THOMAS MORE

Tank op afstandsbediening

Door: Robin Gogne

https://github.com/Rathioz/PE2_Controller
https://github.com/Rathioz/PE2_Tank

Academiejaar 2023 - 2024

Introductie

Aangezien ik voor practise enterprise 1 een tankje had gemaakt op afstandsbediening, wilde ik dit voor practise enterprise 2 ook doen. Hierdoor kan ik mijn kennis over de wereld van mechatronica uitbreiden en een aantal zaken verbeteren tegenover practise enterpise 1.

Het tankje bevat enkele functionaliteiten zoals de aandrijving van rupsbanden, het aansturen van de kanonnen en de kanonnentorens, en het afspelen van geluid door speakers. Ik heb geprobeerd zo accuraat mogelijk een originele mannelijke MARK IV Tanks na te bouwen.

Enkele verbeteringen tegenover PE1 zijn:
Mechanica:

- In het originele tankje trok ik de loop omhoog en omlaag via een touw aan een stepper motor. In dit project zou ik dit mechanisch willen aandrijven met tandwielen
- De aandrijving van de rupsbanden was in het originele project aanvaardbaar maar kan veel beter.
- De rupsbanden wil ik uit een ander materiaal maken aangezien PLA hard is en niet zoveel grip heeft.

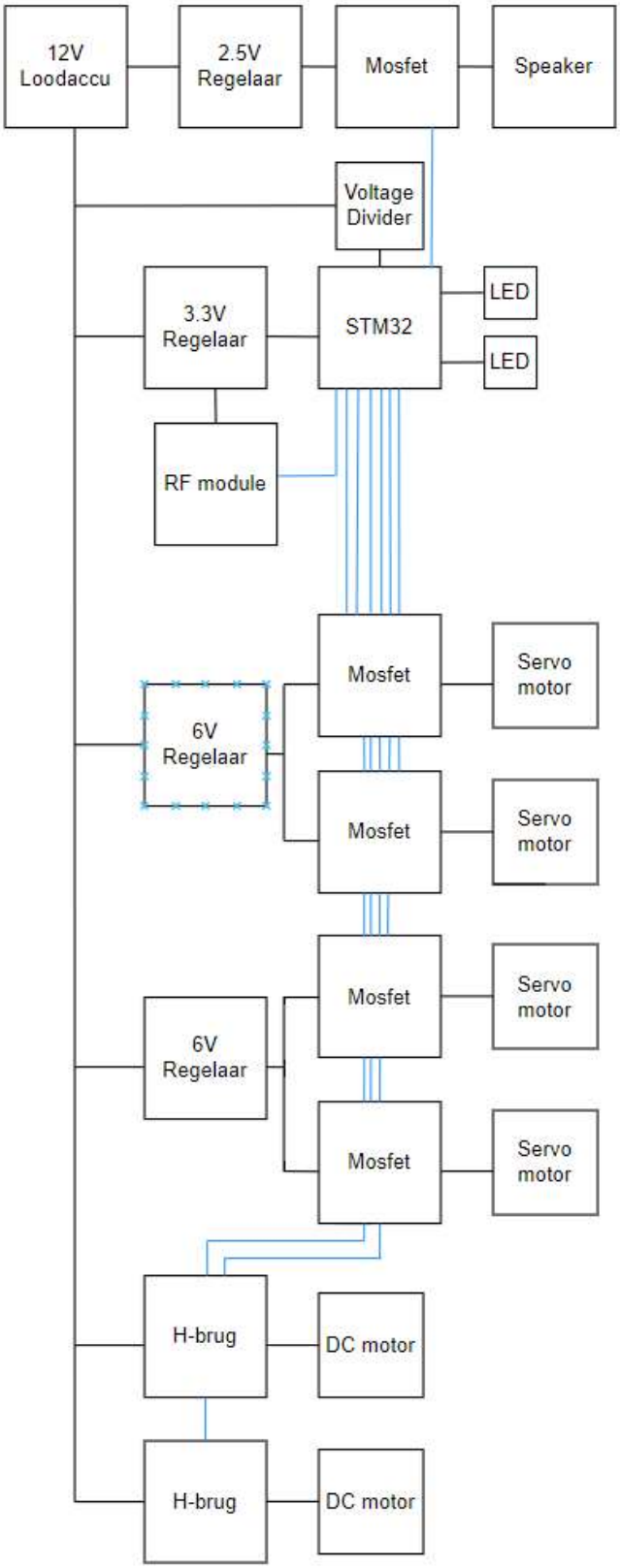
Hardware:

- Het originele project bevatte zeer veel modules. Deze keer wil ik dit tot een minimum beperken.

Software:

- Deze keer wil ik mijn code iets beter structureren door gebruik te maken van functies in plaats van alles in de main while loop te zetten.
- Enkele glitches tegenover vorig jaar uit het project halen.

Blokschema


12V
Loodaccu
2.5V
Regelaar
Mosfet
Speaker
Voltage
Divider
3.3V
Regelaar
STM32
LED
LED
RF module
Mosfet
Servo
motor
6V
Regelaar
Mosfet
Servo
motor
Mosfet
Servo
motor
6V
Regelaar
Mosfet
Servo
motor
H-brug
DC motor
H-brug
DC motor

Testen

Vooraleer ik aan dit project ben begonnen heb ik niet zoveel getest vanwege te weinig tijd door de combinatie van dit vak met de bachelorproef.

Het enige dat ik getest heb vooraleer ik ben begonnen aan het ontwerp van mijn PCB is het benodigde voltage voor de speaker en de werking van een servomotor.

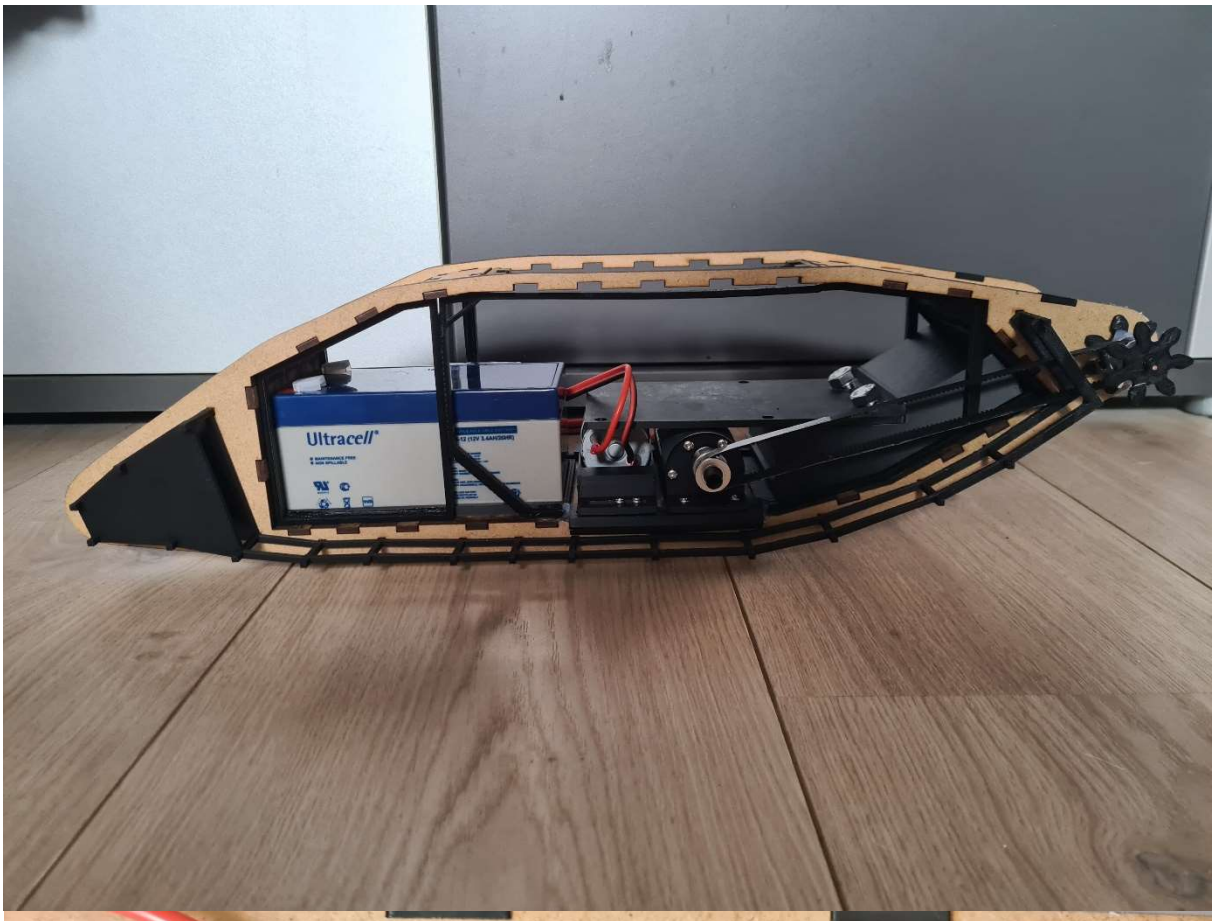
De speaker wordt aangestuurd door een PWM van de microcontroller. De microcontroller kan niet genoeg stroom leveren dus maak ik gebruik van een mosfet die een hogere stroom kan leveren aan de speaker. Ook merkte ik dat 3,3V voor enorm veel ruis veroorzaakte op de speaker. Hierdoor ben ik empirisch gaan bepalen wat een goed voltage is voor de speaker. Ik ben uitgekomen op 2,5V. Als ik hoger ga dan 2,5V komt er veel ruis op de speaker. Als ik onder 2,5V ga wordt de speaker minder luid.

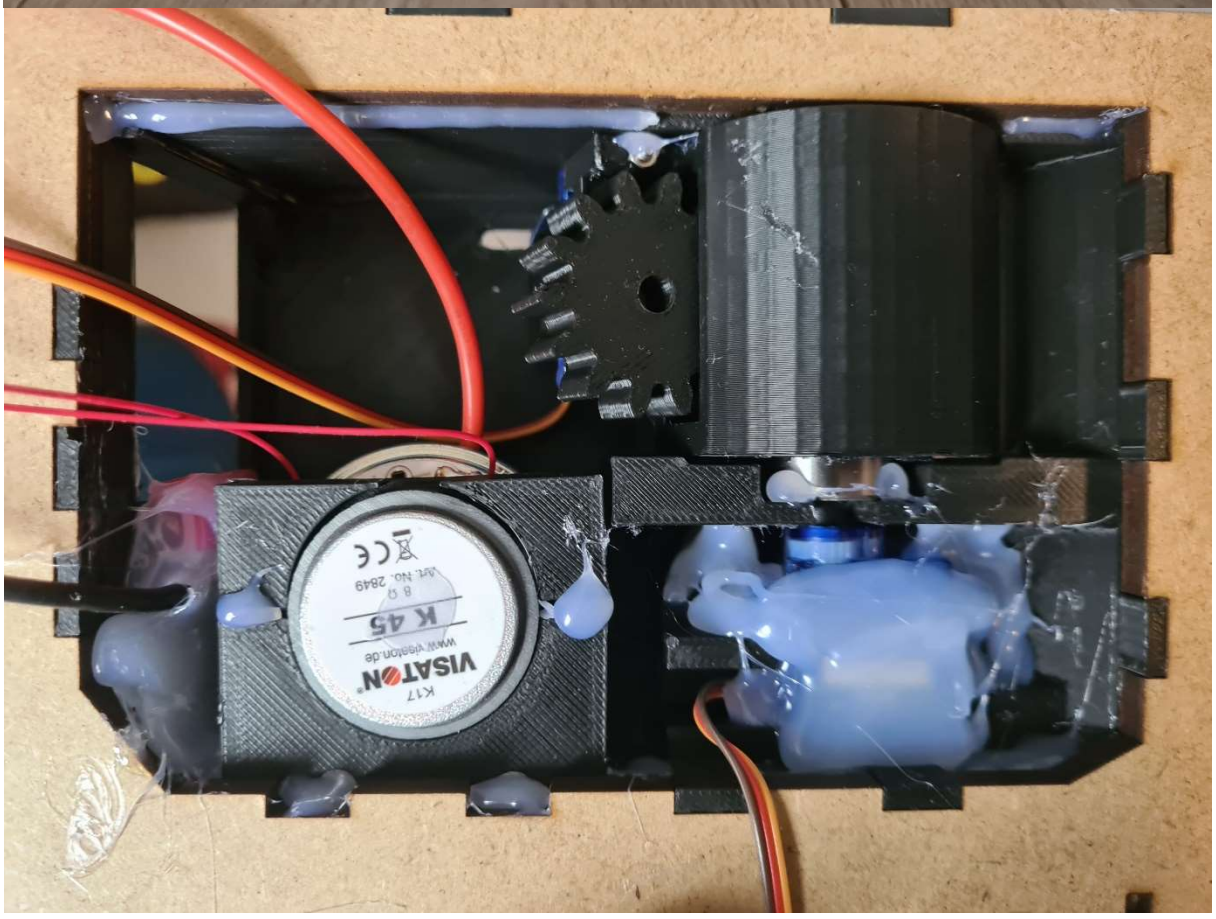
Ik had hiervoor al gewerkt met DC motoren en stepper motoren maar nog niet met servomotoren. Deze heb ik kort uitgetest via het development bord. Hier heb ik snel ontdekt dat een data signaal van 3,3V niet werkt, maar door hier een mosfet tussen te plaatsen die de PWM naar een hoger voltage brengt wel.

