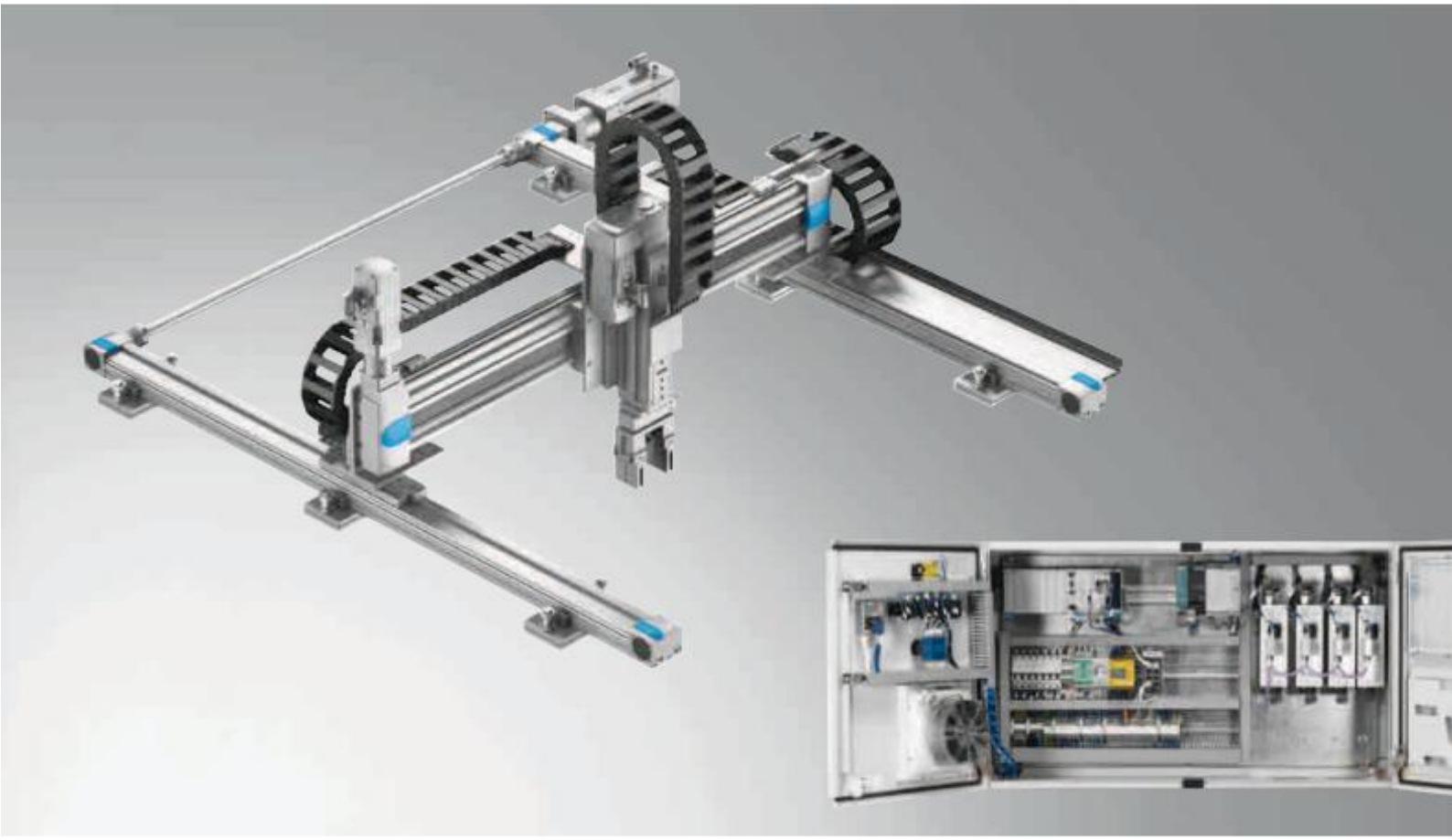
Nieuwe manier van aansturen E-drives bij Festo



T.R. van der Marel

18-12-2014

Auteur:	**Tim van der Marel**
Studentnummer:	10001972
Email:	timvandermarel@gmail.com
Telefoon:	0627542940
Datum:	18-12-2014
Onderwijsinstelling:	**Haagse Hogeschool**
Opleiding:	Elektrotechniek
Stagebegeleiders:	J.E.J. op den Brouw
	M. Can
Bedrijf:	**Festo**
Bedrijfsmentoren:	T. Kiefte
	S. Sengers
Email:	kte@festo.nl
	Sge@festo.nl
Telefoon:	+31 (15) 2518788, +31 (6) 12818704
	+31 (15) 2518890, +31 (6) 14234903

Voorwoord

Deze scriptie heb ik geschreven in het kader van mijn afstuderen van de opleiding Elektrotechniek aan de Haagse Hogeschool te Delft. Afstuderen gebeurde in de vorm van een afstudeeropdracht. Deze afstudeeropdracht heb ik uit mogen voeren bij Festo onder begeleiding van Tim Kiefte en Sebastiaan Sengers.

Met dit document wil ik de lezers informeren over de opdracht, hoe deze is uitgevoerd, de keuzes die gemaakt zijn, handige functies die zijn tegengekomen en het uiteindelijke resultaat.

Ik wil graag mijn begeleiders vanuit Festo, Tim Kiefte en Sebastiaan Sengers, bedanken voor hun begeleiding tijdens mijn afstudeerstage. Daarnaast wil ik Jeffrey Boers bedanken voor zijn suggesties en hulp bij het maken en bedenken van de nieuwe manier van aansturen van de e-drives.

Als laatste wil ik graag mijn begeleiders vanuit de Haagse Hogeschool bedanken, Jesse op den Brouw en Mehmet Can, voor hun begeleiding.

Delft, 18-12-2014

Samenvatting

Festo is één van de marktleiders bij het leveren van automatiseringstechniek. Voor het aansturen van deze automatiseringstechniek maakt Festo gebruik van Programmable Logic Controllers (PLC's), die worden geprogrammeerd met behulp van het programma CODESYS. Festo heeft echter een aantal vragen over de huidige werkwijze. Zo wil Festo weten of de huidige manier van softwareprogrammering efficiënter kan waardoor er binnen de organisatie gestandaardiseerd geprogrammeerd kan worden. Om dit te kunnen onderzoeken zal er eerst een analyse van het huidige systeem moeten worden gemaakt.
In dit document worden de volgende onderdelen uitgelicht en besproken:

- Probleemstelling.
- Analyse van het huidige systeem.
- Verschillende manieren van programmeren met behulp van het programma CODESYS.
- De nieuwe manier van het aansturen van de e-drives en de keuzes die gemaakt zijn.
- Conclusie en aanbeveling naar aanleiding van de probleemstelling, de analyse van het huidige systeem, de mogelijkheden van CODESYS en de nieuwe manier van aansturen.

Bij de analyse van het huidige systeem is er gekeken welke onderdelen er gebruikt worden om een electromechanical drive aan te sturen. Hieruit komt naar boven dat er gebruik wordt gemaakt van een PLC die een motor controller aanstuurt door middel van een CANopen connectie. De PLC maakt gebruik van een functieblok die gebruik maakt van Festo Handling and Positioning Profile (FHPP).
Er zijn verschillende manieren van programmeren met behulp van het programma CODESYS, de twee meest voorkomende bij Festo zijn structured text en functionblock diagram. Er is hier gekeken hoe er objectgeoriënteerd geprogrammeerd kan worden en naar de huidige manier van programmeren bij Festo.
De nieuwe manier van het aansturen van de e-drives gebeurt met behulp van 2 functieblokken. Één functieblok voor het laten bewegen van e-drive en één om te bepalen wanneer er bewogen moet worden. Bij het bedenken en maken van deze functieblokken is er met een objectgeoriënteerd oog gekeken hoe deze op elkaar aangesloten kunnen worden. Door gebruik te maken van 2 functieblokken is de nieuwe manier van aansturen van de e-drives modulair.
Het vernieuwde CTRL functieblok heeft minder ingangen dan het huidige CTRL functieblok. De meeste ingangen zijn samengevoegd tot 1 ingang door middel van een struct.

Inhoudsopgave

Figuren

Tabellen

1. Inleiding

Deze scriptie is uitgevoerd naar aanleiding van het afstuderen van een student van de opleiding Elektrotechniek aan de Haagse Hogeschool te Delft. Dit gebeurde bij en voor Festo: één van de marktleiders bij het leveren van automatiseringstechniek. Voor het aansturen van deze automatiseringstechniek maakt Festo gebruik van Programmable Logic Controllers (PLC's), die worden geprogrammeerd met behulp van het programma CODESYS. Festo heeft echter een aantal vragen over de huidige werkwijze. Zo wil Festo weten of de huidige manier van softwareprogrammering efficiënter kan waardoor er binnen de organisatie gestandaardiseerd geprogrammeerd kan worden. Dit zou een meerwaarde voor Festo betekenen aangezien het dan eenvoudiger is elkaars software te lezen en te begrijpen waardoor applicaties sneller opgezet kunnen worden. De huidige manier van softwareprogramming wordt gedaan met behulp van functieblokken. Een andere vraag die binnen Festo leeft heeft betrekking op de laatste versie van CODESYS die de mogelijkheid biedt om object georiënteerd te programmeren. Festo is geïnteresseerd in de mogelijkheden van object georiënteerd programmeren en wil weten of dit geschikt is voor hun activiteiten.
Door bovenstaande vragen in deze scriptie te onderzoeken kunnen er functieblokken gemaakt worden die voor de klant relatief eenvoudig te gebruiken zijn. Om dit te toetsen is contact met bestaande klanten noodzakelijk. De klanten zijn immers de gebruiker en kunnen aangeven waar zij problemen ervaren. Dit kan vervolgens weer bijdragen aan nieuwe functieblokken die zo goed mogelijk aansluiten op de behoeften van de klanten.

In deze scriptie zal er dus een antwoord komen op de vraag:
Is de huidige manier van programmeren bij Festo de beste manier voor de organisatie?
Er zal onderzoek gedaan worden naar de mogelijkheden van CODESYS. Er zullen functieblokken gemaakt worden die voor de klant relatief eenvoudig in gebruik te nemen zijn.

Deze vraag wordt beantwoord door een analyse te doen van het huidige systeem in hoofdstuk 3. Het onderzoeken van de mogelijkheden van CODESYS gebeurt in hoofdstuk 4. Door de analyse van het huidige systeem en het onderzoek van de mogelijkheden van CODESYS is er een nieuwe manier van aansturen van e-drives bedacht en gemaakt. Dit wordt besproken in de hoofdstukken 5 t/m 7. In hoofdstuk 8 worden de functies die bedacht en gebruikt zijn bij de nieuwe manier van aansturen uitgelegd.

2. Probleemstelling

Bij Festo wordt er momenteel gewerkt met een functieblok voor het bewegen van electromechanical drives (e-drives). Dit functieblok heeft veel in- en uitgangen waardoor het lastig kan zijn om toe te passen en/of uit te leggen. Omdat Festo te maken heeft met klanten die veelal zelf programmeren wil Festo de toepassing en uitleg makkelijker maken voor de klanten. Dit is mogelijk door met behulp van CODESYS het functieblok aan te passen en eventueel andere functieblokken te maken die het gebruik van het functieblok voor het bewegen van drives eenvoudiger maakt. Bij het onderzoeken van nieuwe functieblokken zijn er vanuit Festo een aantal eisen geformuleerd waaraan voldaan moet worden:

- Minder ingangen op het functieblok.
- Het functieblok moet eenvoudig in het gebruik zijn voor de klanten van Festo.
- Er zullen functieblokken geschreven worden die opstellingen aan kunnen sturen die in 1D en 2D kunnen bewegen.
- Het functieblok moet uitbreidbaar zijn door onder andere verschillende functieblokken aan te kunnen sluiten.
- Er zal met een objectgeoriënteerd oog naar de functieblokken gekeken moeten worden en hoe deze op elkaar aansluiten.

Daarnaast zijn er een aantal wensen:

- Het functieblok moet zo geprogrammeerd worden dat de tijd die nodig is om een positie in te laden en hier naartoe bewegen zo kort mogelijk is.

Dit resulteert in onderstaande probleemstellingen die leidend zijn in deze scriptie:
Kan de aansturing van de e-drives eenvoudiger en voldoen aan de eisen vanuit Festo? En is de huidige manier van programmeren bij Festo de beste manier?

Wat er opgeleverd gaat worden:

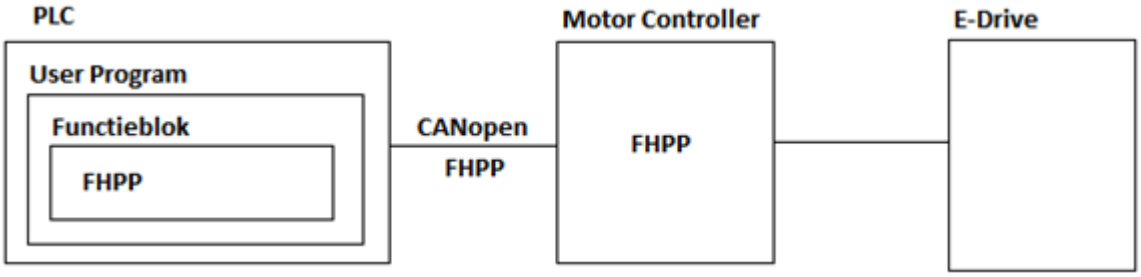
- Nieuw functieblok voor het bewegen e-drive die aan de eisen van Festo voldoet.
- Functieblokken voor 1 dimensionale en 2 dimensionale opstellingen.
- Visueel aspect waarmee de ingangen en uitgangen van de functieblokken weergegeven en aangepast kunnen worden. Dit zal gebeuren met behulp van het programma CODESYS

Wat er niet opgeleverd gaat worden:

- Erdoor declaratie.
- Functieblokken voor 3 dimensionale tot 6 dimensionale opstellingen.

3. Analyse huidig systeem

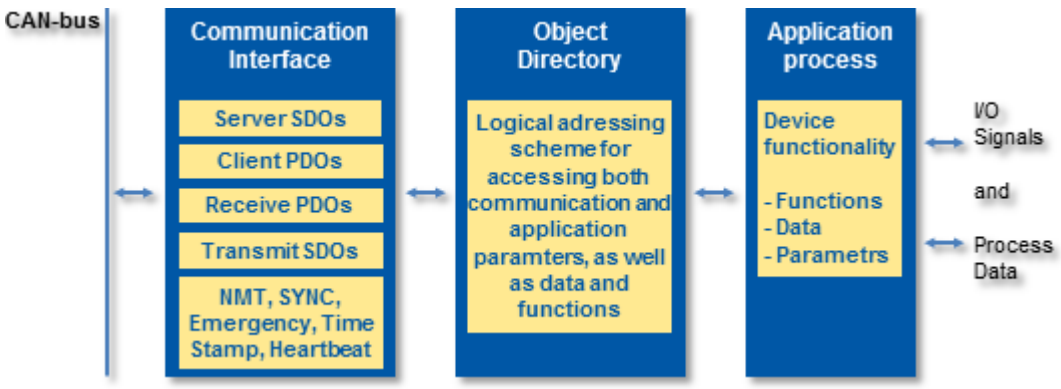
Voordat de nieuwe manier van aansturen van de e-drives bedacht en gemaakt kan worden zal er eerst gekeken moeten worden naar het huidige systeem. Het huidige systeem is opgebouwd uit een PLC, een motor controller en een electromechanical drive (e-drive). De motor controller zorgt ervoor dat de e-drive beweegt, de motor controller wordt weer aangestuurd door de PLC. Dit gebeurt door middel van een CANopen connectie via een Campus. Deze PLC stuurt de motor controller aan met behulp van een functieblok (zie Figuur 1). In dit hoofdstuk zal uitgelegd worden wat CANopen is en hoe de PLC de motor controller aanstuurt.



