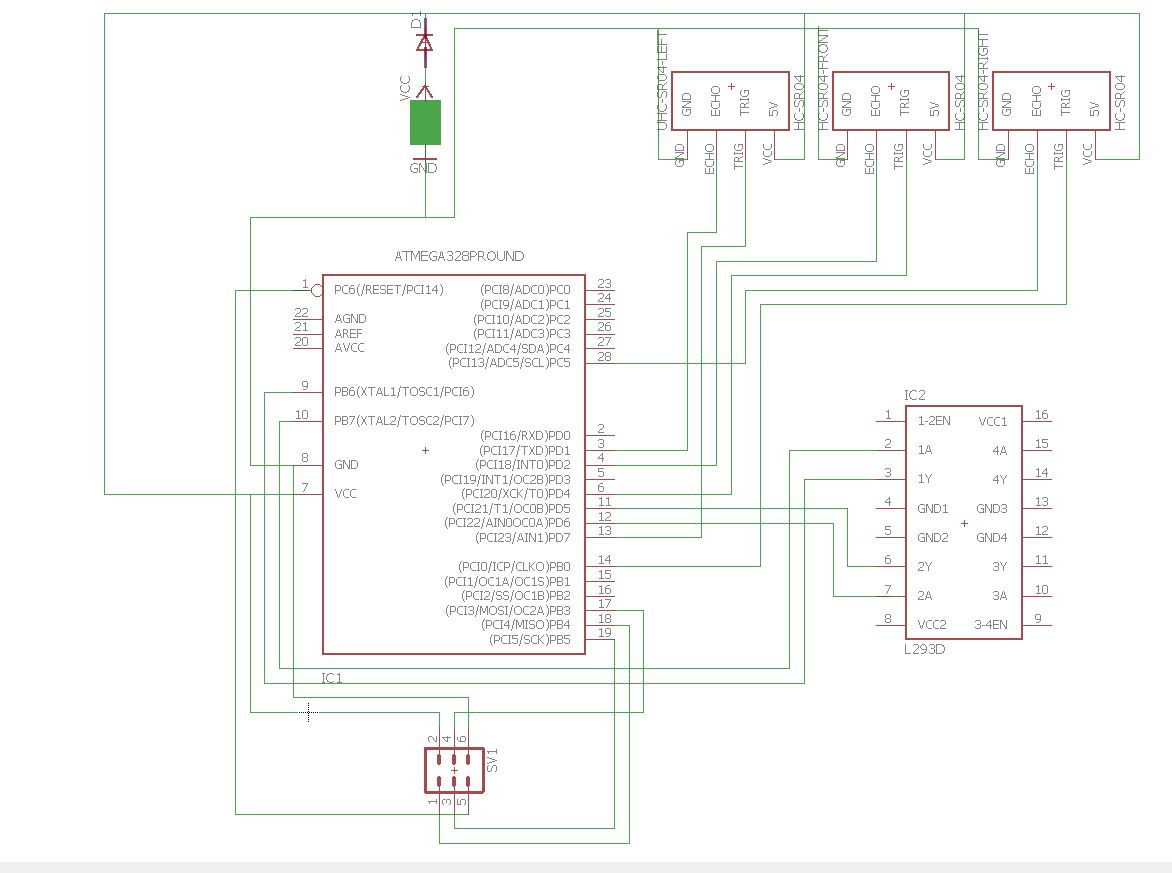
## Afstandssensoren

We beslisten al vroeg dat we wouden 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