Beschrijving

1.1. Mechanica:

De behuizing van de tank bestaat vooral uit een groot deel MDF hout met een interne frame gemaakt van PLA waar alles aan geconnecteerd kan worden. In het middenstuk van de tank bevinden zich de DC motoren, de batterij en de PCB.
De DC motoren gaan de rupsbanden aandrijven via een riem. Aangezien de riem aangespannen moet worden, zijn de DC motoren bevestigd via een rail. De riem wordt bevestigd over een 1/1 connectie van de motoren naar de voorkant van de tank, waar het aandrijftandwiel zich bevindt van de rupsbanden. De rupsbanden zijn zo goed mogelijk visueel nagemaakt van het originele model. De rupsbanden zijn gemaakt uit TPU. Dit is een flexibel materiaal en zorgt voor meer grip tijdens het rijden. In de zijkanten van de tank bevinden zich de servomotoren, de speaker en/of de aan- en uitknop en/of de connector van de batterij. In de zijkant bevinden zich de kanonnen en de kanonnentorens. Deze moeten in een beperkte hoek kunnen draaien. Onder de kannonentoren is rechtstreeks een servomotor bevestigd. Het kanon zelf is via twee 1 op 1 tandwielen geconnecteerd aan de servo.


Ultracell


VISATON
K 45
K17

1.2. Hardware:

1.2.1. Microcontroller:

Als microcontroller heb ik gekozen voor de voorziene STM32F301C8T6 aangezien deze voldoet aan alle benodigdheden zoals genoeg timers voor alle motoren en speakers.

1.2.1.1. Aansluitingen:

- alle voedingspinnen van de microcontroller zijn aangesloten op de 3,3V voeding. Deze zijn ontkoppeld zoals de datasheet weergeeft.
- De boot pin is aan grond aangesloten zodat de main flash als boot wordt gebruikt. Dit is weergegeven in de reference manual.
- De reset pin maakt gebruik van een drukknop. Aan deze pin wordt een pull up weerstand van 40k ohm en een ontkoppel condensator van 0,1µF aangesloten zoals weergegeven in de datasheet.
- De pinnen SWDIO en SWCLK worden gebruikt voor het programmeren en debuggen van de microcontroller. Dit doen we door gebruik te maken van een J-TAG connector met een niet officiële ST-Link.
- Aan de controller is een 8Mhz oscillator geconnecteerd (ECS2016MV080DNTR). Ik heb voor 8Mhz gekozen omdat alle voorbeelden en testcondities in de datasheet zijn uitgevoerd met 8Mhz. Het biedt een goede balans tussen een niet te hoge frequentie dat meer stroom gaat verbruiken en niet te dicht tegen de ondergrens van 4Mhz. Ik heb gekozen voor een oscillator en geen crystal omdat deze iets gemakkelijker zijn om aan te sluiten. De oscillator wordt aangesloten aan 3,3V met een 0,1µF ontkoppel condensator. De uitgang wordt aangesloten op de OSC_IN pin van de microcontroller. De oscillator bevat een tri-state pin die aangesloten is op 3,3V zodat het crystal aanstaat.

1.2.2. Debug:

Om tijdens het programmeren snel een aantal zaken kunnen uit te testen heb ik ervoor gekozen om naast de microcontroller 2 ledjes en 2 knoppen te plaatsen.
- De 2 ledjes (150060VS75000) hebben een voorweerstand van 64.9 ohm → R = U/I = (3,3V – 2V) / 20mA = 65 ohm. Ik heb gekozen voor 64.9 ohm omdat deze goedkoper was dan 65 ohm.
- De 2 drukknoppen zijn actief lagen. Hiervoor hebben ze pull up weerstand van 10k ohm. Ook worden ze ontkoppeld door 2 0.1µF condensatoren.

1.2.3. Voeding:

-Als hoofdvoeding van het systeem maak ik gebruik van een 3400mAH 12V loodaccu (UL12-12) aangezien deze uitstekend zijn om de terugkerende stroom van de motoren op te vangen.
De spanning van de voeding wordt in het oog gehouden door de microcontroller via een voltage divider → (1,5k ohm / (5,6k ohm + 1,5k ohm)) * 14V = 2.96V. Alhoewel het een 12V loodaccu is kan deze opladen tot 14V.

-Om het systeem te voeden met 3,3V wordt er gebruik gemaakt van een wisselende voedingsmodule (173950378). Rondom de module staan enkele randcomponenten zoals 2 filter condensatoren van 4,7µF, een filter spoel van 10µF en een bulk condensator van 100µF voor het opvangen van oscillaties.

- De voeding van de speakers en de voeding van de servos gebruiken dezelfde module (171030601). Deze module is een variabele wisselende regelaar die men kan instellen op een gewenst voltage. Ik maak

gebruik van 3 van deze modules aangezien deze 0.5A kunnen leveren. 1 voor de speaker en 2 voor de servomotoren.

1.2.4. DC-motoren en H-bruggen:

Om de rupsbanden van de tank aan te sturen heb ik gekozen voor DC motoren. Voor de DC motoren heb ik gekozen voor 2 motoren met een gearbox van pololu. Deze motoren werken op 12V en kunnen een stopstroom trekken tot 5,5A. Deze motoren bevatten een gearbox van 100:1 wat ervoor zorgt dat ze een koppel hebben van 34kg.cm wat meer dan genoeg zou moeten zijn voor dit project.

Voor de aansturing van deze motoren heb ik gekozen voor een H brug (TB67H303HG). Deze H brug zorgt ervoor dat de motoren de mogelijkheid hebben om naar beide kanten te kunnen draaien. De H-brug bevat 2 modi Diurect PWM en Constant-current PWM, ik maak gebruik van direct PWM.

1.2.4.1. Aansluitingen:

- De H brug wordt rechtstreeks gevoed door de accu. Deze wordt ontkoppeld volgens de datasheet met een 0.1µF ontkoppel condensator en een 47µF bulk condensator per voedingspin.
- Aan deze H brug zijn ook 2 gronden verbonden, namelijk signal ground en power ground. Men gebruikt aparte gronden zodat de grote stromen die getrokken worden door de motoren geen invloed heeft op de gevoelige digitale elektronica. Alhoewel deze apart verbonden zijn moeten de gronden wel aan elkaar gekoppeld zijn. Dit doen we via een net tie. Dit is een zeer kleine verbinding tussen beide gronden dat ervoor zorgt dat geen storingen doorkunnen.
-Vreg is de uitgang van de interne 5V regelaar die ontkoppeld wordt met een 0,1µF condensator. Aangezien ons systeem werkt op 3,3V maken we hier geen gebruik van.
- Vref, RSA en RSB worden niet gebruikt bij direct PWM en wordt aan grond aangesloten.
- Op de OSC pin wordt een 51k ohm weerstand aangesloten volgens de datasheet die ervoor zorgt dat intern een driehoekgolf wordt gevormd. Deze wordt gebruikt voor de over current detection en thermal shut down circuit.
- De pinnen ALERT 1 en ALERT 2 worden gebruikt om de microcontroller te waarschuwen van undervoltage, over current en thermal shutdown. Deze zijn actief laag en hebben een pull up weerstand van 20k volgens de datasheet. Alhoewel deze pinnen aangesloten zijn op de microcontroller worden ze niet gebruikt vanwege:
    - de UVLO geeft enkel een signaal als het voltage valt onder 6V. Wanneer we dit voltage bereiken is de batterij al lang dood.
    - de ISD geeft enkel een signaal als de stroom groter wordt dan 6,5A. Onze motoren kunnen max 5,5A trekken waardoor dit signaal nooit zal opgeroepen worden.
    - De TSH geeft een signaal als de temperatuur hoger wordt dan 160 graden Celsius. Aangezien we gebruik maken van redelijk grote koelvinnen en empirisch hebben bepaald dat deze niet warm worden, is het niet nodig om dit te gebruiken.
- De select pin is aangesloten aan grond zodat direct pwm mode wordt gekozen.
- Standby word aangesloten op 3,3V want we willen niet dat de motoren uitvallen.
- De stand van pin IN1 en IN2 bepalen de richting van de motor en zijn aangesloten op de microcontroller
- De PWM pin bepaalt de snelheid van de motor en is aangesloten op de microcontroller.
- De uitgang van de H-brug wordt aangesloten aan een 2 pin schroef aansluiting.

1.2.5. Servo motoren:

Ik maak gebruik van 4 after market SG90 servomotoren vanwege hun compacte behuizing. Deze zorgen voor de aansturing van de kanonnen en de kanonnentoren’s.

De servo’s worden gevoed door de 6V regelaar en aangedreven door een N type mosfet die een PWM van de microcontroller ontvangt van 3,3V en deze upped naar de benodigde 6V. Tussen de microcontroller en de Mosfet is een 470 ohm weerstand bevestigd die de microcontroller beschermt. De datalijn van de servo wordt aangesloten aan 6V via een pull up weerstand van 10k zodat deze stabieler blijft.

1.2.6. Speakers:

Per kanon maak ik gebruik van een speaker (1890963). Deze speakers werken op basis van een PWM signaal van de microcontroller. Aangezien de microcontroller niet genoeg stroom kan voorzien en omdat 3,3V teveel ruis veroorzaakt maken ze gebruik van een 2,5V regelaar en een N type mosfet voor de aansturing van de speakers.

1.2.7. RF module:

De RF module (NRF24L01+) wordt gebruikt voor de communicatie met de controller. Deze wordt gevoed door de 3,3V regelaar en is aangesloten aan de SPI pinnen van de microcontroller waarop length matching is toegepast.

1.2.8. SD-kaart:

De SD-kaart is bijgeplaatst voor het opslaan van grotere soundfiles. Deze wordt gevoed door de 3,3V regelaar en is aangesloten aan de SPI pinnen van de microcontroller waarop length matching is toegepast. De SD kaart word momenteel niet gebruikt vanwege de slechte kwaliteiten van het afspelen van soundfiles via PWM.

1.2.9. Header:

Voor eventuele uitbreidingen naar later toe zijn er 4 headers aan de rand van de PCB geplaats. 2 hiervan zijn voor een 3,3V voeding en 2 voor UART.