Figuur 1: Overzicht huidig systeem

3.1 CANopen[6]

CANopen is communicatie- en profiel standaard, zoals gedefinieerd in CENELEC EN 50325-4, voor gedistribueerde industriële automatiseringssystemen gebaseerd op CAN. De basis van CANopen is een profielfamilie die op een “Communicatieprofiel” (Communications Interface) gebaseerd is (zie Figuur 2). Deze specificeert de basis communicatiemechanismen en verschillende Apparaat- of Toepassingsprofielen die de belangrijkste soorten aangesloten apparatuur beschrijven. Hierbij kan gedacht worden aan digitale en analoge I/O-modules, (motor) drivers, programmeerbare controllers of encoders. In deze apparaat profielen worden functionaliteit, parameters en de toegang tot het verwerken van data gespecificeerd. Op basis van deze standaard profielen kan op exact dezelfde manier toegang worden verkregen tot dezelfde soort apparatuur van verschillende producenten. Zo hebben bijvoorbeeld alle stappenmotoren besturingen in principe dezelfde aansturing. Zodoende ontstaat er een grote onafhankelijkheid van de desbetreffende fabrikant en kunnen modules van verschillende leveranciers eenvoudig met elkaar gecombineerd of juist uitgewisseld worden.



Figuur 2: CANopen overzicht onderdelen

Object Directory

Het hart van de CANopen standaard is de beschrijving van de apparaat functionaliteit met behulp van een zogenaamde Object Directory (OD) (zie Figuur 2). Elke invoer in de OD biedt een basis voor een gestandaardiseerde netwerktoegang naar zowel “toepassingsobjecten” zoals ingangs- en uitgangssignalen, apparaat parameters, apparaat functies of netwerkvariabelen van een apparaat als naar ‘communicatieobjecten’ die de communicatiefunctionaliteit van het apparaat beschrijven. Zo kunnen de eigenschappen van apparaten beschreven en geconfigureerd worden.

Communicatie en berichten

Net zoals andere veldbussystemen onderscheidt CANopen twee basis datatransmissiemechanismen: de hoge snelheidsuitwisseling van kleine stukken verwerkingsdata via “Process Data Objects” (PDO) en de toegang tot de gegevens in het OD via de “Service Data Objects” (SDO). PDO’s worden getriggerd door een gebeurtenis, zijn cyclisch of worden opgevraagd als zendobjecten zonder aanvullend protocol overhead. Een PDO kan worden gebruikt voor de transmissie van maximaal 8 bytes aan data. In combinatie met een synchronisatiebericht kan de transmissie en de acceptatie van PDO’s door het volledige netwerk worden gesynchroniseerd (“Synchrone PDO’s”). De toewijzing van toepassingsobjecten aan een PDO (Transmissie-Object) is middels datastructuren (“PDO Mapping”) aan te passen binnen de Object Directory.
Naast deze standaard communicatie voorziet CANopen ook “noodoproepen” met een hoge prioriteit. Hiermee kan het disfunctioneren van een apparaat worden gemeld. Een centrale en gemeenschappelijke systeemtijd wordt verzorgd door een ‘central timing’-bericht. Verder beschikt CANopen over managementfunctionaliteiten zoals het controleren en monitoren van de communicatiestatus van de nodes.
De transmissie van SDO wordt uitgevoerd als een bevestigde dienst met twee CAN voorwerpen in de vorm van een logische peer-to-peer-verbinding tussen twee netwerkapparaten. De adressering van de desbetreffende object woordenboekingangen wordt bereikt door het specificeren index en sub-index van de invoer in het veld gegevens van de betrokken CAN frames. Verzonden gegevens kunnen een onbeperkte lengte hebben. De overdracht van SDO berichten gaat via een aanvullend protocol overhead.
Festo heeft zijn eigen protocol geschreven voor het uitwisselen van data via de CANopen, namelijk FHPP protocol. Dit wordt uitgelegd in paragraaf 3.2.1.

3.2 Aansturing motor controller met behulp van een PLC

De PLC stuurt de motor controller aan met behulp van een functieblok dat gebruik maakt van Festo Handling and Positioning Profile (FHPP).
Dit functieblok is uitgebreid, houdt rekening met veel opties en is hierdoor flexibel. Dit komt onder andere doordat er rekening gehouden wordt met verschillende modes waarin gewerkt kan worden. Dit functieblok is zo geprogrammeerd dat er met de ingang data verschillende handelingen verricht kunnen worden afhankelijk van de mode waarin gewerkt wordt. Wat FHPP inhoudt, wordt uitgelegd in paragraaf 3.2.1. De in- en uitvoergegevens van het functieblok worden via CANopen uitgewisseld tussen de PLC en de motor controller. De werking van dit functieblok wordt uitgelegd in paragraaf 3.2.2. De uitleg van de code van het CTRL functieblok is te vinden in paragraaf 3.2.3 en de code van het CTRL functieblok is te vinden in Bijlage I: Code huidige manier van aansturen electromechanical drives.

3.2.1 FHPP[2]

FHPP staat voor Festo Handling and Positioning Profile en wordt gebruikt bij de aansturing van de motor controller door de PLC. Dit is een geoptimaliseerde data profiel speciaal afgestemd op de gewenste toepassingen en op de hantering en positionering taken. Het biedt gebruikers informatie over de individuele kenmerken van een controller. Het FHPP protocol bepaalt 8 bytes van de invoergegevens en 8 bytes van de uitvoergegevens deze 8 bytes zijn gebaseerd op de 8 bytes die een PDO van CANopen maximaal kan bevatten bij transmissie. Hiervan staan de eerste twee bytes vast (zie Figuur 3). De overige bytes zijn afhankelijk van de FHPP operationele modus die is geselecteerd. Hoe de modus wordt geselecteerd wordt uitgelegd in paragraaf 3.2.3.

Record selection

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Output data	CCON	CPOS	Record no.	Reserved	Reserved			
Input data	SCON	SPOS	Record no.	RSB	Actual position			

Direct mode

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Output data	CCON	CPOS	CDIR	Setpoint value 1	Setpoint value 2			
Input data	SCON	SPOS	SDIR	Actual value 1	Actual value 2			

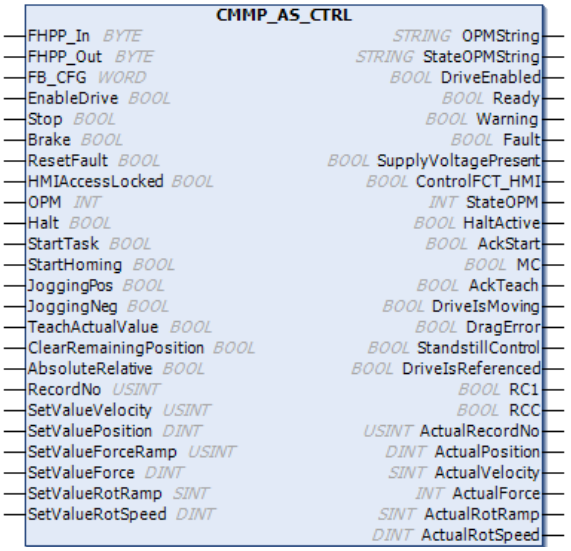
Figuur 3: In- en Uitvoergegevens FHPP

De modus waar het meest gebruik van wordt gemaakt is de Direct Mode. In de Direct mode worden positioneringstaken direct geformuleerd in uitvoergegevens, dit gebeurd met behulp van de bytes CCON, SCON, CPOS, SPOS, CDIR en SDIR. Deze bytes bezitten allemaal 8 bits met hun eigen functie, zoals te zien in Bijlage VIII: FHPP Manual kan elk van deze bits een waarde 1 of 0 hebben. Door de juiste bits 1 of 0 te maken kan er bewogen worden met de e-drives, hoe dit precies wordt gedaan wordt uitgelegd in de paragrafen 3.2.2 en 3.2.3.

De bytes 4 t/m 8 bevatten de waardes van de ingestelde en actuele waardes van de positie en snelheid. Byte 4 heeft per mode een andere betekenis, de meest voorkomende is de Profile Position mode, hier heeft de waarde van Byte 4 een percentage van de maximale ingestelde waarde van de snelheid (zie Bijlage VIII: FHPP Manual). Daarmee wordt bedoeld dat als Byte 4 de waarde 10 heeft er 10% van de maximale ingestelde snelheid wordt bedoeld. Hierdoor is de maximale waarde hiervan 100 en past dit in 1 byte, waarvan de maximale waarde 255 is. De maximale snelheid wordt ingesteld in de motor controller.
Byte 5 t/m 8 bezit de waarde van de ingestelde en actuele positie, deze waarde is opgeslagen als een binair getal. Er worden dus 32 bits gebruikt om de waarde op te slaan. Hier is voor gekozen aangezien er 1 byte (8 bits) maximaal de waarde 255 kan bevatten, afhankelijk van de ingestelde vector kan deze waarde oplopen tot 1000000 bij een e-drive met een lengte van 1 meter.

3.2.2 De werking van het huidige functieblok

Het huidige functieblok heeft veel in- en uitgangen en is hierdoor soms lastig toe te passen en/of uit te leggen (zie Figuur 4). Veel van deze in- en uitgangen worden niet gebruikt, alleen in zeer specifieke gevallen wanneer er in bepaalde modus wordt gewerkt. Om de meest voorkomende werking uit te leggen wordt er gekeken naar de ingangen die het meest van toepassing zijn en de functie van deze ingangen (zie Tabel 1).



Figuur 4: Huidig functieblok voor het aansturen van drives

Tabel 1: Ingangen en hun functies huidig functieblok

Ingang	Functie
FHPPIN	Uitwisselen invoergegevens met de motor controller met behulp van het FHPP protocol.
FHPPOUT	Uitwisselen uitvoergegevens met de motor controller met behulp van het FHPP protocol.
EnableDrive	Ervoor zorgen dat de drive kan bewegen. Dit komt doordat deze de vrijgave regelt.
Stop	Stoppen.
Brake	Rem handmatig bekrachtigen.
ResetFault	Als er een fout is deze resetten. Waardoor er verder kan worden gegaan met de beweging.
OPM	Mode selectie.
Halt	Rem waarmee de overgebleven beweging wordt onthouden.
StartTask	Start met de beweging (mits de juiste bits goed staan).
StartHoming	Start met homen (het referentiepunt van de drive bepalen).
JoggingPos	In positieve richting bewegen met de ingestelde snelheid.
JoggingNeg	In negatieve richting bewegen met de ingestelde snelheid.
SetValueVelocity	De snelheid instellen waarmee bewogen moet worden.
SetValuePosition	De positie instellen waarnaar bewogen moet worden.
SetValueForceRamp	De koppel instellen waarmee bewogen moet worden. Deze waarde is het percentage van de maximale koppel of nominale koppel.
SetValueForce	De koppel instellen waarmee bewogen moet worden. Deze waarde is het percentage van de maximale koppel.
SetValueRotRamp	De snelheidsdrempel instellen waarmee minimaal bewogen moet worden.
SetValueRotSpeed	De snelheid als een harde waarde instellen waarmee bewogen moet worden.

3.2.3 Uitleg code huidig functieblok

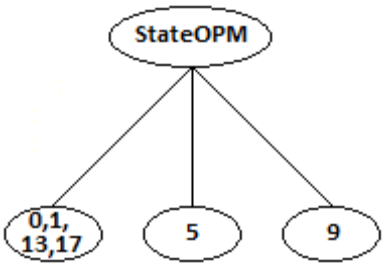
In het huidige functieblok wordt er begonnen met het uitlezen van de invoergegevens, dit gebeurt met behulp van een pointer naar een array van het type byte. De eerste 2 bytes van de array worden in de locale bytes SCON en SPOS gezet, waarvan vervolgens elke bit in de bijbehorende variabele wordt gezet (zie Figuur 5). Deze bytes en variabelen komen overeen met de bytes en bits van het FHPP protocol. Deze bits worden onder andere gebruikt bij het instellen van de mode waarin gewerkt gaat worden, de direct mode of record mode. Met behulp van de byte SDIR en de variabelen StateOperationModeB1 en StateOperationModeB2 bits kan StateOPM worden bepaald. StateOPM wordt gebruikt om de waarde van ACTUALVALUE1 en ACTUALVALUE2 in een uitgang te zetten. ACTUALVALUE1 EN ACTUALVALUE 2 krijgen de waardes van byte 3 en byte 4 t/m 7. Er wordt voor ACTUALVALUE1 gebruik gemaakt van 1 byte en 4 bytes voor ACTUALVALUE2, meestal beginnend met de laagste byte. Zoals uitgelegd in paragraaf 3.2.1 en terug te vinden is in de datasheet die bijgevoegd is inBijlage VIII: FHPP Manual, heeft de waarde van ACTUALVALUE1 een andere betekenis per mode die ingesteld wordt met behulp van de bits uit SDIR en de bits StateOperationModeB1 en StateOperationModeB2. Aangezien er in het nieuwe functieblok uit wordt gegaan dat er gewerkt wordt in Direct mode Position Control wordt er alleen hier naar gekeken.

```
SCON := pFHPP^[0];      // SCON cast to its bits
DriveEnabled                := SCON.0;
Ready                       := SCON.1;
Warning                     := SCON.2;
Fault                       := SCON.3;
SupplyVoltagePresent        := SCON.4;
ControlFCT_HMI              := SCON.5;
StateOperationModeB1        := SCON.6;
StateOperationModeB2        := SCON.7;

SPOS := pFHPP^[1];      // SPOS cast to its bits
HaltActive                  := SPOS.0;
AckStart                    := SPOS.1;
MC                          := SPOS.2;
AckTeach                    := SPOS.3;
DriveIsMoving               := SPOS.4;
DragError                   := SPOS.5;
StandstillControl           := SPOS.6;
DriveIsReferenced           := SPOS.7;
```

Figuur 5: Variabelen in SCON en SPOS zetten

Als er gewerkt wordt in Direct mode Position Control wordt er in het state diagram alleen gebruik gemaakt van state 1. In deze state zal de waarde van ACTUALVALUE1 (met een conversie) in de uitgang ActualVelocity worden gezet en zal de waarde van ACTUALVALUE2 in de uitgang ActualPosition worden gezet. De uitgangen ActualForce, ActualRotRamp en ActualRotSpeed krijgen de waarde 0. Deze zijn echter niet van toepassing in het nieuwe functieblok. Het is ook mogelijk dat StateOPM een andere waarde heeft dan 0, 1, 5, 9, 13 of 17 dan krijgen de uitgangen ActualVelocity, ActualForce, ActualRotRamp, ActualRotSpeed en ActualPosition de waarde 0 aangezien er dan niet in de DirectMode gewerkt wordt. Dit zal niet van toepassing zijn in het nieuwe functieblok.