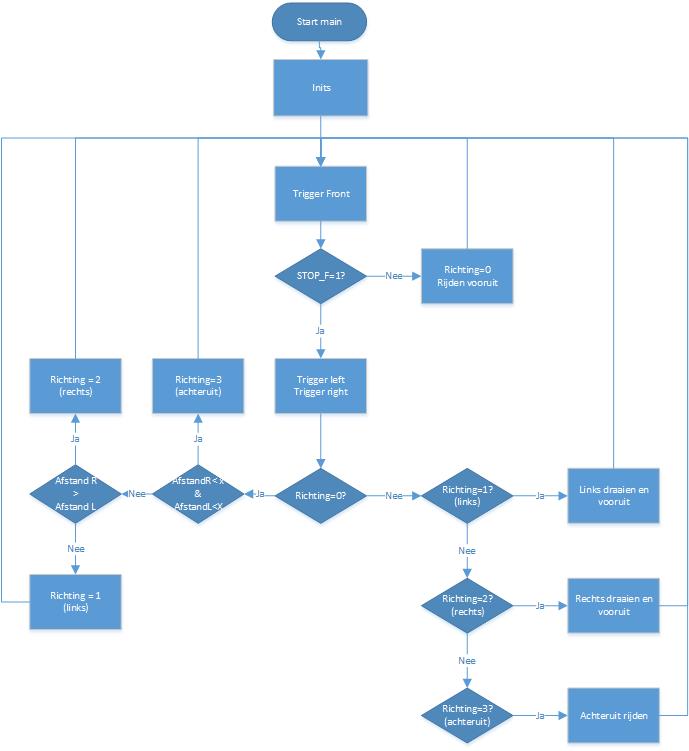
Figuur 2: Bedradingsschema

# 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.

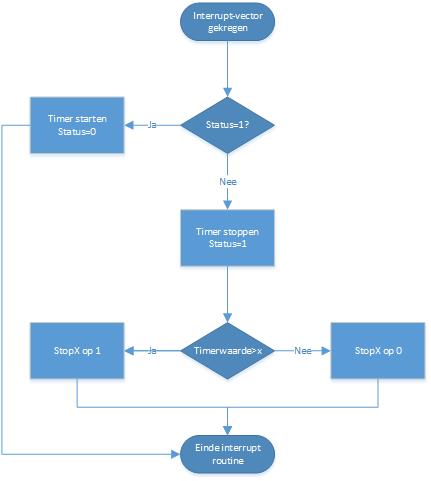
## De main-routine



Figuur 3: 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 4: 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;

volatile uint8\_t richting = 0; //0-vooruit 1-links 2-rechts 3-achteruit

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 programeeromgeving 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 timer1Start();

void timer1Stop();

void timer0Start();

void timer0Stop();

void timer2Start();

void timer2Stop();

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 moeten wijzigen om timers te starten/stoppen.

## 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 regsiter 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 alleeen 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 na 38ms (dus niets gedetecteerd) hier doen we in principe niets mee.

### Main zelf

while(1)

{

//trigger signaal sturen

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

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

\_delay\_us(10);

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

//eind trigger signaal

if(stop\_F == 0)

{

//vooruit rijden

richting = 0;

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

PORTD &=~(1<<UITGANGR);

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

\_delay\_ms(100); //niet te snel rijden (auto niet kapot rijden tijdens tests!)

PORTB &=~(1<<UITGANGV);

\_delay\_ms(100);

}

else

{

//trigger signaal LEFT sturen

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

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

\_delay\_us(10);

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

//eind trigger signaal LEFT

//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(30); //wachten op echo signalen

if(richting == 0)

{

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

{

//rechts rijden

richting = 2;

}

else if(stop\_L == 0)

{

//links rijden

richting = 1;

}

else

{

//achteruit rijden

richting = 3;

}

}

else

{

if(richting == 1) //links

{

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

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

\_delay\_ms(100);

PORTB &=~(1<<UITGANGV);

\_delay\_ms(100);

}

if(richting == 2) //rechts

{

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

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

\_delay\_ms(100);

PORTB &=~(1<<UITGANGV);

\_delay\_ms(100);

}

if(richting == 3) //achteruit

{

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

PORTD &=~(1<<UITGANGR);

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

\_delay\_ms(100);

PORTB &=~(1<<UITGANGA);

\_delay\_ms(100);

}

}

}

}

}

Dit is onze main zelf, we gaan deze niet lijn per lijn overlopen. De basis is:

* &=~(1<<X) 🡺 poort laag zetten
* |=(1<<X) 🡺poort hoog zetten
* De sensoren verwachten een trigger van 10 us. Dit signaal maken we zelf zoals hierboven staat.

## 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, buiten 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 extrenal 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.

Als 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.

Als 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).

Asl 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.

# 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.

Dit komt omdat we enkele lessen problemen gehad hebben met onze microcontroller. Na elke debug-sessie kregen we deze niet meer geprogrameerd, 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.