Tijdens het ontwerpen van de PCB heb ik ervoor gezorgd dat alles aan 1 kant van de PCB stond zodat ik gebruik kan maken van de T962 infrared oven die ik vorig jaar heb aangekocht. Hiervoor heb ik dus ook een stencil gekocht bij de printplaten.

Problemen en oplossingen

Probleem: De servomotoren hebben het probleem dat ze veel last hebben van jitter. Dit komt deels doordat de ADC niet 100% stabiel is, alhoewel deze niet veel varieert. Maar ook door de servomotoren zelf aangezien deze zeer goedkoop waren en van een no-name brand. Ook kan de servomotor bewegen als een andere servomotor bewogen wordt, wat niet de bedoeling is.

Oplossing: Voor dit probleem zoveel mogelijk op te lossen maak ik gebruik van een rollend gemiddelde. Ook zorg ik ervoor dat de PWM enkel aanstaat als er een verandering in positie moet gebeuren van een bepaalde grootte. Dit heeft het probleem voor een groot deel opgelost, alhoewel het zeker nog niet perfect is.

Probleem: De DC motoren hadden last van de ruis op de ADC van de joystick. De ADC waarde ging geregeld 1000 eenheden omhoog en omlaag waardoor de motor kort naar de andere kant wilde rijden en dus schokkend verder ging.

Oplossing: dit probleem is verholpen door ervoor te zorgen dat er minstens 3 achtereenvolgende waardes over een bepaalde grens moeten gaan vooraleer de motor met deze waardes rekening houdt.

Probleem: Tijdens het ontwerpen van de printplaat heb ik per ongeluk de 2 ledjes getekend in 0603mm in plaats van 0603inch. Dit heeft ervoor gezorgd dat de ledjes niet gesoldeerd konden worden.

Oplossing: Helaas is hier geen oplossing voor en moet ik de printplaat debuggen zonder ledjes. Dit is zeker geen groot probleem want de controller bevat ledjes die ik tijdens het debuggen kon aansturen met de PCB van het tankje.

Probleem: Tijdens het solderen heb ik gebruik gemaakt van een kleine T-962 infrarood oven. Deze heeft redelijk goed gewerkt maar mijn soldeerpaste was recentelijk vervallen. Hierdoor had de microcontroller geen 48 pinnen meer maar 4 grote pinnen. De 4 zijdes waren volledig aan elkaar vast gesoldeerd.

Oplossing: Manueel de microcontroller gefixed.

Probleem: De kanonnentorens zouden moeten kunnen draaien aan de hand van servomotoren. De torens zijn iets te nipt ontworpen en door de grote toleranties met 3d printen schuren de onderdelen een beetje waardoor de servo’s het niet altijd gedraaid krijgen.

Oplossing: Momenteel heb ik nog geen tijd voor gehad maar een oplossing zou zijn om de omliggende onderdelen wat te schuren zodat de toren kan draaien.

Probleem: Origineel was er voorzien om de rupsbanden aan te drijven met tandwielen. Vanwege het vreemde formaat heb ik onervaren een rol tandriem gekocht dat met losse uiteindes kwam zodat ik zelf een riem op formaat kon maken. Na de riem aan elkaar te proberen krijgen met superlijm, ductape, een stuk stof en zelfs te naaien, en dat allemaal samen zonder resultaat, heb ik de riem opgegeven.

Oplossing: In plaats van gebruik te maken van een riem maak ik nu gebruik een linaire gearbox die 10 tandwielen naast elkaar heeft staan van aan de motor tot aan het aandrijvingstandwiel.

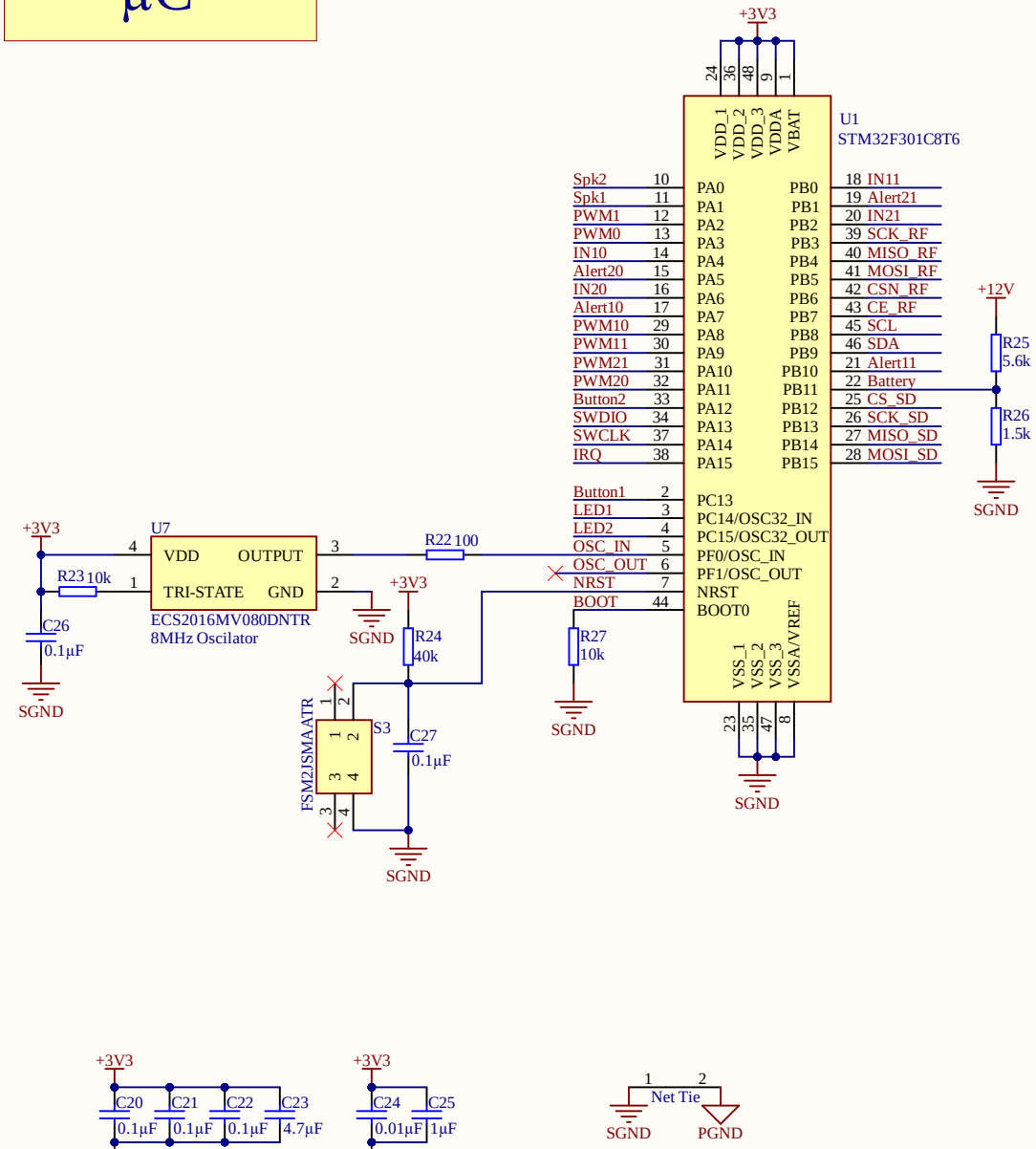
Nieuw probleem: De motor heeft het tandwiel op zijn as uitgefreesd en draait nu los in het tandwiel.

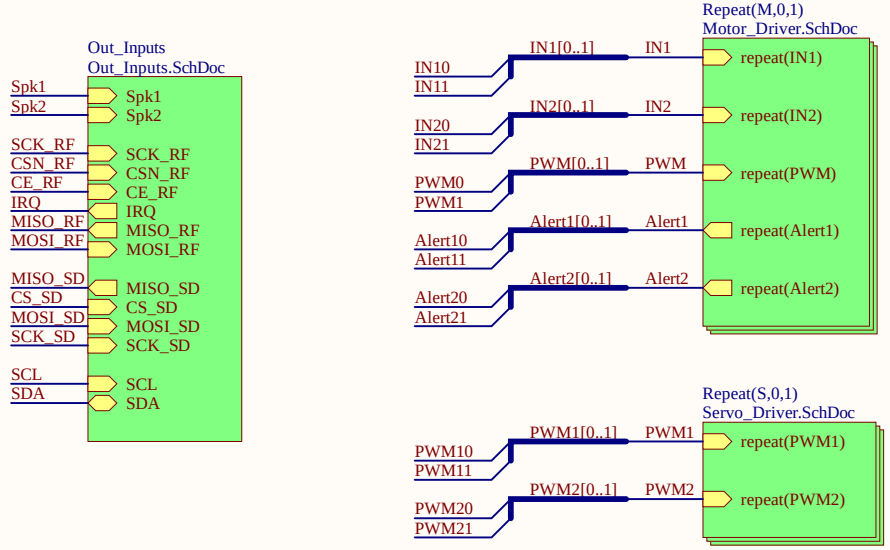
Oplossing: Het tandwiel aan de motor vastvijzen. Dit is nog niet gebeurd.

Conclusie

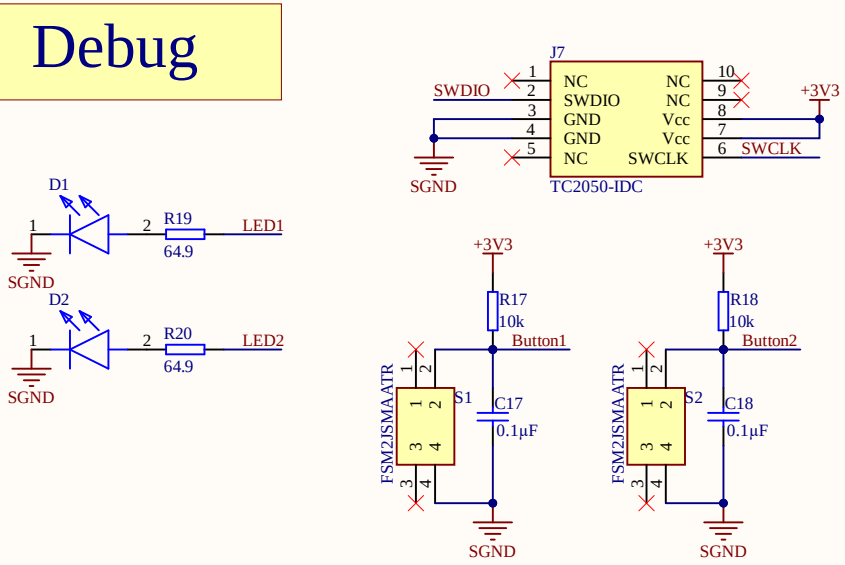
Alhoewel dat momenteel de tank nog niet rijdt ben ik toch tevreden met waar ik ben geraakt. Door dit project heb ik veel beter leren werken met fusion360 en altium en ook heb ik veel beter leren solderen. Door in de toekomst nog wat aanpassingen hier en daar te doen aan het voertuig zal het zeker en vast kunnen rijden.