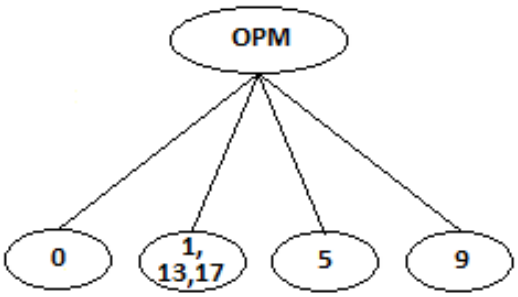
Figuur 6: State Diagram ACTUALVALUE

Zodra de waardes van ACTUALVALUE 1 en ACTUALVALUE2 in de juiste uitgangen zijn gezet wordt de uitvoergegevens ingelezen. Dit gebeurt met behulp van de bytes CCON en CPOS. De waardes van de ingangsvariabelen worden in de juiste bits van CCON en CPOS gezet zodat deze overeen komen met de bytes en bits van CCON en CPOS van de FHPP (zie Figuur 7). Met behulp van de variabelen OperationModeB1 en OperationModeB2 en de byte CDIR, die in de 3e byte wordt gezet van de FHPP, wordt er bepaald welke waardes van de ingangsvariabelen er in de bytes 4 en 5 t/m 8 komen. Dit gebeurt door middel van een state diagram (zie Figuur 8). Aangezien er in het nieuwe functieblok uit wordt gegaan dat er gewerkt wordt in Direct mode Position Control wordt er alleen hier naar gekeken. Als er gewerkt wordt in Direct mode Position Control wordt er in het state diagram alleen gebruik gemaakt van state 1. In deze state wordt de waarde van SetValuevelocity in SETVALUE1 gezet en wordt de waarde van SetValuePosition in SETVALUE2 gezet. Zodra dat gedaan is wordt de waarde van SETVALUE1 in een pointer naar de byte 4 gezet en wordt de waarde van SETVALUE2 in een pointer naar de bytes 5 t/m 8 gezet, meestal beginnend met de laagste byte.

```
(* ----- CCONByte_1 ----- *)
CCON.0   := EnableDrive;
CCON.1   := Stop;
CCON.2   := Brake;
CCON.3   := ResetFault;
CCON.4   := FALSE;
CCON.5   := HMIAccessLocked;
CCON.6   := OperationModeB1;
CCON.7   := OperationModeB2;
pFHPP^[0] := CCON;

(* ----- CPOSByte_2 ----- *)
CPOS.0   := Halt;
CPOS.1   := StartTask;
CPOS.2   := StartHoming;
CPOS.3   := JoggingPos;
CPOS.4   := JoggingNeg;
CPOS.5   := TeachActualValue;
CPOS.6   := ClearRemainingPosition;
CPOS.7   := FALSE;
pFHPP^[1] := CPOS;
```

Figuur 7: De ingangsvariabelen in CCON en CPOS zetten



Figuur 8: State Diagram SETVALUE

4. Mogelijkheden CODESYS

CODESYS is het programma dat bij Festo gebruikt wordt om de PLC's mee te programmeren. CODESYS is een ontwikkelomgeving om automatiseringsproducten te programmeren volgens de internationale industriële standaard IEC 61131-3. Zoals gedefinieerd in de IEC 61131-3 zijn de volgende 5 programmeertalen beschikbaar binnen CODESYS:

- IL (Instruction List), hier valt te denken aan assembleertaal
- ST (Structured Text),Structured Text is een high level language die blok gestructureerd is en die syntactisch lijkt op Pascal.
- LD (Ladder Diagram), Ladder Diagram wordt op grote schaal gebruikt bij sequentiële besturing van een proces of productieoperatie. Ladder diagram is handig om te gebruiken bij eenvoudige, maar kritische systemen.
- FBD (Function Block Diagram), een Function Block Diagram beschrijft een functie tussen ingangen en uitgangen en dient het systeem te beschrijven in één beeld.
- SFC (Sequential Function Chart), Sequential Function Chart is een grafische programmeertaal. Deze kan gebruikt worden bij processen die in stappen ingedeeld kunnen worden.

Daarnaast is er nog een extra programmeermethode beschikbaar in CODESYS die niet is gedefinieerd in IEC 61131-3, namelijk:

- CFC (Continuous Function Chart),Continuous Function Chart is een grafische programmeertaal die gebaseerd is op FBD. In tegenstelling tot FBD heeft deze geen netwerken, maar kan er vrij gepositioneerd worden met de grafische elementen.

Festo maakt voornamelijk gebruik van structured text en/of function block diagram als programmeertaal. Met structured text is het onder andere mogelijk om objectgeoriënteerd te programmeren. Dit wordt uitgelegd in paragraaf 4.1. Op dit moment maakt Festo alleen gebruik van een Functieblok, deze manier van programmeren wordt uitgelegd in paragraaf 4.1.3.

4.1 Objectgeoriënteerd programmeren

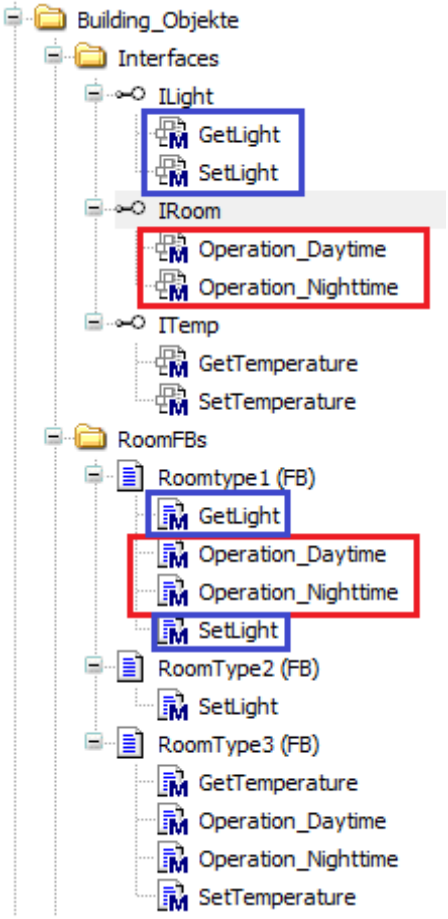
Objectgeoriënteerd programmeren is één van de manieren waarop geprogrammeerd kan worden bij structured text. Bij CODESYS maak je met objectgeoriënteerd programmeren gebruik van function blocks, methods, interfacemethods en interfaces, deze worden in deze paragraaf verder besproken. Dit zal gebeuren met behulp van een voorbeeld programma van CODESYS.

4.1.1 Interface

Een interface kan gezien worden als het domein waarin de methods staan geïnitialiseerd. Als een functieblok gebruik wil maken van een van deze methods zal de interface geïmplementeerd moeten worden. Hoe dit moet zal in paragraaf 4.1.3 worden besproken. In de code van de interface staat alleen de initialisatie van de interface (zie Figuur 9).

```
INTERFACE IRoom
```

Figuur 9: Interface



Figuur 10: Interfaces, Interfacemethods, Functieblokken en Methods

4.1.2 Interfacemethod

De interfacemethods zitten gekoppeld aan de interface. In de interfacemethod wordt de type van de return waarde geïnitialiseerd en is het mogelijk om ingangsvariabelen te declareren (zie Figuur 11). Als een functieblok gebruik wil maken van een method zal deze toegevoegd moeten worden aan het functieblok door middel van de interface van de method te implementeren (zie Figuur 12) en door de method toe te voegen zoals te zien in Figuur 10. Deze moet dan dezelfde benaming hebben als de interfacemethod.

```
METHOD SetLight : BOOL
VAR_INGANG
      b:BOOL;
END_VAR
VAR
END_VAR
```

Figuur 11: Interfacemethod

4.1.3 Functieblok

Een functieblok kan op verschillende manieren gebruikt worden. Een functieblok kan, zoals uitgelegd in hoofdstuk 3, functies bevatten waarmee bepaalde ingangen worden gebruikt om bepaalde uitgangen te krijgen. In het voorbeeld programma van de CODESYS wordt een functieblok op een andere manier gebruikt. Een functieblok wordt gebruikt om de interface te implementeren zodat de methods van deze interface gebruikt kunnen worden in het functieblok. Er is te zien dat er slechts één variabele gedefinieerd wordt. Deze variabele wordt gebruikt door één van de methods van de interfaces IRoom of ILight. In dit functieblok is het mogelijk om de methods van IRoom en ILight aan te roepen en deze worden dan uitgevoerd als functies.

```
FUNCTION_BLOCK Roomtype1 IMPLEMENTS IRoom, ILight
      // simple room with only one light
VAR_INGANG
END_VAR
VAR_OUTPUT
END_VAR
VAR
      bLight:BOOL;
END_VAR
```

Figuur 12: Functieblok

4.1.4 Method

Een method kan gezien worden als de functie van een functieblok waarin deze gebruikt wordt. Dit komt doordat deze op dezelfde manier wordt aangeroepen als een functie. In het voorbeeldprogramma zou dat in het functieblok op de volgende manier gebeuren:

```
GetLight();
```

De method kan ingangsvariabelen nodig hebben en kan variabelen van het functieblok waaraan het gekoppeld zit gebruiken en aanpassen. Een method retourneert zichzelf, dat wil zeggen dat in het voorbeeld GetLight wordt geretouneert. Dit kan handig zijn als je buiten het functieblok de waarde die de method retourneert nodig hebt. Een method zal per functieblok opnieuw geprogrammeerd moeten worden, het type ingangsvariabele(n) en returntype staan vast. Een method kan dus in het ene functieblok een andere functie hebben dan in een ander functieblok.

```
METHOD GetLight : BOOL
VAR_INGANG
END_VAR
VAR
END_VAR

GetLight:=bLight;
```

Figuur 13: Method

4.1.5 Voor- en nadelen objectgeoriënteerd programmeren

Voordelen:

- Doordat er gebruik wordt gemaakt van een standaard definitie zal overal waar een method wordt aangeroepen hetzelfde type waardes worden geretourneerd en gebruikt. Hierdoor kan er ook ingesteld worden dat de ingangsvariabelen hetzelfde zijn en kunnen er dus verschillende functies geschreven worden die dezelfde type ingangsvariabele(n) en return types hebben.
- Door het gebruik van methods en functieblokken blijft de code van het hoofdprogramma kort en overzichtelijk.
- Door het gebruik van methods is het mogelijk om variabelen van een functieblok aan te passen vanuit het hoofdprogramma.

Nadelen:

- Het is relatief lastig om de gehele code te begrijpen en wat er wanneer gebeurt doordat er veel elementen gebruikt worden. Deze elementen hebben onderling een link of erven onderdelen van elkaar over.

Aanbeveling:

Bij grote projecten waar veel functieblokken gebruikt moeten worden met verschillende functies maar gelijke types ingangsvariabele(n) en returntype hebben, is het handig om objectgeoriënteerd te programmeren met CODESYS. Aangezien de variabele(n) uit de interfacemethods worden gehaald en dezelfde variabele(n) gebruikt worden. Dit komt doordat het functieblok deze over erft. Bij kleinere projecten of projecten waar weinig functieblokken gebruikt moeten worden die verschillende functies hanteren die dezelfde types ingangsvariabele(n) en returntype hebben is het niet aan te raden om gebruik te maken van objectgeoriënteerd programmeren. Dit komt doordat er dan voor (bijna) elke functie een nieuwe interface en interfacemethod gemaakt moet worden en dit het programma onnodig lastig maakt (voor de klant).

4.2 Functieblok

Een functieblok kan op verschillende manieren geprogrammeerd worden. Zoals uitgelegd in paragraaf 4.1 kan een functieblok gebruikt worden in combinatie met interfaces, interfacemethods en methods. De manier waarop Festo gebruik maakt van functieblokken, is door in het functieblok functies uit te voeren zonder interfaces, interfacemethods of methods. Zoals uitgelegd in hoofdstuk 3 wordt er bij het huidige functieblok dat gebruikt wordt door Festo de invoergegevens en uitvoergegevens gekoppeld aan de uitgang- en ingangsvariabelen. Het functieblok heeft verschillende functies die bepaald wordt door de invoergegevens. Hierdoor kan één functieblok een grote en ook meerdere taken uitvoeren of kan een combinatie van functieblokken grote of meerdere taken uitvoeren.

4.2.1 Voor- en nadelen gebruik functieblok

Voordelen:

- Er is één functieblok of een aantal, dat samen een of meerdere functies uitvoeren. Hierdoor is eenvoudig om oppervlakkig te begrijpen en/of uit te leggen wat er gebeurt.
- Doordat één functieblok meerdere functies kan uitvoeren is het eenvoudiger om een functieblok flexibel te maken, dan wanneer er gebruik wordt gemaakt van objectgeoriënteerd programmeren.

Nadelen:

- Bij grote projecten waar functieblokken met meerdere functies gebruikt moeten worden kan het onoverzichtelijk worden. Dit komt door de vele in- en uitgangen van de functieblokken.
- Doordat slechts 1 functieblok of een paar functieblokken samen een of meerder functies uitvoeren zijn er veel in- en uitgangen op de functieblokken. Hierdoor kan het lastig zijn om precies te snappen en/of uit te leggen hoe het functieblok werkt. Dit geldt vooral bij grote projecten waar veel functies uitgevoerd moeten worden.

Aanbeveling:

Bij kleine projecten waar een paar functies uitgevoerd moeten worden is het handig om een functieblok te gebruiken. Het blijft dan overzichtelijk en is het duidelijk welke gegevens waar gebruikt worden. Bij grote projecten waar veel functieblokken met meerdere functies gebruikt worden kan het lastig zijn om het overzicht te houden. Hierdoor is het aan te raden om alleen functieblokken te gebruiken bij kleinere projecten om het overzichtelijk te houden en dit bij grotere projecten niet te doen i.v.m. de vele in- en uitgangen.

5. Nieuwe manier van aansturen van e-drives

Voor het aansturen van e-drives wordt er door Festo gebruik gemaakt van een functieblok, ook wel bekend als het CTRL functieblok. Bij de nieuwe manier van het aansturen van e-drives is het huidige functieblok aangepast. Het vernieuwde functieblok zal dezelfde functie als de oude moeten hebben op een aantal wijzigingen na. Het oude functieblok moest in verschillende modes kunnen werken, hierdoor waren er dus veel ingangen die bijna niet gebruikt werden. Het nieuwe functieblok zal gebruikt worden als er gewerkt wordt in de Direct mode Position Control. Er zijn vanuit Festo een aantal eisen waaraan de nieuwe manier van aansturen van e-drives moet voldoen:

- Minder ingangen op het functieblok.
- Het functieblok moet eenvoudig in het gebruik zijn voor de klanten van Festo.
- Er zullen functieblokken geschreven worden die opstellingen aan kunnen sturen die in 1D en 2D kunnen bewegen.
- Het functieblok moet uitbreidbaar zijn door onder andere verschillende functieblokken aan te kunnen sluiten.
- Er zal met een objectgeoriënteerd oog naar de functieblokken gekeken moeten worden en hoe deze op elkaar aansluiten.