µC

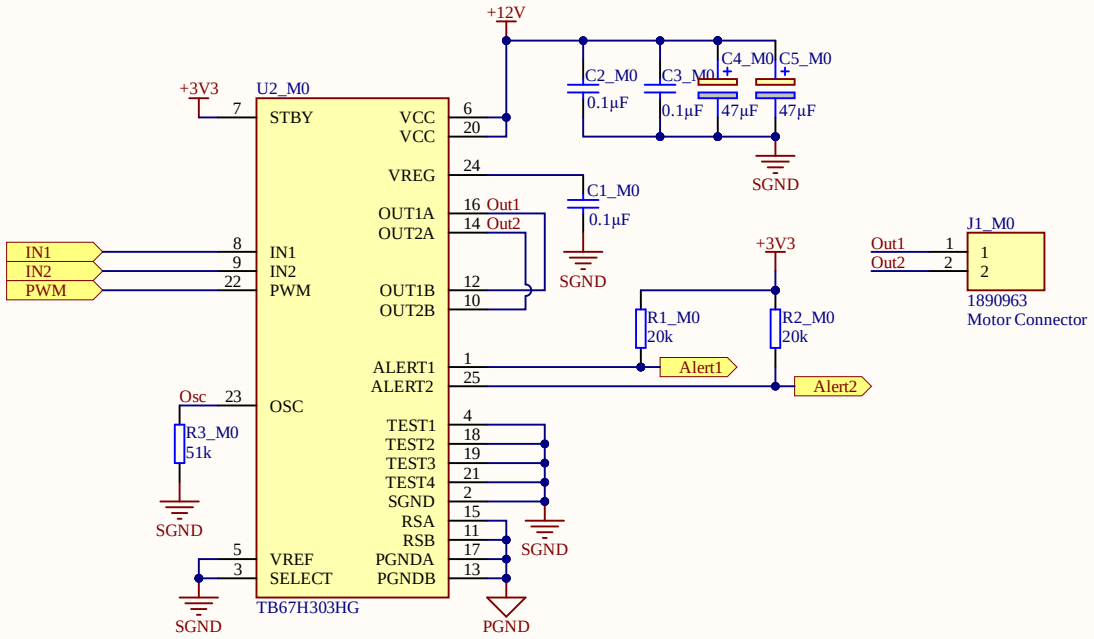




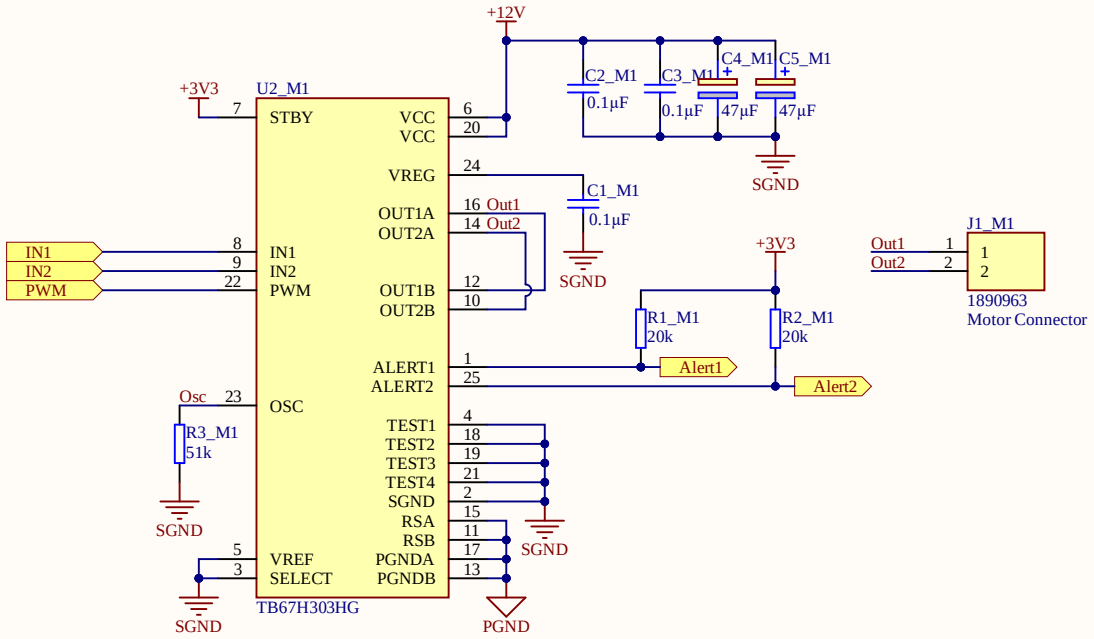
Debug



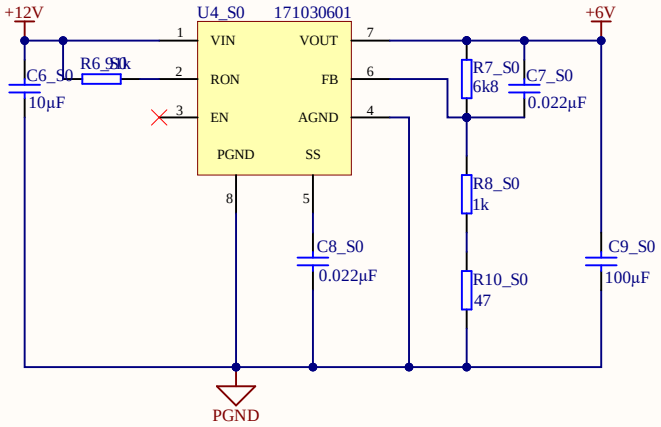
Title		
Size A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Main_Board_Tank.SchDoc		Drawn By:

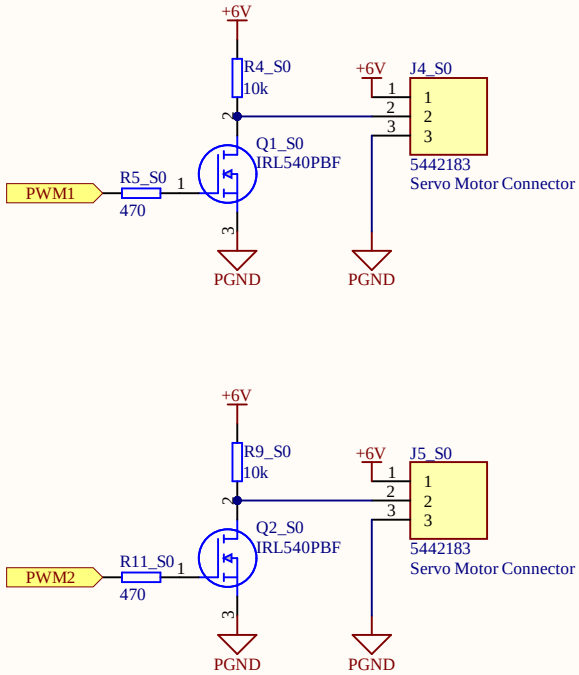


Title		
Size: A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Motor_Driver.SchDoc		Drawn By:

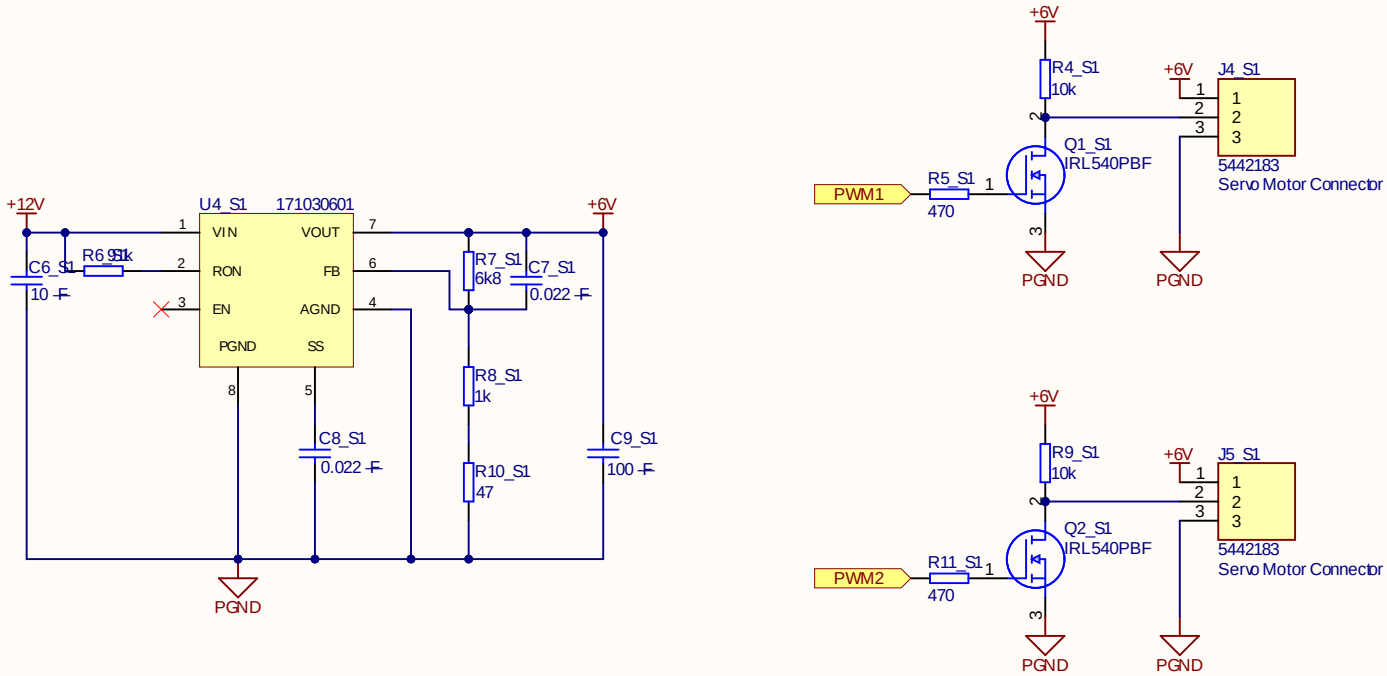


Title		
Size: A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Motor_Driver.SchDoc		Drawn By:



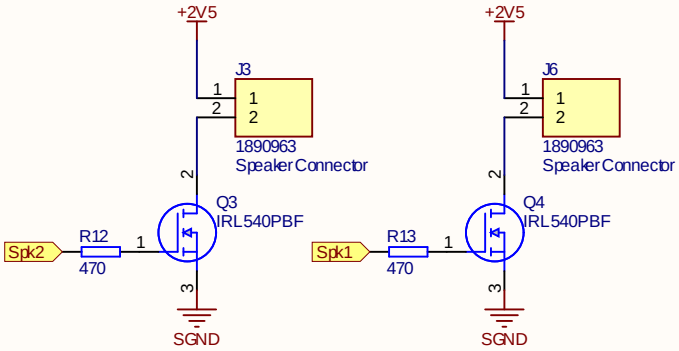


Title		
Size: A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Servo_Driver.SchDoc		Drawn By:

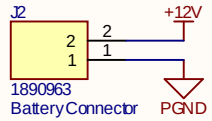


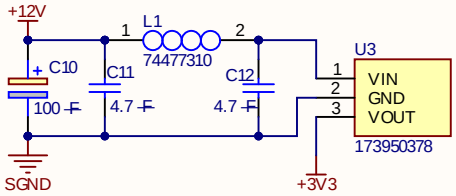
Title		
Size A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Servo_Driver.SchDoc		Drawn By:

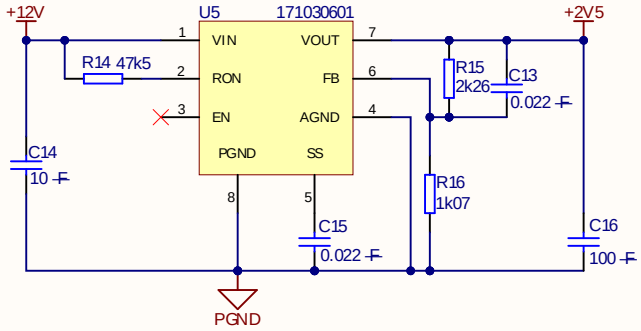
Speakers



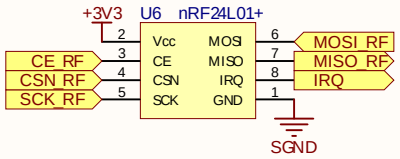
Voeding



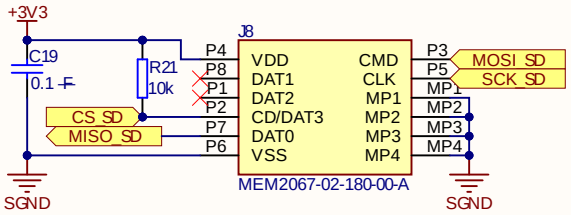




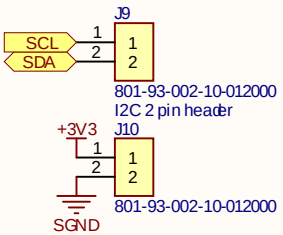
RF



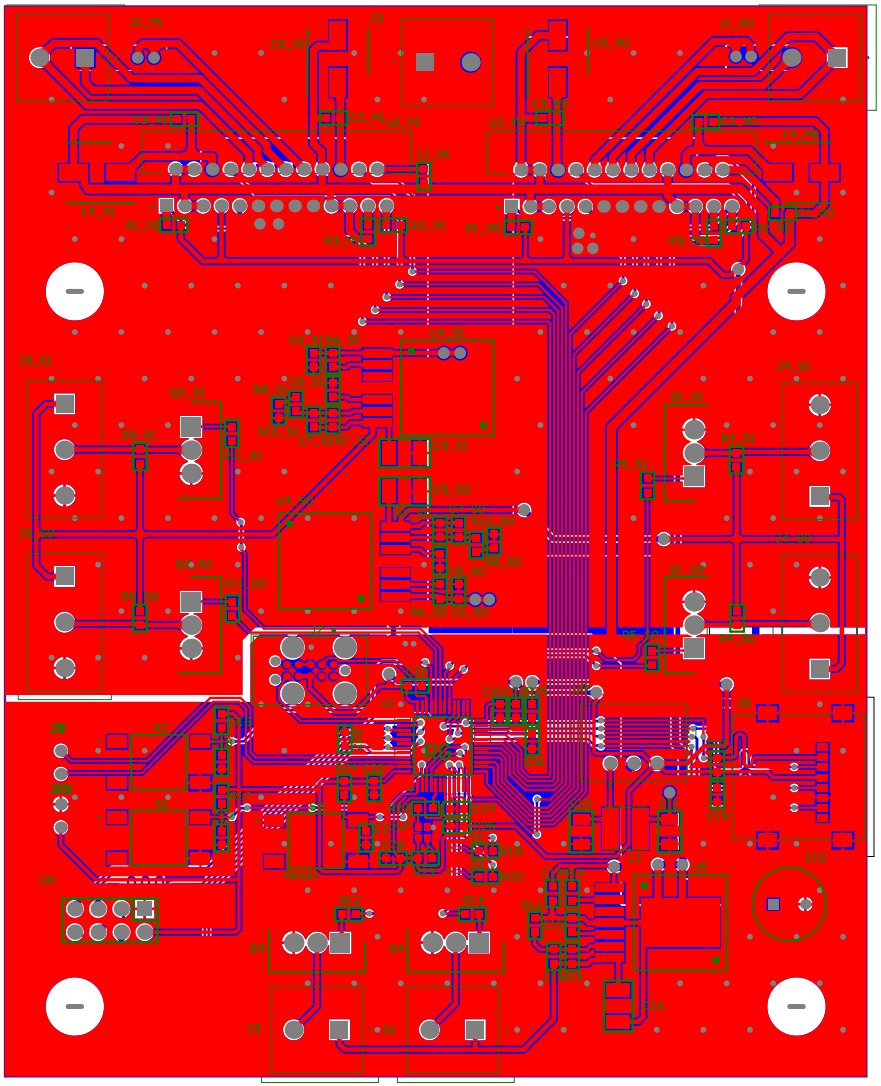
SD Card

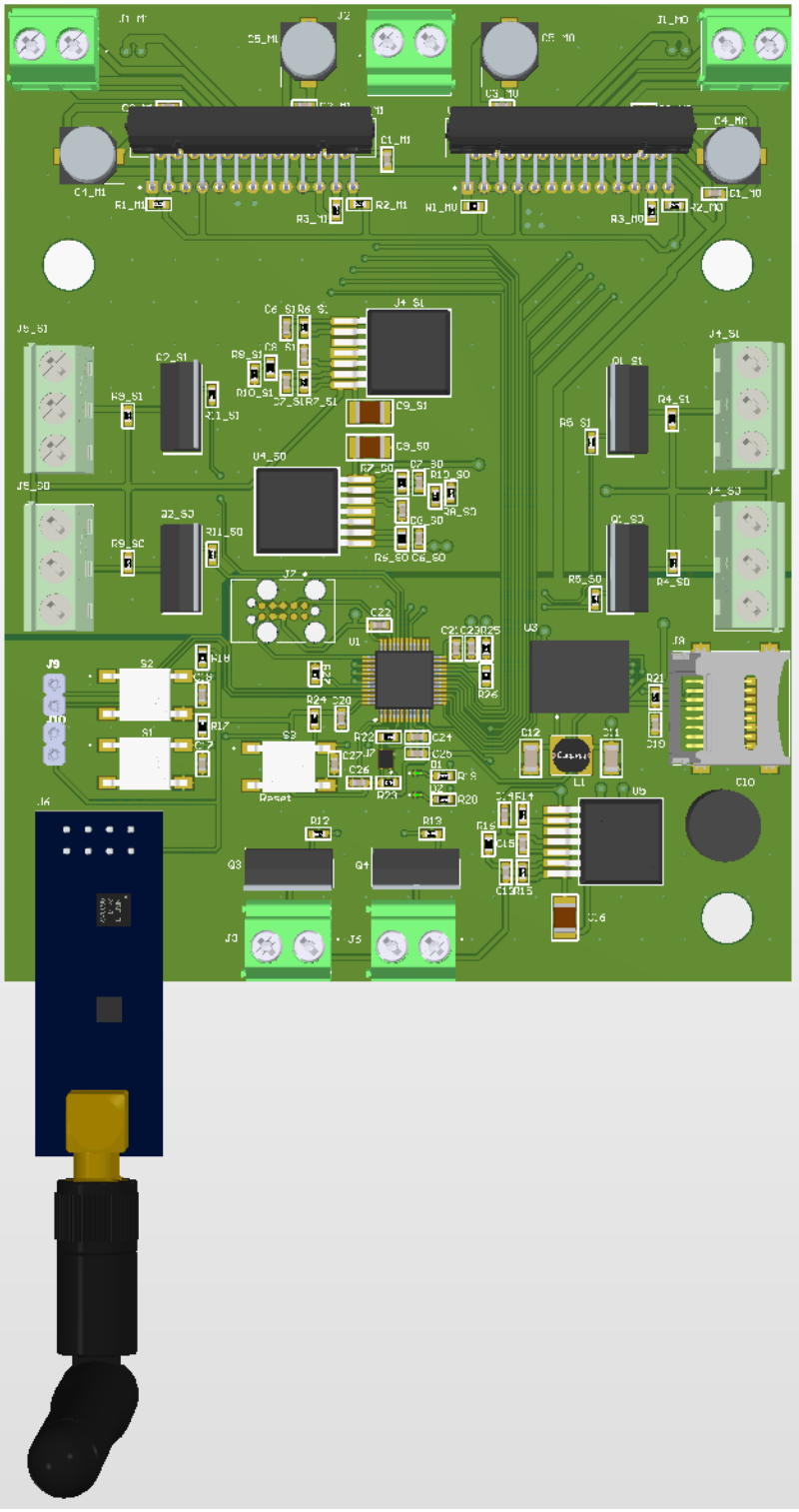


I2C



Title		
Size A4	Number	Revision
Date: 6/02/2024		Sheet of
File: C:\Users\..\Out_Inputs.SchDoc		Drawn By:





```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "NRF24L01.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DEAD_ZONE_LOW 1800
#define DEAD_ZONE_HIGH 2400
#define NOJITTER 3
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 ADC_HandleTypeDef hadc1;

SPI_HandleTypeDef hspi3;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim15;

/* USER CODE BEGIN PV */
 uint8_t RxAddress[] = {0xEE,0xDD,0xCC,0xBB,0xAA}; //Adres waar data naar word gesschreven. Zowel
```

```
Rx als Tx adress moeten overeen komen bij het zenden en ontvangen van data.
 uint8_t RxData[19]; //Ontvangen data buffer. 18 bytes voor de doorgestuurde ADC's en 1 byte voor de status van de knoppen.
 uint32_t ADC_Values[9]; // Array voor het opslaan van de geconverteerde bytes naar 12bit
 uint8_t buttonStates; // Integer voor het opslaan van de status van de knoppen

 uint32_t dutyCycle_G1 = 0; //dutycycle servo kanon 1
 uint32_t index_G1 = 0; //index voor het rollend gemiddelde
 uint32_t dutyCycle_G1_total = 0; //telt de dutycycle op voor een gemiddelde
 uint32_t dutyCycle_G1_count = 0; //houdt bij hoeveel keer al is opgetelt
 uint32_t values_G1[10] = {0}; // Array voor het opslaan van de weaardes voor de berekening van het gemidelde
 uint32_t av_du_G1 = 0; // gemidelde duty cycle
 uint32_t Last_av_du_G1 = 0; // vorige gemidelde duty cycle

 uint32_t dutyCycle_G2 = 0; //dutycycle servo kanon 2
 uint32_t index_G2 = 0; //index voor het rollend gemiddelde
 uint32_t dutyCycle_G2_total = 0; //telt de dutycycle op voor een gemiddelde
 uint32_t dutyCycle_G2_count = 0; //houdt bij hoeveel keer al is opgetelt
 uint32_t values_G2[10] = {0}; // Array voor het opslaan van de weaardes voor de berekening van het gemidelde
 uint32_t av_du_G2 = 0; // gemidelde duty cycle
 uint32_t Last_av_du_G2 = 0; // vorige gemidelde duty cycle


 uint32_t dutyCycle_T1 = 0; //dutycycle servo kanontoren 1
 uint32_t index_T1 = 0; //index voor het rollend gemiddelde
 uint32_t dutyCycle_T1_total = 0; //telt de dutycycle op voor een gemiddelde
 uint32_t dutyCycle_T1_count = 0; //houdt bij hoeveel keer al is opgetelt
 uint32_t values_T1[10] = {0}; // Array voor het opslaan van de weaardes voor de berekening van het gemidelde
 uint32_t av_du_T1 = 0; // gemidelde duty cycle
 uint32_t Last_av_du_T1 = 0; // vorige gemidelde duty cycle

 uint32_t dutyCycle_T2 = 0; //dutycycle servo kanontoren 2
 uint32_t index_T2 = 0; //index voor het rollend gemiddelde
 uint32_t dutyCycle_T2_total = 0; //telt de dutycycle op voor een gemiddelde
 uint32_t dutyCycle_T2_count = 0; //houdt bij hoeveel keer al is opgetelt
 uint32_t values_T2[10] = {0}; // Array voor het opslaan van de weaardes voor de berekening van het gemidelde
 uint32_t av_du_T2 = 0; // gemidelde duty cycle
 uint32_t Last_av_du_T2 = 0; // vorige gemidelde duty cycle

 uint8_t dutyCycle_CH3 = 0;  // Duty cycle procent voor TIM2 Channel 3
 uint8_t dutyCycle_CH4 = 0;  // Duty cycle in procent voor TIM2 Channel 4
 int lowCountM1 = 0; //Teller om na te gaan of de waarde echt laag is of het gewoon jitter is in motor 1.
 int highCountM1 = 0; //Teller om na te gaan of de waarde echt hoog is of het gewoon jitter is in motor 1.
 int lowCountM2 = 0; //Teller om na te gaan of de waarde echt laag is of het gewoon jitter is in motor 2.
 int highCountM2 = 0; //Teller om na te gaan of de waarde echt hoog is of het gewoon jitter is in motor 2.

 uint32_t ADC; //ADC voor het niveau van de batterij

 uint32_t current_Tick;
 uint32_t prev_Tick;
/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI3_Init(void);
static void MX_TIM1_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM15_Init(void);
/* USER CODE BEGIN PFP */
void processData(void);
void adjustDutyCycle(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////// processData combineert alle ontvangen bytes terug naar hun originele 12 bit staat en onthoudt de status van de knoppen
//////////////////// bv. RxData[j] = lower 8 bits = 0100 1100 en RxData[j+1] = upper 4 bits = 0000 1011
////////////////////  RxData[j+1] << 8 = 0000 1011 0000 0000
////////////////////     RxData[j] | RxData[j+1] =          0100 1100
////////////////////                            OR 0000 1011 0000 0000
////////////////////                             = 0000 1011 0100 1100 = 12 bit waarde
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void processData(void){
 NRF24_Receive(RxData); // Ontvang data
 for (int i = 0, j = 0; i < 9; i++, j += 2) {
  ADC_Values[i] = (RxData[j] | (RxData[j+1] << 8));  // Combineert 2 ontvangen data bytes naar een 12 bit value.
 }
 buttonStates = RxData[18]; // Haalt de status van de knoppen uit de ontvangen data
 adjustDutyCycle(); // Roept de functie op voor het toepassen van de nieuwe waardes
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////// adjustDutyCycle gebruikt de ontvangen waardes om de duyclyes van de motoren aan te passen
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void adjustDutyCycle(void) {
 ////////////////////////////////////////////////////
 //////////////////// DC Motor 1 ////////////////////////
 ////////////////////////////////////////////////////
 if (ADC_Values[0] < 4096){ // zorgt ervoor dat foutieve waardes boven 4096 genegeert worden
  if (ADC_Values[0] < DEAD_ZONE_LOW) { // Als de ADC waarde onder de threshold ligt gaat de motor achteruit.
   lowCountM1 += 1;
```

```
   highCountM1 = 0;
   if(lowCountM1 >= NOJITTER) // Als de waarde 3 keer onder de threshold ligt dan weten we dat het
geen jitter is.
   {
    lowCountM1 = 0;
    dutyCycle_CH3 = (DEAD_ZONE_LOW - ADC_Values[0]) * 100 / (DEAD_ZONE_LOW - 500); //
berekent de dutycycle voor een waarde onder de threshold
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2,0); //Configureert IN21
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0,1); //Configureert IN11
   }
  }
  else if (ADC_Values[0] > DEAD_ZONE_HIGH) { // Als de ADC waarde boven de threshold ligt gaat de
motor vooruit.
   highCountM1 += 1;
   lowCountM1 = 0;
   if(highCountM1 >= NOJITTER) // Als de waarde 3 keer onder de threshold ligt dan weten we dat het
geen jitter is.
   {
    highCountM1 = 0;
    dutyCycle_CH3 = (ADC_Values[0] - DEAD_ZONE_HIGH) * 100 / (3550 - DEAD_ZONE_HIGH); //
berekent de dutycycle voor een waarde boven de threshold
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0,0); //Configureert IN11
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2,1); //Configureert IN21
   }
  }
  else {
   highCountM1 = 0;
   lowCountM1 = 0;
   dutyCycle_CH3 = 0; // Motor stops in dead zone
  }
  __HAL_TIM_SET_COMPARE(&htim15, TIM_CHANNEL_1, (2999 * dutyCycle_CH3) / 100); //Gebruikt
de nieuwe duty cycle en laat de motor draaien.
 }
 ////////////////////////////////////////////////////
 ////////////////////////////////////////////////////

 ////////////////////////////////////////////////////
 ////////////////// DC Motor 2 //////////////////////
 ////////////////////////////////////////////////////
 if (ADC_Values[7] < 4096){ // zorgt ervoor dat foutieve waardes boven 4096 genegeert worden
  if (ADC_Values[7] < DEAD_ZONE_LOW) { // Als de ADC waarde onder de threshold ligt gaat de motor
achteruit.
   lowCountM2 += 1;
   highCountM2 = 0;
   if(lowCountM2 >= NOJITTER) // Als de waarde 3 keer onder de threshold ligt dan weten we dat het
geen jitter is.
   {
    lowCountM2 = 0;
    dutyCycle_CH4 = (DEAD_ZONE_LOW - ADC_Values[7]) * 100 / (DEAD_ZONE_LOW - 500); //
berekent de dutycycle voor een waarde onder de threshold
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4,0); //Configureert IN10
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6,1); //Configureert IN20
   }
  }
  else if (ADC_Values[7] > DEAD_ZONE_HIGH) { // Als de ADC waarde boven de threshold ligt gaat de
```

```
motor vooruit.
   highCountM2 += 1;
   lowCountM2 = 0;
   if(highCountM2 >= NOJITTER) // Als de waarde 3 keer onder de threshold ligt dan weten we dat het
geen jitter is.
   {
    highCountM2 = 0;
    dutyCycle_CH4 = (ADC_Values[7] - DEAD_ZONE_HIGH) * 100 / (3550 - DEAD_ZONE_HIGH); //
berekent de dutycycle voor een waarde boven de threshold
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6,0); //Configureert IN20
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4,1); //Configureert IN10
   }
  }
  else {
   highCountM2 = 0;
   lowCountM2 = 0;
   dutyCycle_CH4 = 0; // Motor stops in dead zone
  }
  __HAL_TIM_SET_COMPARE(&htim15, TIM_CHANNEL_2, (2999 * dutyCycle_CH4) / 100); //Gebruikt
de nieuwe duty cycle en laat de motor draaien.
 }
 //////////////////////////////////////////////////////
 //////////////////////////////////////////////////////

 //////////////////////////////////////////////////////
 ////////////////////// Kanon servo 1 ////////////////////
 //////////////////////////////////////////////////////
 dutyCycle_G1 = 3600 + ((ADC_Values[5] * (5200 - 3600)) / 4095); // the duty cycly = min value * % *
(Max vcalue - Min Value) (20ms = 60khz)
 if (dutyCycle_G1 < 3600) // De waarden limiteren tot hun max/minimum
 {
  dutyCycle_G1 = 3600;
 }
 if (dutyCycle_G1 > 5200)
 {
  dutyCycle_G1 = 5200;
 }
 dutyCycle_G1_total = dutyCycle_G1_total - values_G1[index_G1] + dutyCycle_G1;
 values_G1[index_G1] = dutyCycle_G1;
 index_G1 = (index_G1 + 1) % 10;
 if (dutyCycle_G1_count < 10) { // Het rollend gemiddelde moet eerst tot 10 waarden stijgen nadien blijft
deze hier.
  dutyCycle_G1_count++;
 }
 av_du_G1 = dutyCycle_G1_total / dutyCycle_G1_count; // Bereken de gemiddelde duty cycle
 if (av_du_G1 < 3600) // De waarden limiteren tot hun max/minimum
 {
  av_du_G1 = 3600;
 }
 if (av_du_G1 > 5200)
 {
  av_du_G1 = 5200;
 }
 if (av_du_G1 > Last_av_du_G1)
 {
```

```
  if ((av_du_G1 - Last_av_du_G1) > 150) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_G1 = av_du_G1;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3,av_du_G1); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
   HAL_Delay(20);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
  }
 }
 if (av_du_G1 < Last_av_du_G1)
 {
  if ((Last_av_du_G1 - av_du_G1) > 150) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_G1 = av_du_G1;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3,av_du_G1); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
   HAL_Delay(40);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
  }
 }
 ////////////////////////////////////////////////////
 ////////////////////////////////////////////////////