Daarnaast zijn er een aantal wensen vanuit Festo:

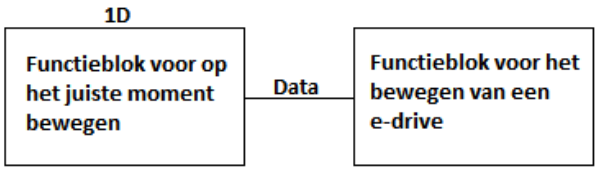
- Het startbit automatisch laag maken.
- Het functieblok moet zo geprogrammeerd worden dat de tijd die nodig is om een positie in te laden en hier naartoe bewegen zo kort mogelijk is.

Om de nieuwe manier van aansturen van e-drives aan deze eisen en wensen te laten voldoen is er een concept bedacht. Dit concept wordt uitgelegd in paragraaf 5.1.

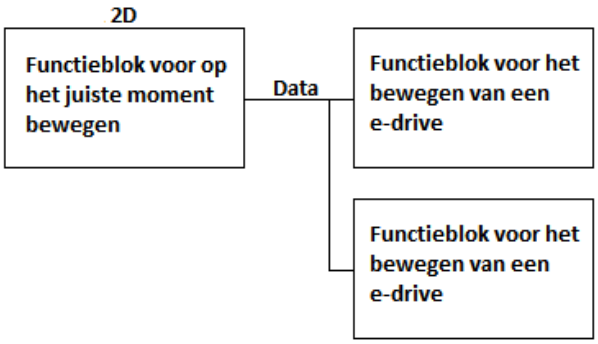
Het vernieuwde functieblok zal eenvoudig in gebruik moeten zijn voor de klanten van Festo. Om problemen met de huidige functieblokken te identificeren, en de wensen van de klanten beter in beeld te krijgen, is ervoor gekozen om contact te zoeken met oude klanten van Festo. Tijdens dit contact zijn er een aantal problemen naar voren gekomen die relevant waren voor de nieuwe functieblokken.
Deze problemen kwamen grotendeels overeen met de eisen en wensen vanuit Festo. Op basis van het contact met klanten en de wensen en eisen van Festo is het vernieuwde functieblok gemaakt.

5.1 Het concept

Voor de nieuwe manier van aansturen e-drives is er met een objectgeoriënteerd oog naar de mogelijkheden gekeken. Hierdoor is er besloten om één functieblok te maken voor het bewegen van een e-drive en één functieblok om ervoor te zorgen dat de e-drive op het juiste moment naar de juiste positie zal bewegen. Deze functieblokken zullen onderling data uitwisselen. Het functieblok voor het bewegen van een e-drive zal zo gemaakt worden dat het voor zowel 1 dimensionale opstellingen als 2 dimensionale opstellingen gebruikt kan worden. Er zullen verschillende functieblokken komen voor op het juiste moment bewegen voor 1 dimensionale opstellingen (zie Figuur 14) en 2 dimensionale opstellingen (zie Figuur 15).



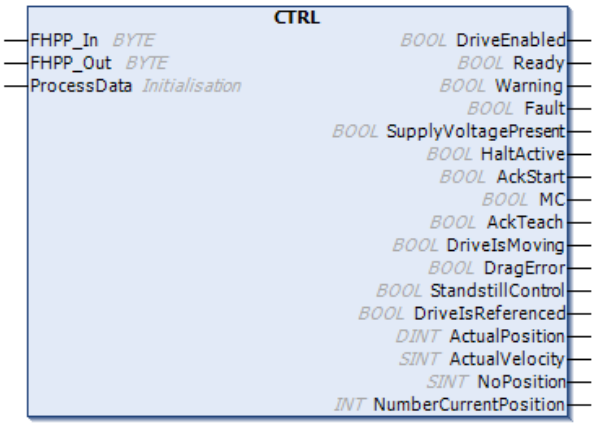
Figuur 14: Concept 1D aansturing



Figuur 15: Concept 2D aansturing

Door de functieblokken op deze manier te gebruiken kan het functieblok voor het bewegen van een e-drive gebruikt worden voor elke opstellingen. De functieblokken voor op het juiste moment bewegen zijn eenvoudig uit te breiden naar functieblokken voor opstellingen van 3 dimensionaal to 6 dimensionaal.

5.2 Uitleg functieblok voor het bewegen van een e-drive



Figuur 16: Vernieuwd CTRL functieblok

Het vernieuwde functieblok voor het bewegen van een e-drive moet dezelfde functie hebben als het huidige functieblok als er in de Direct mode Position Control wordt gewerkt. Daarnaast zal het aan de eerder beschreven eisen moeten voldoen. Omdat dit functieblok data uit zal gaan wisselen met een ander functieblok zal deze eenvoudig en overzichtelijk moeten zijn. Daarom is er begonnen met het verwijderen van de ingangen die niet gebruikt werden in de Direct mode Position Control. Hierdoor vielen een aantal ingangen weg. Desondanks bleven er teveel ingangen over. De volgende stap was het samenvoegen van de overgebleven ingangen tot één ingang. Hoe dit is gedaan wordt toegelicht in paragraaf 5.2.1. De gedachte achter het samenvoegen van de ingangen naar één ingang is uitbreidbaarheid. Doordat er slechts één ingang variabel is kan deze relatief eenvoudig aangestuurd worden door een ander functieblok, die door de gebruiker van het vernieuwde functieblok zelf geschreven kan worden of door gebruik te maken van het functieblok One_Dimensional of Two_Dimensional. Het functieblok One_Dimensional wordt uitgelegd in hoofdstuk 6. Het functieblok Two_Dimensional wordt uitgelegd in hoofdstuk 7. Doordat er een aantal ingangen weg zijn gehaald verdwijnen er ook een aantal geprogrammeerde functies van het huidige functieblok. Hoe de programmering van het nieuwe functiebok werkt wordt uitgelegd in paragraaf 5.2.2.

5.2.1 Nieuwe manier van inlezen van de ingangen

In het vernieuwde functieblok is er gekeken of het mogelijk is om verschillende ingangen samen te voegen. Er werd gedacht aan de manier waarop de ingang en uitvoergegevens van het huidige functieblok wordt in- en uitgelezen. Dit gebeurt door middel van een pointer naar een array van bytes. Na het onderzoeken en testen van het gebruik van een array van het bytes kwam de code tot stand, te zien in Figuur 17. Dit werkte, het enige nadeel was dat het niet duidelijk was welke waarde bij welke variabele hoorde. Omdat er zoveel mogelijk ingangen samengevoegd moesten worden werd er ook gekeken of het mogelijk was om ook de ingestelde positie en snelheid via de array van bytes in te lezen in het functieblok. Dit bleek niet mogelijk aangezien deze niet van het type byte zijn. Hierdoor is er verder gezocht naar andere mogelijkheden om meerdere variabelen van verschillende types samen te voegen tot één variabele. Hier kwam een struct uit, aangezien het hierbij duidelijker is welke variabelen van het functieblok er bij welke variabelen van de struct hoort (zie Figuur 18). Dit komt doordat de variabelen van de struct een naam kunnen krijgen.

```
ProcessData[0].0              := TRUE;
ProcessData[0].1              := TRUE;
.
.
.
```

Figuur 17: Initialisatie met een array

```
ProcessData.EnableDrive       := TRUE;
ProcessData.Stop              := TRUE;
.
.
.
```

Figuur 18: Initialisatie met een struct

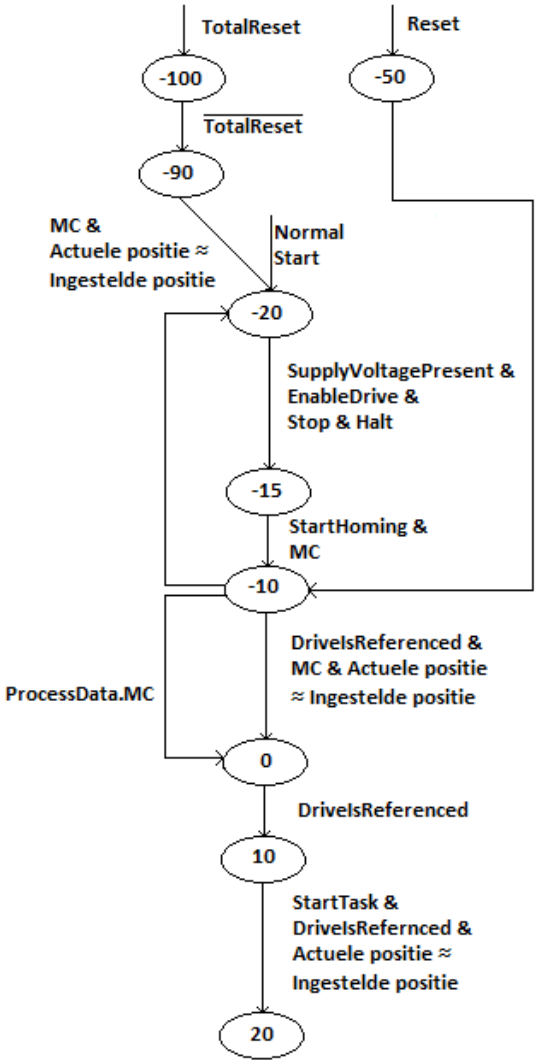
Er is dus gekozen om een struct te gebruiken, voor zowel de overzichtelijkheid en om de verschillende soorten variabelen in één variabele te gebruiken. Er kan in een struct zowel variabelen van het type integer als bijvoorbeeld variabelen van het type bool gebruikt worden. Met andere woorden alle type variabelen kunnen in een struct gebruikt worden. In de struct die gebruikt wordt voor de ingangsvariabelen zijn de volgende variabelen gezet:

- EnableDrive (BOOL)
- Stop (BOOL)
- Brake (BOOL)
- Halt (BOOL)
- StartTask (BOOL)
- StartHoming (BOOL)
- AbsoluteRelative (BOOL)
- Reset (BOOL)
- TotalReset (BOOL)
- HC (BOOL)
- MC (BOOL)
- Positions (ARRAY [0..100] OF DINT)
- Velocities (ARRAY [0..100] OF USINT)
- Mode (ARRAY [0..100] OF BOOL)
- NumberCurrentPosition (INT)
- ActualPosition (DINT)
- SupplyVoltagePresent (BOOL)

De variabelen EnableDrive, Stop, Brake, Halt, StartTask, StartHoming en AbsoluteRelative zijn de ingangsvariabelen die in het huidige functieblok gebruikt worden. Deze variabelen hebben dus nog dezelfde functie. De variabelen Reset, TotalReset HC, MC, NumberCurrentPosition, ActualPosition, Positions, Velocities en Mode zijn variabelen die in het huidige functieblok nog niet gebruikt werden. De variabele Reset wordt gebruikt om te resetten en vervolgens verder te kunnen gaan met de bewegingen. Deze valt dus te vergelijken met de ResetFault van het huidige functieblok.
TotalReset wordt gebruikt om een complete reset te geven. Hierdoor wordt er naar de positie 0 bewogen en zal er opnieuw geïnitialiseerd moeten worden, waarna vervolgens naar de 1^{e} positie bewogen kan worden.
De variabelen NumberCurrentPosition en ActualPosition worden gebruikt voor een terugkoppeling via de struct naar One_Dimensional of naar het functieblok dat door de gebruiker geprogrammeerd is.
De variabelen Positions, Velocities en Mode worden gebruikt om de volgende positie te bepalen, met welke snelheid daar naartoe moet worden bewogen en of er automatisch naar de volgende positie met de volgende snelheid moet worden bewogen. Door de variabele Mode kunnen er dus bewegingen in geprogrammeerd worden. Dit is vooral handig bij 2 tot 6 dimensionale opstellingen.
De variabele SupplyVoltagePresent wordt gebruikt om te indexeren of er een CANOpen connectie is en dat de juiste voltage is aangesloten.

5.2.2 Uitleg programma

Door het gebruik van een struct is er in het vernieuwde functieblok veel veranderd ten opzichte van het huidige functieblok. Er wordt nu gebruik gemaakt van slechts 3 ingangen.
Er wordt in de nieuwe situatie begonnen met het uitlezen van de struct en de variabelen van deze struct in de juiste variabelen van het functieblok te zetten. Dit is gedaan om zo het verschil met het huidige functieblok zo klein mogelijk te houden. Als de struct is uitgelezen en in de juiste variabelen is gezet wordt de invoergegevens uitgelezen. Dit vindt plaats op dezelfde manier als in het huidige functieblok, alleen is de variabele SDIR eruit gehaald aangezien deze gebruikt werd om de mode te selecteren. In het vernieuwde functieblok wordt er altijd in de Direct mode Position Control gewerkt.
Zodra de invoergegevens zijn uitgelezen zullen deze gegevens gebruikt worden om de juiste uitvoergegevens te krijgen. Hetgeen dat meteen opvalt, is dat er gebruik gemaakt wordt van een state diagram. Het is hier mogelijk om gebruik te maken van een if-then-else (wat in de states zelf ook gebruikt wordt). Er is gekozen om gebruik te maken van een state diagram omdat dit een algemene afspraak is bij Festo. Daarnaast is het vernieuwde functieblok geschreven om het gebruik eenvoudiger te maken en met een state diagram is het voor de meeste personen eenvoudiger om in gebruik te nemen en/of uit te leggen. Bij het maken van de states is er gekeken wanneer welke bits hoog of laag moeten zijn, hier zijn de states uit gekomen te zien in Figuur 19. Hierin is ook te zien aan welke voorwaarden voldaan moet worden voordat er van de huidige state naar de volgende state te gaan. In deze states worden ook variabelen aangepast. Welke variabelen er worden aangepast in welke state is te zien in Bijlage V: Variabelen verandering per state vernieuwd CTRL functieblok.
Om te zien in welke state er moet worden begonnen wordt er eerst gekeken of Reset en/of TotalReset hoog zijn. Als Reset hoog is wordt er in State -50 begonnen. Hier worden alle bits weer goed gezet om te kunnen bewegen. Vervolgens zal er weer verder worden gegaan met bewegen.
Als TotalReset hoog is wordt er in state -100 begonnen. Hier wordt er naar de positie 0 van de e-drive bewogen, waarna vervolgens alle bits laag gemaakt worden. Alle bits zullen dan opnieuw ingesteld moeten worden, daarnaast zal er ook opnieuw gehomed moeten worden.
Er wordt in state -20 begonnen als zowel Reset als TotalReset laag zijn.
Zoals in Bijlage V: Variabelen verandering per state vernieuwd CTRL functieblok en Figuur 19 beschreven, zal er begonnen worden in state -20, mits Reset en TotalReset laag zijn. In state -20 zal er gekeken of de juiste voltage aanwezig is en of de juiste bits hoog staan om te kunnen bewegen. Zodra dit het geval is zal er gewacht worden totdat StartHoming hoog wordt gemaakt om te kunnen homen, dit gebeurt in state -15. Tijdens het homen wordt er met behulp van de functie MaxPositions berekend hoeveel posities en snelheden er in de array staan. De werking van MaxPositions wordt uitgelegd in hoofdstuk 0. In de state -10 wordt er gekeken of er gehomed is.