 ////////////////////////////////////////////////////
 ////////////////// Kanon servo 2 //////////////////
 ////////////////////////////////////////////////////
 dutyCycle_G2 = 3600 + ((ADC_Values[2] * (5200 - 3600)) / 4095); // the duty cycly = min value * % * (Max vcalue - Min Value) (20ms = 60khz)
 if (dutyCycle_G2 < 3600) // De waarden limiteren tot hun max/minimum
 {
  dutyCycle_G2 = 3600;
 }
 if (dutyCycle_G2 > 5200)
 {
  dutyCycle_G2 = 5200;
 }
 dutyCycle_G2_total = dutyCycle_G2_total - values_G2[index_G2] + dutyCycle_G2; // de totaalsom van het rollent gemiddelde = De som - de oudste waar + de nieuwste waarde
 values_G2[index_G2] = dutyCycle_G2;
 index_G2 = (index_G2 + 1) % 10;
 if (dutyCycle_G2_count < 10) { // Het rollend gemiddelde moet eerst tot 10 waarden stijgen nadien blijft deze hier.
  dutyCycle_G2_count++;
 }
 av_du_G2 = dutyCycle_G2_total / dutyCycle_G2_count; // Bereken de gemiddelde duty cycle
```

```
 if (av_du_G2 < 3600) // De waarden limiteren tot hun max/minimum
 {
  av_du_G2 = 3600;
 }
 if (av_du_G2 > 5200)
 {
  av_du_G2 = 5200;
 }
 if (av_du_G2 > Last_av_du_G2)
 {
  if ((av_du_G2 - Last_av_du_G2) > 150) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_G2 = av_du_G2;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,av_du_G2); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
   HAL_Delay(20);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
  }
 }
 if (av_du_G2 < Last_av_du_G2)
 {
  if ((Last_av_du_G2 - av_du_G2) > 150) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_G2 = av_du_G2;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,av_du_G2); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
   HAL_Delay(40);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
  }
 }
 ////////////////////////////////////////////////////
 ////////////////////////////////////////////////////

 ////////////////////////////////////////////////////
 ////////////////// Kanonentoren servo 1 ////////////
 ////////////////////////////////////////////////////

 dutyCycle_T1 = 3000 + ((ADC_Values[4] * (6000 - 3000)) / 4095); // the duty cycly = min value * % * (Max vcalue - Min Value) (20ms = 60khz)
 if (dutyCycle_T1 < 3000) // De waarden limiteren tot hun max/minimum
 {
  dutyCycle_T1 = 3000;
 }
 if (dutyCycle_T1 > 6000)
 {
```

```
  dutyCycle_T1 = 6000;
 }
 dutyCycle_T1_total = dutyCycle_T1_total - values_T1[index_T1] + dutyCycle_T1; // de totaalsom van het rollent gemiddelde = De som - de oudste waar + de nieuwste waarde
 values_T1[index_T1] = dutyCycle_T1;
 index_T1 = (index_T1 + 1) % 10;
 if (dutyCycle_T1_count < 10) { // Het rollend gemiddelde moet eerst tot 10 waarden stijgen nadien blijft deze hier.
  dutyCycle_T1_count++;
 }
 av_du_T1 = dutyCycle_T1_total / dutyCycle_T1_count; // Bereken de gemiddelde duty cycle
 if (av_du_T1 < 3000) // De waarden limiteren tot hun max/minimum
 {
  av_du_T1 = 3000;
 }
 if (av_du_T1 > 6000)
 {
  av_du_T1 = 6000;
 }
 if (av_du_T1 > Last_av_du_T1)
 {
  if ((av_du_T1 - Last_av_du_T1) > 500) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_T1 = av_du_T1;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4,av_du_T1); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
   HAL_Delay(20);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
  }
 }
 if (av_du_T1 < Last_av_du_T1)
 {
  if ((Last_av_du_T1 - av_du_T1) > 500) // er mag pas bewogen worden van zodra er een verschil is van meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_T1 = av_du_T1;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4,av_du_T1); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
   HAL_Delay(40);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
  }
 }
 ////////////////////////////////////////////////////
 ////////////////////////////////////////////////////

 ////////////////////////////////////////////////////
 ////////////////// Kanonentoren servo 2 ////////////
 ////////////////////////////////////////////////////
```

```
 dutyCycle_T2 = 3000 + ((ADC_Values[3] * (6000 - 3000)) / 4095); // the duty cycly = min value * % *
(Max vcalue - Min Value) (20ms = 60khz)
 if (dutyCycle_T2 < 3000) // De waarden limiteren tot hun max/minimum
 {
  dutyCycle_T2 = 3000;
 }
 if (dutyCycle_T2 > 6000)
 {
  dutyCycle_T2 = 6000;
 }
 dutyCycle_T2_total = dutyCycle_T2_total - values_T2[index_T2] + dutyCycle_T2; // de totaalsom van het
rollent gemiddelde = De som - de oudste waar + de nieuwste waarde
 values_T2[index_T2] = dutyCycle_T2;
 index_T2 = (index_T2 + 1) % 10;
 if (dutyCycle_T2_count < 10) { // Het rollend gemiddelde moet eerst tot 10 waarden stijgen nadien blijft
deze hier.
  dutyCycle_T2_count++;
 }
 av_du_T2 = dutyCycle_T2_total / dutyCycle_T2_count; // Bereken de gemiddelde duty cycle
 if (av_du_T2 < 3000) // De waarden limiteren tot hun max/minimum
 {
  av_du_T2 = 3000;
 }
 if (av_du_T2 > 6000)
 {
  av_du_T2 = 6000;
 }
 if (av_du_T2 > Last_av_du_T2)
 {
  if ((av_du_T2 - Last_av_du_T2) > 150) // er mag pas bewogen worden van zodra er een verschil is van
meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_T2 = av_du_T2;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,av_du_T2); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
   HAL_Delay(20);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
  }
 }
 if (av_du_T2 < Last_av_du_T2)
 {
  if ((Last_av_du_T2 - av_du_T2) > 150) // er mag pas bewogen worden van zodra er een verschil is van
meer dan 100 eenheiden tegen over vorige waarde.
  {
   Last_av_du_T2 = av_du_T2;
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,av_du_T2); // gebruik de gevonde duty cycle
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
   HAL_Delay(40);
```

```
   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
  }
 }
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI3_Init();
  MX_TIM1_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_TIM15_Init();
  /* USER CODE BEGIN 2 */

  NRF24_Init(); // Initialiseert de NRF24L01+ module.
  NRF24_RXMode(RxAddress, 10); // Configureert de NRF24L01+ module in Receive mode.
  HAL_TIM_Base_Start_IT(&htim1); //Start timer 1 gebruikt voor de servo motoren.
  HAL_TIM_Base_Start_IT(&htim15); //Start timer 2 gebruikt voor de DC motoren.
  HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1); //Start de PWM voor DC motor 1.
  HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_2);  //Start de PWM voor DC motor 2.
  __HAL_TIM_SET_COMPARE(&htim15, TIM_CHANNEL_1, 0); // Initialisert de counter op 0 van DC
motor 1 zodat deze niet automatisch start met rijden.
  __HAL_TIM_SET_COMPARE(&htim15, TIM_CHANNEL_2, 0); // Initialisert de counter op 0 van DC
motor 2 zodat deze niet automatisch start met rijden.
  HAL_ADC_Start(&hadc1); // Start de ADC voor het batterij niveau.
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
   current_Tick = HAL_GetTick();
   if (HAL_ADC_PollForConversion(&hadc1, 1000000) == HAL_OK)
   {
    ADC = HAL_ADC_GetValue(&hadc1);
   }
   if(ADC > 3020) //kijkt na of het batterij niveau nog boven 11.8V is.
   {
    if (isDataAvailable(1) == 1) // Kijkt na of er data beschikbaar is.
    {
     prev_Tick = HAL_GetTick();
     processData(); //Als data beschikbaar is gaan we deze data verwerken.
    }
   }
   if (prev_Tick + 500 < current_Tick)
   {
    prev_Tick = HAL_GetTick();
    HAL_TIM_PWM_Stop(&htim15, TIM_CHANNEL_1);
    HAL_TIM_PWM_Stop(&htim15, TIM_CHANNEL_2);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
   }
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV2;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL15;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_TIM1|RCC_PERIPHCLK_TIM15;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  PeriphClkInit.Tim15ClockSelection = RCC_TIM15CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_14;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.SamplingTime = ADC_SAMPLETIME_19CYCLES_5;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief SPI3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI3_Init(void)
{

  /* USER CODE BEGIN SPI3_Init 0 */

  /* USER CODE END SPI3_Init 0 */

  /* USER CODE BEGIN SPI3_Init 1 */

  /* USER CODE END SPI3_Init 1 */
  /* SPI3 parameter configuration*/
  hspi3.Instance = SPI3;
  hspi3.Init.Mode = SPI_MODE_MASTER;
  hspi3.Init.Direction = SPI_DIRECTION_2LINES;
  hspi3.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi3.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi3.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
```

```
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI3_Init 2 */

  /* USER CODE END SPI3_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 19;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 59999;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
```

```
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_LOW;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_ENABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
  sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
  sBreakDeadTimeConfig.Break2Filter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 599;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_LOW;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
```

```
  * @brief TIM15 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM15_Init(void)
{

  /* USER CODE BEGIN TIM15_Init 0 */

  /* USER CODE END TIM15_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM15_Init 1 */

  /* USER CODE END TIM15_Init 1 */
  htim15.Instance = TIM15;
  htim15.Init.Prescaler = 0;
  htim15.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim15.Init.Period = 2999;
  htim15.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim15.Init.RepetitionCounter = 0;
  htim15.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim15) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim15, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim15) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim15, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_ENABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim15, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim15, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim15, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM15_Init 2 */

  /* USER CODE END TIM15_Init 2 */
  HAL_TIM_MspPostInit(&htim15);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, IN10_Pin|IN20_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, IN11_Pin|IN21_Pin|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : IN10_Pin IN20_Pin */
  GPIO_InitStruct.Pin = IN10_Pin|IN20_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : IN11_Pin IN21_Pin PB6 PB7 */
  GPIO_InitStruct.Pin = IN11_Pin|IN21_Pin|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PA12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
```

```
}
#endif /* USE_FULL_ASSERT */
```

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "NRF24L01.h"
#include <stdint.h>
#include <string.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

SPI_HandleTypeDef hspi3;

TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart3;

/* USER CODE BEGIN PV */
 uint8_t TxAddress[] = {0xEE,0xDD,0xCC,0xBB,0xAA}; //Adres waar data naar word geschreven. Zowel Rx als Tx adress moeten overeen komen bij het zenden en ontvangen van data.
```

```
 uint8_t Check; //int voor return waarde van transmit functie om na te gaan of deze succesvol was.

 uint32_t ADC_Value[9] = {0}; // Array voor het opslaan van alle ADC waarden.
 uint8_t data[19] = {0}; // Elke ADC waarde is 12 bit. We gaan elke adc waarde opslaan in 2 bytes . 1
extra byte voor de knoppen.
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_SPI3_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void transformData(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////// transformData transformeert de 12bit ADC waarden naar 2 verzendbare 8 bit waardes
////////////////////// bv. ADC_Value[i] = 0000 1011 0001 1111
//////////////////////  data[dataIndex] = ADC_Value[i] & 0xFF = 0000 1011 0001 1111
//////////////////////           AND 0000 0000 1111 1111
//////////////////////                = 0000 0000 0001 1111 = 0001 1111 = Laagste 8 bits
//////////////////////     ADC_Value[i] >> 8 = 0000 1011 0001 1111 >> 8 = 0000 0000 0000 1011
//////////////////////     data[dataIndex] = (ADC_Value[i] >> 8) & 0x0F = 0000 0000 0000 1011
//////////////////////                                          AND 0000 0000 0000 1111
//////////////////////                                            = 0000 0000 0000 1011 = 0000 1011 = Hoogste 8 bits
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void transformData(void) {
 int dataIndex = 0; // Index voor data array
 for (int i = 0; i < 9; i++) {
  data[dataIndex] = ADC_Value[i] & 0xFF; // slaagt de laagste 8 bist op.
  dataIndex++;
  data[dataIndex] = (ADC_Value[i] >> 8) & 0x0F; // slaagt de 4 hoogste waardes op.
  dataIndex++;
 }
 data[dataIndex] = (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_12)) | (HAL_GPIO_ReadPin(GPIOC,
GPIO_PIN_15) << 1); // slaagt de status van de knoppen op.
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
```

```
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI3_Init();
  MX_ADC1_Init();
  MX_USART3_UART_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */

  NRF24_Init(); // Initialiseert de NRF24L01+ module.
  NRF24_TXMode(TxAddress, 10); // Configureert de NRF24L01+ module in Transmit mode.

  HAL_TIM_Base_Start_IT(&htim1); // Start timer 1 voor het sampelen van de ADC's.
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)ADC_Value, 9); //Start de ADC met DMA.
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
   if(ADC_Value[8] > 3100) //kijkt na of het batterij niveau nog boven 7V is.
   {
    transformData(); // Roept de transformData functie op
    Check = NRF24_Transmit(data); // Verzendt de getransformeerde data en krijgt een return waarde: 1
zenden is gelukt , 0 zenden is niet gelukt
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, Check); // visualiseert of het zenden gelukt is of niet met
een ledje.
   }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

```
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV2;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL15;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_TIM1;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{
```

```
/* USER CODE BEGIN ADC1_Init 0 */

/* USER CODE END ADC1_Init 0 */

ADC_ChannelConfTypeDef sConfig = {0};

/* USER CODE BEGIN ADC1_Init 1 */

/* USER CODE END ADC1_Init 1 */

/** Common config
*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_TRGO;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 9;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
hadc1.Init.LowPowerAutoWait = DISABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_1;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.SamplingTime = ADC_SAMPLETIME_19CYCLES_5;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_2;
sConfig.Rank = ADC_REGULAR_RANK_2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
```

```
sConfig.Channel = ADC_CHANNEL_3;
sConfig.Rank = ADC_REGULAR_RANK_3;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_4;
sConfig.Rank = ADC_REGULAR_RANK_4;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_10;
sConfig.Rank = ADC_REGULAR_RANK_5;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_11;
sConfig.Rank = ADC_REGULAR_RANK_6;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_12;
sConfig.Rank = ADC_REGULAR_RANK_7;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_13;
sConfig.Rank = ADC_REGULAR_RANK_8;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_14;
```

```
  sConfig.Rank = ADC_REGULAR_RANK_9;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief SPI3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI3_Init(void)
{

  /* USER CODE BEGIN SPI3_Init 0 */

  /* USER CODE END SPI3_Init 0 */

  /* USER CODE BEGIN SPI3_Init 1 */

  /* USER CODE END SPI3_Init 1 */
  /* SPI3 parameter configuration*/
  hspi3.Instance = SPI3;
  hspi3.Init.Mode = SPI_MODE_MASTER;
  hspi3.Init.Direction = SPI_DIRECTION_2LINES;
  hspi3.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi3.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi3.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI3_Init 2 */

  /* USER CODE END SPI3_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
```

```
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 0;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 5999;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */
```

```
  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 11500;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
```

```
  HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LED4_Pin|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin : Button1_Pin */
  GPIO_InitStruct.Pin = Button1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LED3_Pin */
  GPIO_InitStruct.Pin = LED3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : Button3_Pin Button4_Pin Button2_Pin */
  GPIO_InitStruct.Pin = Button3_Pin|Button4_Pin|Button2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : LED4_Pin PB6 PB7 */
  GPIO_InitStruct.Pin = LED4_Pin|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
```

```
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