Figuur 19: State Diagram vernieuwde CTRL functieblok

Zodra er gehomed is zal er gekeken worden of StartTask hoog is en of de bits die nodig zijn om te kunnen bewegen goed staan, dit gebeurd in de states 0 en 10.
Als dit het geval is zal er naar de ingestelde positie bewogen worden en zal met behulp van de if-then-else in state 20 (zie Bijlage II: Code vernieuwd CTRL functieblok) de volgende positie in de array bepaald worden. In deze state wordt er gekeken of het bewegen naar de ingestelde positie klaar is. De volgende positie wordt pas bepaald als er voorgaande beweging gedaan is. Er is voor gekozen om dit pas te doen als de voorgaande beweging is afgehandeld en niet als deze is ingezet om ervoor te zorgen dat er geen positie wordt over geslagen als er een error komt en er gereset moet worden.

Zodra het bewegen naar de ingestelde positie is afgehandeld wordt de waarde van de volgende positie en snelheid in SetValuePosition en SetValueVelocity gezet en zal er naar de vorige state worden gegaan.
Er zal dus tussen de states 10 en 20 gewisseld worden, waarbij gekeken wordt of er naar de ingestelde positie bewogen moet worden en als dat gebeurd is zal de volgende positie en snelheid in SetValuePosition en SetValueVelocity gezet worden.
Als laatste worden de uitvoergegevens ingelezen op dezelfde manier als bij het huidige functieblok, alleen wordt er niet gekeken welke informatie er in de bytes 4 t/m 8 gezet moet worden. Dit is al bekend, dit zal altijd de ingestelde positie en snelheid zijn.

Voordelen:

- 3 ingangen, hierdoor is het functieblok overzichtelijker.
- Door middel van een state diagram is de code overzichtelijker en zal deze dus eenvoudiger begrepen worden.
- Het functieblok is uitbreidbaar doordat andere functieblokken er op aangesloten kunnen worden, zoals het functieblok One_Dimensional.
- De struct zorgt ervoor dat het duidelijk is welke variabele aangepast wordt.

Nadelen:

- Bepaalde onderdelen, waaronder de array met posities en velocities moeten op een specifieke manier geïnitialiseerd worden.

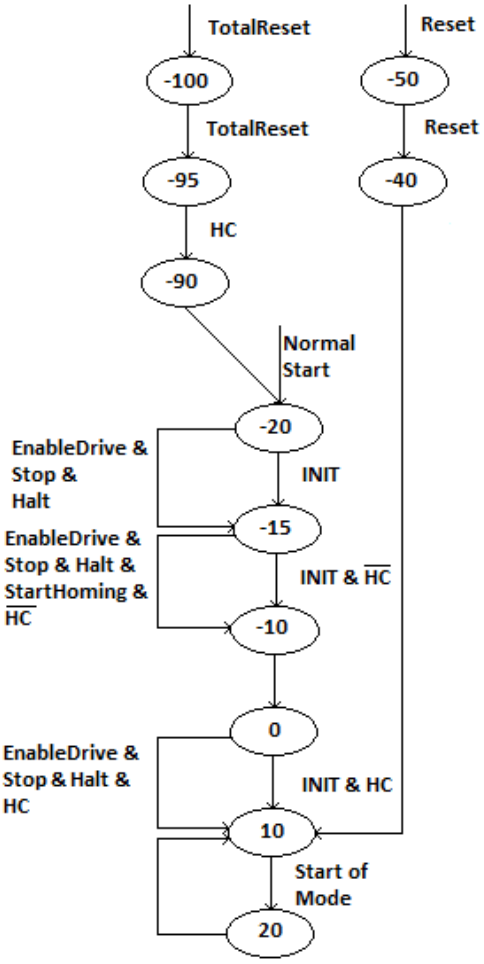
6. Aansturen 1 dimensionale opstellingen

Voor het aansturen van een 1 dimensionale opstelling wordt er gebruik gemaakt van het vernieuwde CTRL functieblok en van een functieblok voor op het juiste moment bewegen. Dit functieblok heeft One_Dimensional en kan alleen gebruikt worden voor het aansturen van één e-drive met behulp van het vernieuwde CTRL functieblok. In het functieblok One_Dimensional is gebruik gemaakt van states, in Figuur 20 is te zien aan welke voorwaarden voldaan moet worden om naar de volgende state te gaan. In dit functieblok moest er geregeld worden wanneer er bewogen moet worden. Er moet hier dus gekeken en geregeld worden welke bits er hoog of laag moeten zijn op bepaalde momenten. Het functieblok One_Dimensional heeft 6 ingangen (zie Tabel 2). De code van het functieblok One_Dimensional is te vinden in Bijlage III: Code One_Dimensional

Tabel 2: Ingangen One_Dimensional

Ingang	Functie
Init	Initialiseren, de bits zullen goed worden gezet en er zal gehomed worden.
Start	Startbit, er zal naar de volgende positie bewogen worden.
Coordinates	Array met coördinaten en bit voor het automatisch naar de volgende positie bewegen.
Velocities	Array met velocities.
Reset	Reset waarbij alle bits weer goed worden gezet en er verder kan worden gegaan met bewegen.
TotalReset	Complete reset. Er zal naar positie 0 van de e-drive worden bewogen en alle bits worden laag gemaakt. Er zal opnieuw geïnitialiseerd moeten worden.

Als er normaal wordt begonnen, wordt er in de state -20 gewacht totdat de ingang Init hoog is. Wanneer dit het geval is worden de bits om te kunnen bewegen goed gezet en wordt er naar de volgende state (-15) gegaan. Welke variabelen er per state aangepast worden is te vinden in Bijlage VI: Verandering variabelen per state One_Dimensional. In deze state wordt StartHoming hoog gemaakt zodat er gehomed wordt. Dit gebeurt in een aparte state omdat de bits die nodig zijn om te kunnen bewegen minimaal één cycle goed moeten staan voordat StartHoming hoog wordt. Als dit niet het geval is zal er niet gehomed worden en moet StartHoming eerst weer laag en hoog gemaakt worden voordat er gehomed wordt. Vervolgens wordt er naar state -10 gegaan, in deze state worden de arrays met posities en velocities geconverteerd naar een standaard type waardoor deze in het CTRL functieblok gebruikt kunnen worden. Dit gebeurt met de method Convert en de functies X_Coordinate, X_Velocities en Mode. Er is gekozen om dit in deze state toen doen zodat dit tijdens het homen gebeurt waardoor er optimaal van de tijd gebruik wordt gemaakt.
Zodra er gehomed is wordt er gewacht totdat de ingang Start hoog is. Als Start hoog is wordt StartTask in de struct hoog gemaakt zodat de e-drive zal gaan bewegen met behulp van het CTRL functieblok. Als laatste zal er gekeken worden of de beweging voltooid is. Dit gebeurt in state 20 met behulp van de struct.



Figuur 20: State diagram

De Reset en TotalReset zijn toegevoegd om volledig te kunnen resetten en om te resetten om vervolgens verder te gaan met de bewegingen.
Als TotalReset hoog is wordt er naar de positie 0 van de e-drive bewogen en worden alle bits laag gemaakt en zal er weer opnieuw geïnitialiseerd moeten worden.
Als Reset hoog wordt gemaakt zullen alle bits weer goed gezet worden om te kunnen bewegen en worden de faults afgehandeld. Hierna zal er verder worden gegaan met de beweging. Hier moest dus goed gekeken worden op welk moment er wordt gereset. Als deze midden in een beweging zit zal deze beweging afgemaakt moeten worden en als de beweging al afgehandeld was moet er gewacht worden totdat Start hoog wordt of als er automatisch naar de volgende positie bewogen moet worden.

Naast dat het mogelijk is om de bewegingen aan te sturen met behulp van de ingangen INIT en Start is het ook mogelijk om dit te doen door de variabelen van de struct te benaderen. Het is dus mogelijk om zelf alle bits goed te zetten en te homen en vervolgens met de ingang Start naar de ingestelde positie bewegen.

7. Aansturen 2 dimensionale opstellingen

Voor het aansturen van 2 dimensionale opstellingen zal er gebruik worden gemaakt van twee vernieuwde CTRL functieblokken, één functieblok voor de aansturing van de x-as (horizontale bewegingen) en één functieblok voor de aansturing van de y-as (verticale bewegingen). Daarnaast wordt er een functieblok gebruikt voor de juiste as op het juiste moment bewegen. Dit functieblok heet Two_Dimensional en kan alleen gebruikt worden voor het aansturen van twee drives met behulp van twee vernieuwde CTRL functieblokken zoals eerder genoemd. Het functieblok Two_Dimensional was lastiger te bedenken en maken dan One_Dimensional aangezien er hier gebruik wordt gemaakt van e-drives. Two_Dimensional is gebaseerd op One_Dimensional echter zullen sommige dingen dubbel moeten gebeuren in verband met de twee e-drives. In Two_Dimensional wordt er gebruik gemaakt van states, en is te zien aan welke voorwaarden voldaan moeten worden om naar de volgende state te gaan. Dit functieblok heeft 6 ingangen (zie Tabel 3). De code van het functieblok Two_Dimensional is te vinden inBijlage IV: Code Two_Dimensional.

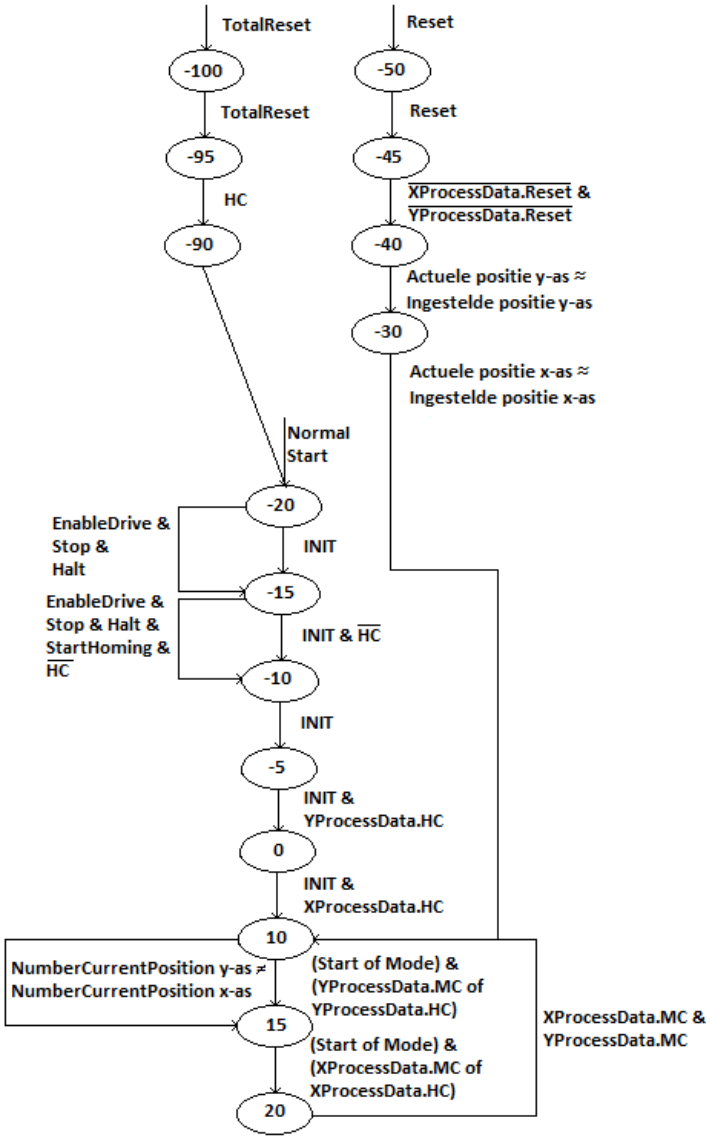
Tabel 3: Ingangen Two_Dimensional

Ingang	Functie
Init	Initialiseren, de bits zullen goed worden gezet en er zal gehomed worden.
Start	Startbit, er zal naar de volgende positie bewogen worden.
Coordinates	Array met coördinaten en bit voor het automatisch naar de volgende positie bewegen.
Velocities	Array met velocities.
Reset	Reset waarbij alle bits weer goed worden gezet en er verder kan worden gegaan met bewegen.
TotalReset	Complete reset. Er zal naar positie 0 van de e-drive worden bewogen en alle bits worden laag gemaakt. Er zal opnieuw geïnitialiseerd moeten worden.

Als er normaal wordt gestart, wordt er in de state -20 gestart. In deze state worden alle bits voor zowel de x-as als voor de y-as goed gezet zodra de ingang Init hoog wordt gemaakt. Welke variabelen er aangepast worden per state is te vinden in Bijlage VII: Verandering variabelen per state Two_Dimensional. Zodra de bits goed zijn gezet wordt er in state -15 de StartHoming van de y-as hoog gemaakt, mits de ingang Init hoog is. Vervolgens zal er in de state -10 alle posities en snelheden voor zowel de x-as als de y-as berekend worden met behulp van de method Convert en de functies X_Coordinates, X_Velocities, Mode, Y_Coordinates en Y_Velocities. Deze functies en method worden uitgelegd in hoofdstuk 0.
Nadat deze posities en snelheden berekend zijn, wordt er gewacht totdat de y-as gehomed is en de ingang Init hoog is voordat de StartHoming van de x-as hoog wordt gemaakt. Dit gebeurt in de state -5.
In de state 0 wordt er gekeken of het homen van de x-as voltooid is. Vervolgens zal er in de state 10 gekeken worden of er bewogen moet worden. Dit wordt gedaan door te kijken of de ingang Start hoog is of dat Mode hoog is. StartTask van de y-as zal dan hoog gemaakt worden, de y-as zal altijd als eerste bewegen aangezien deze tegen onderdelen in de omgeving aan kan komen. Vervolgens zal er in de state 15 gekeken worden of de beweging van de y-as voltooid is en of Start of Mode hoog is. Als er aan deze voorwaarden voldaan wordt zal StartTask van de x-as hoog gemaakt worden. In state 20 wordt er gekeken of zowel de beweging van de y-as als de beweging van de x-as voltooid zijn voordat er weer naar state 10 zal worden gegaan. In state 20 wordt Start laag gemaakt als de beweging van beide assen voltooid is. Doordat Start hier laag gemaakt wordt zal de gebruiker Start slechts 1x hoog moeten maken als er een beweging op zowel de y-as als op de x-as moet plaats vinden.

Net als in het functieblok One_Dimensional is er de optie om te resetten en vervolgens weer door te bewegen en de optie om volledig te resetten toe gevoegd.
TotalReset werkte bijna hetzelfde als bij One_Dimensional. In het functieblok Two_Dimensional zal er eerst met de y-as naar positie 0 van de e-drive worden bewogen. Zodra dat afgehandeld is zal hetzelfde gebeuren voor de x-as. Als de drive van de x-as op positie 0 is zullen alle bits laag gemaakt worden en moet er opnieuw geïnitialiseerd worden.
Reset was echter een stuk ingewikkelder dan bij One_Dimensional. Dit kwam doordat het mogelijk is dat de y-as of de x-as niet op de ingestelde positie staat zodra er gereset wordt. Hierdoor zal er in state -45 gekeken worden of de y-as op de ingestelde positie is. Is dit niet het geval zal deze eerst naar de juiste positie bewegen. Vervolgens zal er in state -40 gekeken worden of de x-as op de ingestelde positie is, dit gebeurd pas als de y-as op de ingestelde positie is. Als de x-as niet op de ingestelde positie is zal deze naar de juiste positie bewegen. Vervolgens wordt er in state -30 gekeken of de x-as op de ingestelde positie is. Is dit het geval zal er verder worden gegaan met bewegen, dit gebeurt doordat er naar state 10 word gegaan.

Om het nog eenvoudiger voor de klant te maken is de functie toegevoegd waarmee het zichtbaar wordt wat de gebruiker moet doen. Er komt dan onder andere te staan dat er gewacht wordt op een CANopen connectie of dat de drive naar de ingestelde coördinaten beweegt (de waardes van deze coördinaten worden weergegeven). Het is ook mogelijk dat de ingangsvariabelen Start hoog gemaakt moet worden. Er wordt dan weergegeven “Waiting for Start”. Hoe dit precies is gedaan is terug te vinden in Bijlage IV: Code Two_Dimensional.



Figuur 21: State diagram Two_Dimensional

8. Extra functies en methods

Tijdens het analyseren van het huidige systeem, het onderzoeken van de mogelijkheden met CODESYS en het schrijven van het nieuwe functieblok zijn er een aantal functies en methods geschreven. Deze functies en methods zijn geschreven om het gebruik van het functieblok eenvoudiger te maken. De methods worden op een andere manier gebruikt dan uitgelegd in paragraaf 4.1. Deze methods zijn specifiek voor dat functieblok. Daarnaast zijn er een aantal handige functies gevonden die gebruikt kunnen worden bij toekomstige projecten en problemen. Al deze gevonden en geschreven functies en methods worden in dit hoofdstuk uitgelegd.

8.1 MaxPositions

MaxPositions is een functie die tot stand is gekomen bij het bedenken en maken van het functieblok One_Dimensional. Deze functie was gemaakt om te berekenen hoeveel posities en velocities er in de arrays staan. Deze functie is geïmplementeerd in het functieblok CTRL. De functie ziet er als volgt uit:

```
FUNCTION MaxPositions : INT
VAR_INGANG
      ARRAY1                              : ARRAY [0..30] OF DINT;
      ARRAY2                              : ARRAY [0..30] OF USINT;
END_VAR
VAR
      X                                   : INT;
END_VAR
```

```
WHILE X<100 DO
      IF NOT ((ARRAY1[X] = 0) AND (ARRAY2[X] = 0))THEN
            MaxPositions                  := X;

      END_IF
      X                                   := X + 1;
END_WHILE
```

Figuur 22: Code functie MaxPositions

Deze functie kijkt voor elke waarde in de arrays met posities en snelheden of deze beide gelijk zijn aan 0. Als dit het geval is betekend het dat er geen positie en/of snelheid op die plaats in de array staat. Je kunt dit aannemen aangezien er bij een snelheid met een waarde 0 niets zal bewegen. Het is onmogelijk om met 0% van het maximale te bewegen. Daarnaast wordt de array automatisch met de waarde 0 gevuld. MaxPositions wordt geretourneerd met daarin de waarde van de het aantal posities en velocities.

8.2 Mode

Mode is een functie die tot stand is gekomen bij het bedenken en maken van het functieblok One_Dimension. Deze functie haalt de laatste waarde uit de ingang array met coördinaten en zet deze in een array van het type bool. Deze functie kan in elk functieblok gebruikt worden, deze wordt dus in zowel het functieblok One_Dimensional gebruikt als in Two_Dimensional.

```
FUNCTION Mode : ARRAY [0..100] of BOOL;
VAR_INGANG
      ModeSelect                          : ARRAY [0..100] OF
Six_Dimensions;
END_VAR
VAR
      NoPosition                          : INT;
END_VAR
```

```
WHILE NoPosition < 100 DO
      IF ModeSelect[NoPosition][6] = 1 THEN
            Mode[Noposition]        := TRUE;
      ELSIF ModeSelect[NoPosition][6] = 0 THEN
            Mode[Noposition]        := FALSE;
      END_IF
      NoPosition                          := NoPosition + 1;
END_WHILE
```

Figuur 23: Code functie Mode

8.3 X_Coordinates

X_Coordinate is tot stand gekomen bij het bedenken en schrijven van het functieblok One_Dimensional. Deze functie haalt de eerste waardes uit de array van het type Six_Dimensions die de ingevulde posities bevat en stopt deze in een array van het type DINT. Dit wordt gedaan met het oog op de uitbreidbaarheid van het systeem. Deze functie zal dus gebruikt kunnen worden in opstellingen die 1 dimensionaal kunnen bewegen tot opstellingen die 6 dimensionaal kunnen bewegen.

```
FUNCTION X_Coordinates : ARRAY [0..100] OF DINT;
VAR_INGANG
      Coordinates                   : ARRAY [0..100] OF Six_Dimensions;
END_VAR
VAR
      NoPosition                    : INT;
END_VAR
```

```
WHILE NoPosition < 100 DO
      X_Coordinates[NoPosition]           := Coordinates[NoPosition][0];
      NoPosition                    := NoPosition + 1;
END_WHILE
```

Figuur 24: Code functie X_Coordinates

8.4 X_Velocities

X_Velocities is tot stand gekomen bij het bedenken en schrijven van het functieblok One_Dimensional. Deze functie haalt de eerste waardes uit de array van het type Six_Dimensions die de ingevulde velocities bevat en stopt deze in een array van het type USINT. Dit wordt gedaan met het oog op de uitbreidbaarheid van het systeem. Deze functie zal dus gebruikt kunnen worden in opstellingen die 1 dimensionaal kunnen bewegen tot opstellingen die 6 dimensionaal kunnen bewegen.

```
FUNCTION X_Velocities : ARRAY [0..100] OF USINT;
VAR_INGANG
      Velocities                    : ARRAY [0..100] OF Six_Dimensions;
END_VAR
VAR
      NoPosition                    : INT;
END_VAR
---------------------------------------------------------------------------

WHILE NoPosition < 100 DO
      X_Velocities[NoPosition]            :=
DINT_TO_USINT(Velocities[NoPosition][0]);
      NoPosition                    := NoPosition + 1;
END_WHILE
```

Figuur 25: Code functie X_Velocities

8.5 Y_Coordinates

Y_Coordinates is tot stand gekomen bij het bedenken en schrijven van het functieblok Two_Dimensional. Deze functie haalt de tweede waarde uit de array van het type Six_Dimensions met de ingevulde posities van zowel de X-as als de Y-as en stopt deze in een array van het type DINT. Dit wordt gedaan met het oog op de uitbreidbaarheid van het systeem. Deze functie zal dus gebruikt kunnen worden in opstellingen die 2 dimensionaal kunnen bewegen tot opstellingen die 6 dimensionaal kunnen bewegen.

```
FUNCTION Y_Coordinates : ARRAY [0..100] OF DINT
VAR_INGANG
      Coordinates                   : ARRAY [0..100] OF Six_Dimensions;
END_VAR
VAR
      NoPosition                    : INT;
END_VAR
---------------------------------------------------------------------------

WHILE NoPosition < 100 DO
      Y_Coordinates[NoPosition]           := Coordinates[NoPosition][1];
      NoPosition                    := NoPosition + 1;
END_WHILE
```

Figuur 26: Code functie Y_Coordinates

8.6 Y_Velocities

Y_Velocities is tot stand gekomen bij het bedenken en schrijven van het functieblok Two_Dimensional. Deze functie haalt de tweede waarde uit de array van het type Six_Dimensions met de ingevulde velocities van zowel de X-as als de Y-as en stopt deze in een array van het type USINT. Dit wordt gedaan met het oog op de uitbreidbaarheid van het systeem. Deze functie zal dus gebruikt kunnen worden in opstellingen die 2 dimensionaal kunnen bewegen tot opstellingen die 6 dimensionaal kunnen bewegen.

```
FUNCTION Y_Velocities : ARRAY [0..100] OF USINT
VAR_INGANG
      Velocities                    : ARRAY [0..100] OF Six_Dimensions;
END_VAR
VAR
      NoPosition                    : INT;
END_VAR
```

```
WHILE NoPosition < 100 DO
      Y_Velocities[NoPosition]            :=
DINT_TO_USINT(Velocities[NoPosition][1]);
      NoPosition                    := NoPosition + 1;
END_WHILE
```

Figuur 27: Code functie Y_Velocities

8.7 SIZEOF

SIZEOF is een operator die gebruikt kan worden bij het automatisch bepalen van de lengte van de arrays met de posities en velocities. Met deze operator kan de grootte bepaald worden van elke datatype, het geeft het aantal bytes weer die nodig zijn om het type weer te geven. Om te berekenen hoeveel waardes een array heeft zal er gedeeld moeten worden door SIZEOF van het datatype waarmee de array gevuld is.
Stel de array POS heeft 5 waardes, dan wordt het SIZEOF(POS)/SIZEOF(DINT). Je krijgt dus het totale aantal bytes van de array (20) gedeeld door het aantal byte van DINT (4), je krijgt dus 20/4 en dat is 5.

9. Conclusie

Tijdens de afstudeerstage waren onderstaande probleemstellingen leidend:
Kan de aansturing van de e-drives eenvoudiger en voldoen aan de eisen vanuit Festo? En is de huidige manier van programmeren bij Festo de beste manier?

Kan de aansturing van de e-drives eenvoudiger en voldoen aan de eisen vanuit Festo?
Door het huidige systeem te analyseren en door te onderzoeken wat er mogelijk is met het programma CODESYS kon er antwoord komen op de gestelde vraag.
De eisen vanuit Festo zijn:

- Minder ingangen op het functieblok.
- Het functieblok moet eenvoudig in het gebruik zijn voor de klanten van Festo.
- Er zullen functieblokken geschreven worden die opstellingen aan kunnen sturen die 1 dimensionaal en 2 dimensionaal kunnen bewegen.
- Het functieblok moet uitbreidbaar zijn door onder andere verschillende functieblokken aan te kunnen sluiten.
- Er zal met een objectgeoriënteerd oog naar de functieblokken gekeken moeten worden en hoe deze op elkaar aansluiten.

De aansturing van de e-drives kan eenvoudiger en voldoen aan de eisen vanuit Festo. Door de meeste ingangen samen te voegen tot één ingang door middel van een struct zijn het aantal ingangen gereduceerd van 24 tot 3. Doordat er een ander functieblok (One_Dimensional of Two_Dimensional) gebruikt wordt om te bepalen wanneer er bewogen moet worden met de e-drive is de nieuwe manier van aansturen van de e-drives modulair. Door de combinatie van minder ingangen en de twee functieblokken is de nieuwe manier van aansturen van de e-drives eenvoudiger (voor de klant). Er hoeft nu slechts twee arrays in gevuld te worden met posities en velocities en moeten de twee functieblokken op elkaar worden aangesloten met behulp van de struct. Ook kan de klant per positie instellen of er automatisch naar de volgende positie bewogen moet worden. Als dit niet het geval is zal de klant zelf op start moeten drukken om een beweging plaats te laten vinden. Door gebruik te maken van de mogelijkheid om automatisch naar de volgende positie te bewegen kunnen er bewegingen in geprogrammeerd worden. Er kan hier gedacht worden aan een 2 dimensionale pick and place systemen waar gewacht moet worden totdat een sensor detecteert dat er een voorwerp ligt. Zodra deze gedetecteerd is kan er een beweging plaats gaan vinden.

Is de huidige manier van programmeren bij Festo de beste manier?
Om uit te zoeken of de huidige manier van programmeren bij Festo de beste manier is, is er gekeken naar de mogelijke manieren van programmeren bij het programma CODESYS. Hier is geconcludeerd dat objectgeoriënteerd programmeren handig is bij grote projecten waar veel functieblokken soortgelijke functies gebruiken waarvan de ingangen types en return type hetzelfde zijn. Aangezien dit bij Festo veelal niet het geval is, is het niet aan te raden om op deze manier te programmeren. De huidige manier van programmeren bij Festo gebeurt met behulp van functieblokken. Deze functieblokken kunnen één of meerdere taken uitvoeren. Dit is handig bij kleine projecten waar één of een aantal taken uitgevoerd moeten worden, wat veelal het geval is bij Festo. Hieruit valt te concluderen dat de huidige manier van programmeren bij Festo de beste manier is.

De nieuwe manier van aansturen e-drives

Het in gebruik nemen van de functieblokken is getest door een aantal programmeurs bij Festo. Hieruit bleek dat er in een aanzienlijk kortere periode een 1 dimensionale of 2 dimensionale opstelling geprogrammeerd kan worden. Bij de huidige manier van aansturen duurde het meestal tussen een halve en hele dag, terwijl dit met de nieuw manier van aansturen tussen 30 en 60 min duurt.

Doordat het vernieuwde CTRL functieblok op dezelfde manier werkt in een 1 dimensionale opstelling als in een 2 dimensionale opstelling kan er geconcludeerd worden dat dit een goede basis is voor het bewegen van e-drives. Er zijn nu functieblokken geprogrammeerd die 1 dimensionale en 2 dimensionale opstellingen kunnen aansturen, dit kan eenvoudig uitgebreid worden voor 3 dimensionale tot 6 dimensionale opstellingen.

10.Aanbevelingen

Naar aanleiding van het analyseren van het huidige systeem, het onderzoeken van de mogelijkheden van CODESYS en het bedenken en maken van de nieuwe manier van aansturen van e-drives zijn er een aantal onderdelen bedacht die in de toekomst gebruikt kunnen worden maar niet uitgewerkt. Deze worden in dit hoofdstuk besproken.

Objectgeoriënteerd programmeren

Uit het onderzoeken van de mogelijkheden van CODESYS is er geconstateerd dat objectgeoriënteerd programmeren vooral handig is bij grote projecten waar veelal de functies worden gebruikt met hetzelfde ingangen type en return type. Dit is dus niet aan te raden voor het gebruik bij Festo aangezien zij CODESYS gebruiken om kleinere specifiekere projecten te programmeren.

Functieblokken voor 3 tot 6 dimensionale opstellingen

De functieblokken die momenteel gemaakt zijn kunnen alleen gebruikt worden voor opstellingen die 1 dimensionaal en 2 dimensionaal kunnen bewegen. Dit is eenvoudig uitbreidbaar naar opstellingen die tot 6 dimensionaal kunnen bewegen. Dit kan op dezelfde manier als de functieblokken One_Dimensional en Two_Dimensional.

Nog eenvoudiger voor de klant

De nieuwe functieblokken kunnen nog eenvoudiger voor de klant worden gemaakt. Dit kan door een “Shell” over de twee functieblokken te maken. Hierdoor is de uitwisseling van data tussen de twee functieblokken verborgen voor de klant. Dit heeft de meeste invloed op een opstelling die 6 dimensionaal kan bewegen, hier zou er dan één functieblok gebruikt moeten worden in plaats van zeven.

Error indicatie

In de functieblokken One_Dimensional en Two_Dimensional zijn de ingangen Reset en TotalReset, deze worden gebruikt om te kunnen resetten. Reset om te resetten en vervolgens weer door gaan met de beweging. TotalReset om volledig te resetten door naar de positie 0 te bewegen en alle bits laag te maken, waarna weer opnieuw de juiste bits hoog moeten worden gemaakt en gehomed moet worden. Deze ingangen wil je pas gebruiken als er een error is opgetreden. Op dit moment is er nog geen indicatie van een error. Dit kan nog toegevoegd worden door middel van codes (dus op dezelfde manier als bij de motor controller). Je krijgt dan bijvoorbeeld: E11-1, dit betekent een error tijdens het homen (zie Motor Controller Manual[4]).