```
/*
 * NRF24L01+.c
 *
 *      Credit/Source: https://www.youtube.com/watch?v=mB7LsiscM78&ab_channel=ControllersTech
 */
#include "stm32f3xx_hal.h"
#include "NRF24L01.h"

extern SPI_HandleTypeDef hspi3; //Define SPI handler
#define NRF24_SPI &hspi3

#define NRF24_CE_PORT GPIOB
#define NRF24_CE_PIN  GPIO_PIN_7

#define NRF24_CSN_PORT GPIOB
#define NRF24_CSN_PIN  GPIO_PIN_6

void CSN_Select (void) //Makes CSN(active LOW) 0 and selects the device
{
  HAL_GPIO_WritePin(NRF24_CSN_PORT, NRF24_CSN_PIN,0);
}

void CSN_UnSelect (void)//Makes CSN(active LOW) 1 and unselects the device
{
  HAL_GPIO_WritePin(NRF24_CSN_PORT, NRF24_CSN_PIN,1);
}

void CE_Enable (void) //Makes CE 1 and Enables the device
{
  HAL_GPIO_WritePin(NRF24_CE_PORT, NRF24_CE_PIN,1);
}

void CE_Disable (void) //Makes CE 0 and Disables the device
{
  HAL_GPIO_WritePin(NRF24_CE_PORT, NRF24_CE_PIN,0);
}

void nrf24_WriteReg (uint8_t Reg, uint8_t Data) // Write 1 byte to the register
{
 uint8_t buf[2]; // Buffer that stores 2 bytes
 buf[0] = Reg|1<<5; //First Byte is Register address
 buf[1] = Data; //Second Byte is the Data
 // Select the device
 CSN_Select();

 HAL_SPI_Transmit(NRF24_SPI,buf,2,1000); // Send buffer with size 2

 // Release the device
 CSN_UnSelect();

}

void nrf24_WriteRegMulti (uint8_t Reg, uint8_t *Data, int size) // Write many bytes starting from a
particular register
{
```

```
 uint8_t buf[1]; // Buffer that stores 1 byte
 buf[0] = Reg|1<<5; //Byte is Register address

 // Select the device
 CSN_Select();

 HAL_SPI_Transmit(NRF24_SPI,buf,1,100); // Send Register address
 HAL_SPI_Transmit(NRF24_SPI,Data,size,100); //Send all data bytes at ones

 // Release the device
 CSN_UnSelect();

}

uint8_t nrf24_ReadReg (uint8_t Reg) // Read 1 byte
{
 uint8_t data=0; // Variable to store data
 // Select the device
 CSN_Select();

 HAL_SPI_Transmit(NRF24_SPI,&Reg,1,100); // Send register address where we want to read data from
 HAL_SPI_Receive(NRF24_SPI,&data,1,100); // Read 1 byte from register

 // Release the device
 CSN_UnSelect();

 return data;
}

void nrf24_ReadRegMulti (uint8_t Reg,uint8_t *data,int size) // Read multiple bytes starting from register
pointed to variable where you want to store data
{
 // Select the device
 CSN_Select();

 HAL_SPI_Transmit(NRF24_SPI,&Reg,1,100); // Send register address where we want to read data from
 HAL_SPI_Receive(NRF24_SPI,data,size,1000); // Read X bytes from register

 // Release the device
 CSN_UnSelect();
}

//Send command to the NRF24
void nrfsendCmd(uint8_t cmd)
{
 // Select the device
 CSN_Select();

 HAL_SPI_Transmit(NRF24_SPI,&cmd,1,100);

 // Release the device
 CSN_UnSelect();
}

void nrf24_reset(uint8_t REG)
```

```
{
 if (REG == STATUS)
 {
  nrf24_WriteReg(STATUS, 0x00);
 }

 else if (REG == FIFO_STATUS)
 {
  nrf24_WriteReg(FIFO_STATUS, 0x11);
 }

 else {
 nrf24_WriteReg(CONFIG, 0x08);
 nrf24_WriteReg(EN_AA, 0x3F);
 nrf24_WriteReg(EN_RXADDR, 0x03);
 nrf24_WriteReg(SETUP_AW, 0x03);
 nrf24_WriteReg(SETUP_RETR, 0x03);
 nrf24_WriteReg(RF_CH, 0x02);
 nrf24_WriteReg(RF_SETUP, 0x0E);
 nrf24_WriteReg(STATUS, 0x00);
 nrf24_WriteReg(OBSERVE_TX, 0x00);
 nrf24_WriteReg(CD, 0x00);
 uint8_t rx_addr_p0_def[5] = {0xE7, 0xE7, 0xE7, 0xE7, 0xE7};
 nrf24_WriteRegMulti(RX_ADDR_P0, rx_addr_p0_def, 5);
 uint8_t rx_addr_p1_def[5] = {0xC2, 0xC2, 0xC2, 0xC2, 0xC2};
 nrf24_WriteRegMulti(RX_ADDR_P1, rx_addr_p1_def, 5);
 nrf24_WriteReg(RX_ADDR_P2, 0xC3);
 nrf24_WriteReg(RX_ADDR_P3, 0xC4);
 nrf24_WriteReg(RX_ADDR_P4, 0xC5);
 nrf24_WriteReg(RX_ADDR_P5, 0xC6);
 uint8_t tx_addr_def[5] = {0xE7, 0xE7, 0xE7, 0xE7, 0xE7};
 nrf24_WriteRegMulti(TX_ADDR, tx_addr_def, 5);
 nrf24_WriteReg(RX_PW_P0, 0);
 nrf24_WriteReg(RX_PW_P1, 0);
 nrf24_WriteReg(RX_PW_P2, 0);
 nrf24_WriteReg(RX_PW_P3, 0);
 nrf24_WriteReg(RX_PW_P4, 0);
 nrf24_WriteReg(RX_PW_P5, 0);
 nrf24_WriteReg(FIFO_STATUS, 0x11);
 nrf24_WriteReg(DYNPD, 0);
 nrf24_WriteReg(FEATURE, 0);
 }
}

void NRF24_Init (void)
{
 //Disable chip before config
 CE_Disable();


 nrf24_reset(0);

 nrf24_WriteReg(CONFIG,0); // Will be configured later

 nrf24_WriteReg(EN_AA,0); // No Auto Ack
```

```
 nrf24_WriteReg(EN_RXADDR,0); // Not enabling any data pipe right now

 nrf24_WriteReg(SETUP_AW,0x03); // 5 bytes for the TX/RX address

 nrf24_WriteReg(SETUP_RETR,0); // No retransmision (no shockburst)

 nrf24_WriteReg(RF_CH,0); // setup during Tx or Rx

 nrf24_WriteReg(RF_SETUP,0x0E); // Power = 0db, Data Rate = 2Mbps

 //Disable chip before config
 CE_Enable();
}

//setup TX mode
void NRF24_TXMode(uint8_t *address, uint8_t channel) // adress of receiver pipe and channel number
{
 //Disable chip before config
 CE_Disable();

 nrf24_WriteReg(RF_CH,channel); // Select the channel

 nrf24_WriteRegMulti(TX_ADDR,address,5); // Write the TX adress 5 bytes;

 //Power up device without changing other bits of register
 uint8_t config = nrf24_ReadReg(CONFIG);
 config = config|1<<1;
 nrf24_WriteReg(CONFIG,config);

 //Disable chip before config
 CE_Enable();
}

//transmit data
uint8_t NRF24_Transmit (uint8_t *data)
{
 uint8_t cmdtosend = 0;

 // Select the device
 CSN_Select();

 //Payload command
 cmdtosend = W_TX_PAYLOAD;
 HAL_SPI_Transmit(NRF24_SPI,&cmdtosend,1,100);

 // send the payload
 HAL_SPI_Transmit(NRF24_SPI,data,32,1000);

 // Select the device
 CSN_UnSelect();

 HAL_Delay(1);

 uint8_t fifostatus = nrf24_ReadReg(FIFO_STATUS);
```

```
 if ((fifostatus&(1<<4)) && (!(fifostatus&(1<<3))))
 {
  cmdtosend = FLUSH_TX;
  nrfsendCmd(cmdtosend);
  nrf24_reset(FIFO_STATUS);
  return 1;
 }

 return 0;
}

void NRF24_RXMode(uint8_t *address,uint8_t channel)
{
 //Disable chip before config
 CE_Disable();

 nrf24_WriteReg(RF_CH,channel); // Select the channel

 uint8_t en_rxaddr = nrf24_ReadReg(EN_RXADDR);
 en_rxaddr = en_rxaddr|(1<<1); // first make sure no other pipes get disabled.
 nrf24_WriteReg(EN_RXADDR,en_rxaddr); // select data pipe 1
 nrf24_WriteRegMulti(RX_ADDR_P1,address,5); // 5 byte address for data pipe1

 nrf24_WriteReg(RX_PW_P1,32); // 32 byte payload size for pipe1

 //Power up device without changing other bits of register
 uint8_t config = nrf24_ReadReg(CONFIG);
 config = config|(1<<1) | (1<<0);
 nrf24_WriteReg(CONFIG,config);

 //Disable chip before config
 CE_Enable();
}

uint8_t isDataAvailable(int pipenum)
{
 uint8_t status = nrf24_ReadReg(STATUS);

 if ((status&(1<<6))&&(status&(pipenum<<1)))
 {
  nrf24_WriteReg(STATUS,(1<<6));
  return 1;
 }
 return 0;
}

void NRF24_Receive (uint8_t *data)
{
 uint8_t cmdtosend = 0;

 // Select the device
 CSN_Select();

 //Payload command
```

```
 cmdtosend = R_RX_PAYLOAD;
 HAL_SPI_Transmit(NRF24_SPI,&cmdtosend,1,100);

 // send the payload
 HAL_SPI_Receive(NRF24_SPI,data,19,1000);

 // Select the device
 CSN_UnSelect();

 HAL_Delay(1);

 cmdtosend = FLUSH_RX;
 nrfsendCmd(cmdtosend);
}
```

```
/*
 * NRF24L01.h
 *
 *      Credit/Source: https://www.youtube.com/watch?v=mB7LsiscM78&ab_channel=ControllersTech
 */

#ifndef INC_NRF24L01_H_
#define INC_NRF24L01_H_

void NRF24_Init (void);
void NRF24_TXMode(uint8_t *address, uint8_t channel);
uint8_t NRF24_Transmit (uint8_t *data);
void NRF24_RXMode(uint8_t *address,uint8_t channel);
uint8_t isDataAvailable(int pipenum);
void NRF24_Receive (uint8_t *data);

/* Memory Map */
#define CONFIG      0x00
#define EN_AA       0x01
#define EN_RXADDR   0x02
#define SETUP_AW    0x03
#define SETUP_RETR  0x04
#define RF_CH       0x05
#define RF_SETUP    0x06
#define STATUS      0x07
#define OBSERVE_TX  0x08
#define CD          0x09
#define RX_ADDR_P0  0x0A
#define RX_ADDR_P1  0x0B
#define RX_ADDR_P2  0x0C
#define RX_ADDR_P3  0x0D
#define RX_ADDR_P4  0x0E
#define RX_ADDR_P5  0x0F
#define TX_ADDR     0x10
#define RX_PW_P0    0x11
#define RX_PW_P1    0x12
#define RX_PW_P2    0x13
#define RX_PW_P3    0x14
#define RX_PW_P4    0x15
#define RX_PW_P5    0x16
#define FIFO_STATUS 0x17
#define DYNPD     0x1C
#define FEATURE     0x1D

/* Instruction Mnemonics */
#define R_REGISTER    0x00
#define W_REGISTER    0x20
#define REGISTER_MASK 0x1F
#define ACTIVATE      0x50
#define R_RX_PL_WID   0x60
#define R_RX_PAYLOAD  0x61
#define W_TX_PAYLOAD  0xA0
#define W_ACK_PAYLOAD 0xA8
#define FLUSH_TX      0xE1
#define FLUSH_RX      0xE2
```

```
#define REUSE_TX_PL   0xE3
#define NOP           0xFF

#endif /* INC_NRF24L01_H_ */
```