Bronnen

1. Handleiding CODESYS [PDF]
   Organisatie: 3S-Smart Software Solutions [www.codesys.com]
   http://www.wago.com/wagoweb/documentation/759/eng_manu/333/m07590333_00000000_1en.pdf

2. FHPP manual [PDF]
   Organisatie: Festo [www.festo.nl]
   http://www.festo.com/net/nl_nl/SupportPortal/Downloads/343073/319397/555696g1.pdf

3. PLC handleiding [PDF]
   Organisatie: Festo [www.festo.nl]
   http://www.festo.com/net/nl_nl/SupportPortal/Downloads/345362/335605/8036062g1.pdf

4. Motor Controller Manual [PDF]
   Organisatie: Festo [www.festo.nl]
   http://www.festo.com/net/nl_nl/SupportPortal/Downloads/345484/346048/8037610g1.pdf

5. Huidig CTRL functieblok handleiding [PDF]
   Organisatie: Festo [www.festo.nl]
   http://www.festo.com/net/nl_nl/SupportPortal/Downloads/357489/340846/Festo_Motion_Lib_V03p05.exe

6. CANopen [website]
   http://www.canopen.nl/canopen-intro/#more-649

Bijlage

Bijlage I: Code huidige manier van aansturen electromechanical drives

```
(*-------------------------------------------------------------
     initial block routine
-------------------------------------------------------------*)
pFHPP := ADR(FHPP_In);  // Pointer to ingang data array

(* ===== assignment of ingang data ===== *)

SCON := pFHPP^[0];      // SCON cast to its bits
DriveEnabled                 := SCON.0;
Ready                        := SCON.1;
Warning                      := SCON.2;
Fault                        := SCON.3;
SupplyVoltagePresent         := SCON.4;
ControlFCT_HMI               := SCON.5;
StateOperationModeB1         := SCON.6;
StateOperationModeB2         := SCON.7;

SPOS := pFHPP^[1];      // SPOS cast to its bits
HaltActive                   := SPOS.0;
AckStart                     := SPOS.1;
MC                           := SPOS.2;
AckTeach                     := SPOS.3;
DriveIsMoving                := SPOS.4;
DragError                    := SPOS.5;
StandstillControl            := SPOS.6;
DriveIsReferenced            := SPOS.7;

IF StateOperationModeB1 THEN  // ===== directmode active =====
 (* SDIR Byte_3 *)
  SDIR := pFHPP^[2];
   StateContinuousMode       := SDIR.3;
   StateControlModeB2        := SDIR.2;
   StateControlModeB1        := SDIR.1;
   ActualRecordNo            := 0;
   (* Istwert1 Byte_4 *)
  ACTUALVALUE1 := pFHPP^[3];
   RC1                       := FALSE;
   RCC                       := FALSE;
ELSE                          // ===== recordmode active =====
   (* Byte_3 = Record Number *)
   StateContinuousMode       := 0;
   StateControlModeB2        := 0;
   StateControlModeB1        := 0;
   ActualRecordNo            := BYTE_TO_USINT (pFHPP^[2]);

  ACTUALVALUE1 := pFHPP^[3];  // ==== record state Byte_4 ====
   RC1                       := ACTUALVALUE1.0;
   RCC                       := ACTUALVALUE1.1;
END_IF
```

```
(* =====    Actual Value2 from byte 5 .. 8 ===== *)
pByte := ADR(ACTUALVALUE2);
IF (FB_CFG AND 16#0001) = 1 THEN  (* High Byte First *)
  pByte^[0] := pFHPP^[7];
  pByte^[1] := pFHPP^[6];
  pByte^[2] := pFHPP^[5];
  pByte^[3] := pFHPP^[4];
ELSE
  pByte^[0] := pFHPP^[4];
  pByte^[1] := pFHPP^[5];
  pByte^[2] := pFHPP^[6];
  pByte^[3] := pFHPP^[7];
END_IF

(* Mounting operation mode state *)
StateOPM.0 := StateOperationModeB1;
StateOPM.1 := StateOperationModeB2;
StateOPM.2 := StateControlModeB1;
StateOPM.3 := StateControlModeB2;
StateOPM.4 := StateContinuousMode;

CASE StateOPM OF
  0, 1, 13, 17:
    //OPM_RECORD_SEL:      // ( 0) recordmode
    //OPM_DIRECT_POS:      // ( 1) directmode position control
    //OPM_OPTIMISED:       // (13) directmode position control energie
optimized
    //OPM_TRACKING:        // (17) directmode contiuousmode
    IF (StateOPM = 0) THEN
      ActualVelocity := 0;
    ELSE
      ActualVelocity  := BYTE_TO_SINT(ACTUALVALUE1);
            END_IF
    ActualForce     := 0;
    ActualRotRamp   := 0;
    ActualRotSpeed  := 0;
    ActualPosition  := ACTUALVALUE2;

  5: //OPM_DIRECT_TRQ:      // (5) directmode force control
    ActualVelocity  := 0;
    ActualForce     := BYTE_TO_SINT(ACTUALVALUE1);
    ActualRotRamp   := 0;
    ActualRotSpeed  := 0;
    ActualPosition  := ACTUALVALUE2;

  9: //OPM_DIRECT_VELO:     // (9) directmode rotation-speed-control
    ActualVelocity  := 0;
    ActualForce     := 0;
    ActualRotRamp   := BYTE_TO_SINT(ACTUALVALUE1);
    ActualRotSpeed  := ACTUALVALUE2;
    ActualPosition  := 0;
  ELSE
    ActualVelocity  := 0;
    ActualForce     := 0;
    ActualRotRamp   := 0;
    ActualRotSpeed  := 0;
    ActualPosition  := 0;
END_CASE
```

```
CASE OPM OF
0: //OPM_RECORD_SEL:      // (0) recordmode
SETVALUE1 := 0;    // Byte_4
SETVALUE2 := 0;    // Byte 5-8

1, 13, 17:
// OPM_DIRECT_POS:      // ( 1) directmode position control
// OPM_OPTIMISED:       // (13) directmode position control energie optimized
// OPM_TRACKING:        // (17) directmode contiuousmode
SETVALUE1 := SetValueVelocity;     // Byte_4
SETVALUE2 := SetValuePosition;     // Byte 5-8

5: //OPM_DIRECT_TRQ:      // (5) directmode force control
(* Byte_4 *)
SETVALUE1        := SetValueForceRamp;
(* Byte 5-8  *)
IF SetValueForce > 100 THEN
SETVALUE2 := 100;
ELSIF SetValueForce < (-100) THEN
SETVALUE2 := -100;
ELSE
SETVALUE2 := SetValueForce;
END_IF

9: //OPM_DIRECT_VELO:     // (9) directmode rotation-speed-control
SETVALUE1 := INT_TO_BYTE(SetValueRotRamp);  // Byte_4
SETVALUE2  := SetValueRotSpeed;             // Byte 5-8
END_CASE

(* Copy SETVALUE 1 to Output Pointer*)
pFHPP^[3] := SETVALUE1;

(* Copy SETVALUE 2 to Output Pointer*)
pByte := ADR(SETVALUE2);
IF (FB_CFG AND 16#0001) = 1 THEN  // High Byte First
pFHPP^[7] := pByte^[0];
pFHPP^[6] := pByte^[1];
pFHPP^[5] := pByte^[2];
pFHPP^[4] := pByte^[3];
ELSE
pFHPP^[4] := pByte^[0];
pFHPP^[5] := pByte^[1];
pFHPP^[6] := pByte^[2];
pFHPP^[7] := pByte^[3];
END_IF
(* ===== FB_END ===== *)
```

```
(*-------------------------------------------------------------------------
----------------*)
(* assignment of output data
*)
(*-------------------------------------------------------------------------
----------------*)
(* decoding of operation mode *)
OperationModeB1     := OPM.0;
OperationModeB2     := OPM.1;
ControlModeB1       := OPM.2;
ControlModeB2       := OPM.3;
ContinuousMode      := 0; (* Function not reachable *)

OPMString := GetOPMString(OPM);    // Set the OPM string

pFHPP := ADR(FHPP_Out);
(* ----- CCON     Byte_1 ----- *)
CCON.0    := EnableDrive;
CCON.1    := Stop;
CCON.2    := Brake;
CCON.3    := ResetFault;
CCON.4    := FALSE;
CCON.5    := HMIAccessLocked;
CCON.6    := OperationModeB1;
CCON.7    := OperationModeB2;
pFHPP^[0] := CCON;

(* ----- CPOS     Byte_2 ----- *)
CPOS.0    := Halt;
CPOS.1    := StartTask;
CPOS.2    := StartHoming;
CPOS.3    := JoggingPos;
CPOS.4    := JoggingNeg;
CPOS.5    := TeachActualValue;
CPOS.6    := ClearRemainingPosition;
CPOS.7    := FALSE;
pFHPP^[1] := CPOS;

(* >>>>     directmode  <<<<*)
IF OperationModeB1 THEN
  (* CDIR Byte_3 *)
  CDIR.0    := AbsoluteRelative;
  CDIR.1    := ControlModeB1;
  CDIR.2    := ControlModeB2;
  CDIR.3    := FALSE;
  CDIR.4    := FALSE;
  CDIR.5    := FALSE;
  CDIR.6    := FALSE;
  CDIR.7    := FALSE;
  pFHPP^[2] := CDIR;
ELSE                          (* >>> recordmode <<< *)
  pFHPP^[2] := USINT_TO_BYTE(RecordNo); // Satznr. Byte_3
END_IF
```

Bijlage II: Code vernieuwd CTRL functieblok

```
(*----- initial block routine -----*)
(*----- reading the data from the ingangtructure ----*)
EnableDrive                                 := ProcessData.EnableDrive;
Stop                                        := ProcessData.Stop;
Brake                                       := ProcessData.Brake;
ResetFault                                  := ProcessData.Reset;
Halt                                        := ProcessData.Halt;
StartTask                                   := ProcessData.StartTask;
StartHoming                                 := ProcessData.StartHoming;
AbsoluteRelative                            := ProcessData.AbsoluteRelative;
CoordinatesArray                            := ProcessData.Positions;
VelocitiesArray                             := ProcessData.Velocities;
SetValuePosition                            := CoordinatesArray[NoPosition];
SetValueVelocity                            := VelocitiesArray[NoPosition];
ProcessData.ActualPosition                  := ActualPosition;

pFHPP := ADR(FHPP_In);  // Pointer to ingang data array

(*----- assignment of ingang data -----*)
SCON                                        := pFHPP^[0];     // SCON cast
to its bits
DriveEnabled                                := SCON.0;
Ready                                       := SCON.1;
Warning                                     := SCON.2;
Fault                                       := SCON.3;
SupplyVoltagePresent                        := SCON.4;

SPOS                                        := pFHPP^[1];     // SPOS cast
to its bits
HaltActive                                  := SPOS.0;
AckStart                                    := SPOS.1;
MC                                          := SPOS.2;
AckTeach                                    := SPOS.3;
DriveIsMoving                               := SPOS.4;
DragError                                   := SPOS.5;
StandStillControl                           := SPOS.6;
DriveIsReferenced                           := SPOS.7;

(*----- Actual Position and Velocity -----*)
ActualVelocity                              := pFHPP^[3];     // Actual
Velocity

pByte                                       := ADR(ActualPosition);// Actual
Position
  pByte^[0]                                 := pFHPP^[4];
  pByte^[1]                                 := pFHPP^[5];
  pByte^[2]                                 := pFHPP^[6];
  pByte^[3]                                 := pFHPP^[7];
```

```
IF ProcessData.Reset    THEN
      State                                 := -50;
END_IF
IF   ProcessData.TotalReset THEN
      State                                 := -100;
END_IF


CASE State OF

      -100: // Total reset
            IF ProcessData.TotalReset THEN
                  ProcessData.TotalReset  := FALSE;
            ELSE
                  Position                := SetValuePosition;
                  Velocity                := SetValueVelocity;
                  State                   := -90;
            END_IF

      -90:
            IF MC AND ABS( ActualPosition - position < 4) THEN
                  ProcessData.HC          := TRUE;
                  StartTask               := FALSE;
                  State                   := -20;
            END_IF

      -50: // Reset
            IF NOT Fault THEN
                  ProcessData.Reset       := FALSE;
                  StartTask               := FALSE;
                  StartHoming             := FALSE;
                  ProcessData.StartTask   := FALSE;
                  State                   := -10;
            END_IF

      -20: // Init Drive
            IF SupplyVoltagePresent AND DriveEnabled AND Ready AND
HaltActive THEN
                  State                   := -15;
            END_IF
```

```
      -15: // Wait for homing
            IF StartHoming AND MC THEN
                  NoPosition              := 0;
                  State                   := -10;
            END_IF

      -10: // Check if homing is completed
            NoOfPositions           := MaxPositions(ProcessData.Positions,
ProcessData.Velocities);
            IF DriveIsReferenced AND MC AND ABS((ActualPosition -
      position) < 4)THEN
                  ProcessData.HC          := TRUE;
                  StartHoming             := FALSE;
                  ProcessData.StartHoming := FALSE;
                  State                   := 0;
            ELSIF ProcessData.HC THEN
                  State                   := 0;
            ELSE
                  State                   := -20;
            END_IF

      0: // Idle and check if drive is referenced
            IF DriveIsReferenced THEN
                  State                   := 10;
            END_IF

      10: // Get nextposition and velocity
            IF StartTask AND DriveIsReferenced AND ABS(ActualPosition -
Position) < 4 THEN
                  Position                := SetValuePosition;
                  Velocity                := SetValueVelocity;
                  MC                      := FALSE;
                  ProcessData.MC          := FALSE;
                  State                   := 20;
            END_IF

      20:   // Check if Motion is completed
            IF DriveIsReferenced AND ABS(ActualPosition - Position) < 50
THEN
                  IF NoPosition = NoOfPositions  THEN
                        NoPosition        := 0;
                        NumberCurrentPosition:= NoOfPositions + 1;
                  ELSE
                        NoPosition        := NoPosition + 1;
                        NumberCurrentPosition:= NoPosition;
                  END_IF

                  ProcessData.NumberCurrentPosition :=
NumberCurrentPosition;
                  ProcessData.StartTask   := FALSE;
                  StartTask               := FALSE;
                  ProcessData.MC          := TRUE;
                  State                   := 0;
                  END_IF
END_CASE
```

```
pFHPP := ADR(FHPP_Out);
(* ----- CCON     Byte_1 ----- *)
CCON.0                                      := EnableDrive;
CCON.1                                      := Stop;
CCON.2                                      := Brake;
CCON.3                                      := ResetFault;
CCON.4                                      := FALSE;
CCON.5                                      := FALSE;
CCON.6                                      := TRUE;
CCON.7                                      := FALSE;
pFHPP^[0]                                   := CCON;

(* ----- CPOS     hByte_2 ----- *)
CPOS                                        := 0;
CPOS.0                                      := Halt;
CPOS.1                                      := StartTask;
CPOS.2                                      := StartHoming;
pFHPP^[1]                                   := CPOS;

(* ----- CDIR    Byte_3 ----- *)
CDIR                                        := 0;
CDIR.0                                      := AbsoluteRelative;
pFHPP^[2]                                   := CDIR;

(* ----- Set Velocity and Position ----- *)
pFHPP^[3]                                   := Velocity;

pByte                                       := ADR(Position);
pFHPP^[4]                                   := pByte^[0];
pFHPP^[5]                                   := pByte^[1];
pFHPP^[6]                                   := pByte^[2];
pFHPP^[7]                                   := pByte^[3];
```

Bijlage III: Code One_Dimensional

```
IF reset THEN
      TempState                                   := State;
      State                                       := -50;
END_IF
IF TotalReset THEN
      State                                       := - 100;
END_IF


CASE STATE OF

      -100: // Total reset
            IF TotalReset THEN
                  ProcessData.StartTask           := FALSE;
                  HC                              := FALSE;
                  ProcessData.HC                  := FALSE;
                  ProcessData.TotalReset          := TRUE;
                  State                           := -95;
                  TotalReset                      := FALSE;
            ELSE
                  ProcessData.StartTask           := TRUE;
                  State                           := -95;
            END_IF

      -95: // Get position 0
                  ProcessData.Positions           := ERRORPOS;
                  ProcessData.Velocities          := ERRORVEL;
                  ProcessData.StartTask           := TRUE;
                  IF ProcessData.HC THEN
                        State                     := -90;
                  END_IF
      -90: // go to coordinate 0
            IF ProcessData.HC THEN
                  ProcessData.EnableDrive         := FALSE;
                  ProcessData.Stop                := FALSE;
                  ProcessData.Brake               := TRUE;
                  ProcessData.Halt                := FALSE;
                  ProcessData.StartTask           := FALSE;
                  ProcessData.StartHoming         := FALSE;
                  ProcessData.HC                  := FALSE;

                  Enabled                         := FALSE;
                  HC                              := FALSE;
                        State                     := -20;
            END_IF
```

```
-50: //reset
      IF Reset THEN
            ProcessData.StartTask               := FALSE;
            ProcessData.StartHoming             := FALSE;
            Enabled                             := FALSE;
            Reset                               := FALSE;
            ProcessData.Reset                   := TRUE;
            State                               := -40;
            END_IF

-40:
      IF NOT ProcessData.Reset THEN
            ProcessData.EnableDrive             := TRUE;
            state                               := TempState;
                  END_IF

-20: //Initialisation
      IF Init OR (ProcessData.EnableDrive AND ProcessData.Stop AND
ProcessData.Halt) THEN
            ProcessData.EnableDrive             := TRUE;
            ProcessData.Stop                    := TRUE;
            ProcessData.Brake                   := FALSE;
            ProcessData.Halt                    := TRUE;
            ProcessData.StartTask               := FALSE;
            ProcessData.StartHoming             := FALSE;

            Enabled                             := TRUE;
            STATE                               := -15;
            END_IF


-15: // StartHoming
      IF Init AND NOT ProcessData.HC THEN
            ProcessData.StartHoming             := TRUE;
            State                               := -10;
      ELSIF (ProcessData.EnableDrive AND ProcessData.Stop AND
ProcessData.Halt AND ProcessData.StartHoming) AND NOT ProcessData.HC THEN
            State                               := -10;
            END_IF

-10: // Get the X coordinates from the struct
            TempCoordinates                     := Convert(Coordinates);
            TempVelocityArray                   := Convert(Velocities);
            TempPositions                       :=
X_coordinate(TempCoordinates);
            TempVelocities                      :=
X_Velocities(TempVelocityArray);
            TempMode                            := Mode(TempCoordinates);
            ProcessData.Positions               := TempPositions;
            ProcessData.Velocities              := TempVelocities;
            ProcessData.Mode                    := TempMode;
            State                               := 0;
```

```
0: // Check if homing is completed
      IF ProcessData.HC AND (Init  OR (ProcessData.EnableDrive AND
ProcessData.Stop AND ProcessData.Halt)) THEN
            HC                                    := TRUE;
            ProcessData.StartHoming               := FALSE;
            Enabled                               := FALSE;
            Init                                  := FALSE;
            STATE                                 := 10;
            END_IF

10: // Go to nextposition
      IF Start OR TempMode[ProcessData.NumberCurrentPosition-1] THEN
            ProcessData.Positions                 := TempPositions;
            ProcessData.Velocities                := TempVelocities;
            ProcessData.Mode                      := TempMode;
            ProcessData.StartTask                 := TRUE;
            ProcessData.Reset                     := FALSE;
            State                                 := 20;
            END_IF


20: // check if motion is completed
      IF ProcessData.MC THEN
            ProcessData.StartTask                 := FALSE;
            ProcessData.MC                        := FALSE;
            State                                 := 10;
            END_IF

END_CASE
Start                                             := FALSE;
```

Bijlage IV: Code Two_Dimensional

```
IF Reset THEN
        State                                           := -50;
END_IF
IF TotalReset THEN
        State                                           := - 100;
END_IF
IF NOT XProcessData.MC AND NOT YProcessData.MC THEN
        Enabled                                         := TRUE;
ELSE
        Enabled                                         := FALSE;
END_IF
IF XProcessData.SupplyVoltagePresent AND YProcessData.SupplyVoltagePresent
THEN
CASE State OF
-100: // Total reset
        IF TotalReset THEN
                Start                                   := FALSE;
                XProcessData.StartTask                  := FALSE;
                YProcessData.StartTask                  := FALSE;
                HC
                XProcessData.HC                         := FALSE;
                XProcessData.MC                         := FALSE;
                XProcessData.TotalReset                 := TRUE;
                YProcessData.HC                         := FALSE;
                YProcessData.MC                         := FALSE;
                YProcessData.TotalReset                 := TRUE;
                State                                   := -95;
                TotalReset                              := FALSE;
                YProcessData.NumberCurrentPosition      := 0;
                XProcessData.NumberCurrentPosition      := 0;
                Text                                    := "Resetting
Completely";
        ELSE
                XProcessData.StartTask                  := TRUE;
                YProcessData.StartTask                  := TRUE;
                State
        END_IF

-95: // Get position 0 for y-as
                YProcessData.Positions                  := ERRORPOS;
                YProcessData.Velocities                 := ERRORVEL;
                XProcessData.Positions                  := ERRORPOS;
                XProcessData.Velocities                 := ERRORVEL;
                YProcessData.StartTask                  := TRUE;
        IF YProcessData.HC THEN
                State                                   := -90;
        END_IF
```

```
-90: // Get position 0 for x-as
        XProcessData.StartTask                          := TRUE;
        IF XProcessData.HC THEN
                XProcessData.EnableDrive                := FALSE;
                XProcessData.Stop                       := FALSE;
                XProcessData.Brake                      := TRUE;
                XProcessData.Halt                       := FALSE;
                XProcessData.StartTask                  := FALSE;
                XProcessData.StartHoming                := FALSE;
                XProcessData.HC                         := FALSE;
                YProcessData.EnableDrive                := FALSE;
                YProcessData.Stop                       := FALSE;
                YProcessData.Brake                      := TRUE;
                YProcessData.Halt                       := FALSE;
                YProcessData.StartTask                  := FALSE;
                YProcessData.StartHoming                := FALSE;
                YProcessData.HC                         := FALSE;

                Enabled                                 := FALSE;
                HC                                      := FALSE;
                State                                   := -20;
        END_IF

-50: //reset
        IF Reset THEN
                XProcessData.EnableDrive                := TRUE;
                XProcessData.Stop                       := TRUE;
                XProcessData.Brake                      := FALSE;
                XProcessData.Halt                       := TRUE;
                XProcessData.MC                         := FALSE;
                XProcessData.StartTask                  := FALSE;
                XProcessData.StartHoming                := FALSE;
                YProcessData.EnableDrive                := TRUE;
                YProcessData.Stop                       := TRUE;
                YProcessData.Brake                      := FALSE;
                YProcessData.Halt                       := TRUE;
                YProcessData.MC                         := FALSE;
                YProcessData.StartTask                  := FALSE;
                YProcessData.StartHoming                := FALSE;
                Enabled                                 := FALSE;
                Reset                                   := FALSE;
                XProcessData.Reset                      := TRUE;
                YProcessData.Reset                      := TRUE;
                Start                                   := FALSE;
                State                                   := -45;
                Text                                    := "Resetting";
        END_IF

-45: // check if y-as is in the right position
        IF NOT XProcessData.Reset AND NOT YProcessData.Reset THEN
                IF NOT (ABS(YProcessData.ActualPosition -
YPositions[YProcessData.NumberCurrentPosition-1]) < 4) THEN
                        YProcessData.StartTask          := TRUE;
                END_IF
                State                                   := -40;
        END_IF
```

```
-40: // check if x-as is in the right position
        YProcessData.StartTask                          := FALSE;
        IF (ABS(YProcessData.ActualPosition -
YPositions[YProcessData.NumberCurrentPosition-1]) < 4)THEN
                IF NOT (ABS(XProcessData.ActualPosition -
XPositions[XProcessData.NumberCurrentPosition -1]) < 4) THEN
                        XProcessData.StartTask          := TRUE;
                END_IF
                State                                   := -30;
        END_IF


-30: // check if y-as is at the same number of position as the x-as
        IF ABS(XProcessData.ActualPosition -
XPositions[XProcessData.NumberCurrentPosition -1]) < 4 THEN
                XProcessData.StartTask                  := FALSE;
                State                                   := 10;
                        END_IF

-20: //Initialisation
        Text                                            := "Waiting for INIT";
        Waiting                                         := TRUE;
        IF Init THEN
                XProcessData.EnableDrive                := TRUE;
                XProcessData.Stop                       := TRUE;
                XProcessData.Brake                      := FALSE;
                XProcessData.Halt                       := TRUE;
                XProcessData.StartTask                  := FALSE;
                XProcessData.StartHoming                := FALSE;
                YProcessData.EnableDrive                := TRUE;
                YProcessData.Stop                       := TRUE;
                YProcessData.Brake                      := FALSE;
                YProcessData.Halt                       := TRUE;
                YProcessData.StartTask                  := FALSE;
                YProcessData.StartHoming                := FALSE;
                Waiting                                 := FALSE;
                Enabled                                 := TRUE;
                STATE                                   := -15;
                Text                                    := "Initializing";
        END_IF

-15: // StartHoming y-as
        IF Init THEN
                YProcessData.StartHoming                := TRUE;
                State                                   := -5;
                Text                                    := "Homing";
                        END_IF
```

```
-10: // get the positions
        TempCoordinates                         := Convert(Coordinates);
        TempVelocityArray                       := Convert(Velocities);
        XPositions                              :=
X_Coordinates(TempCoordinates);
        XVelocities                             :=
X_Velocities(TempVelocityArray);
        TempMode                                := Mode(TempCoordinates);
        XProcessData.Positions                  := XPositions;
        XProcessData.Velocities                 := XVelocities;
        XProcessData.Mode                       := TempMode;

        YPositions                              :=
Y_Coordinates(TempCoordinates);
        YVelocities                             :=
Y_Velocities(TempVelocityArray);
        YProcessData.Positions                  := YPositions;
        YProcessData.Velocities                 := YVelocities;

        Enabled                                 := TRUE;
        State                                   := -5;

-5: // check if y-as homing is complete and start homing x-as
        IF YProcessData.HC AND Init THEN
                YProcessData.StartHoming        := FALSE;
                XProcessData.StartHoming        := TRUE;
                State                           := 0;
                        END_IF

0: // Check if homing is complete
        IF XProcessData.HC AND Init THEN
                Init                            := FALSE;
                XProcessData.StartHoming        := FALSE;
                Enabled                         := FALSE;
                HC                              := TRUE;
                State                           := 10;
        END_IF

10: // Go to next position in the y-as
        IF NOT TempMode[YProcessData.NumberCurrentPosition-1] THEN
                Text                            := "Waiting for Start";
                Waiting                         := TRUE;
        ELSE
                Text                            := "Continous Mode";
                Waiting                         := FALSE;
        END_IF
        IF NOT YProcessData.NumberCurrentPosition =
XProcessData.NumberCurrentPosition THEN
                State                           := 15;
        ELSIF (Start OR TempMode[YProcessData.NumberCurrentPosition-1]) AND
(YProcessData.MC OR YProcessData.HC) THEN
                YProcessData.StartTask          := TRUE;
                YProcessData.HC                 := FALSE;
                YProcessData.MC                 := FALSE;
                Text                            :=
Standard64.WCONCAT(Standard64.WCONCAT("Moving to ",
DINT_TO_WSTRING(Coordinates[YProcessData.NumberCurrentPosition][0])),
Standard64.WCONCAT(", ",
DINT_TO_WSTRING(Coordinates[XProcessData.NumberCurrentPosition][1])));
                Waiting                         := FALSE;
                State                           := 15;
        END_IF
```

```
15: // If y-as has mc then go to nextposition x-as
        YProcessData.StartTask                  := FALSE;
        IF (Start OR TempMode[XProcessData.NumberCurrentPosition-1]) AND
YProcessData.MC AND (XProcessData.MC OR XProcessData.HC) THEN
                XProcessData.MC                 := FALSE;
                XProcessData.HC                 := FALSE;
                XProcessData.StartTask          := TRUE;
                State                           := 20;
        END_IF

20: // check if motions are completed
        XProcessData.StartTask                  := FALSE;
        IF  XProcessData.MC AND YProcessData.MC THEN
                Start                           := FALSE;
                Init                            := FALSE;
                State                           := 10;
        END_IF
END_CASE
ELSE
        Text
END_IF
```

Bijlage V: Variabelen verandering per state vernieuwd CTRL functieblok

State	Variabelen	Nieuwe waarde variabelen	Voorwaarden	Volgende state
-100	ProcessData.TotalReset	FALSE	ProcessData.TotalReset = TRUE	
	Position	SetValuePosition	ProcessData.TotalReset = FALSE	-90
	Velocity	SetValueVelocity		
-90	ProcessData.HC	TRUE	MC = TRUE en Actuele positie ≈ Ingestelde positie	-20
	StartTask	FALSE		
-50	ProcessData.Reset	FALSE	Fault = FALSE	-10
	StartTask	FALSE		
	StartHoming	FALSE		
	ProcessData.StartTask	FALSE		
-20			SupplyVoltagePresent = TRUE en DriveEnabled = TRUE en Ready = TRUE en HaltActive = TRUE	-15
-15	NoPosition	0	StartHoming = TRUE en MC = TRUE	-10
-10	NoOfPositions	MaxPositions (functie)		
	ProcessData.HC	TRUE	DriveIsReferenced = TRUE en MC = TRUE en Actuele positie ≈ Ingestelde positie	0
	StartHoming	FALSE		
	ProcessData.StartHoming	FALSE		
			Niet voldoen aan de bovenstaande voorwaarden en ProcessData.HC = TRUE	0
			Niet voldoen aan de twee bovenstaande voorwaarden	-20
0			DriveIsReferenced = TRUE	10
10	Position	SetValuePosition	StartTask = TRUE en DriveIsReferenced = TRUE en Actuele positie ≈ Ingestelde positie	20
	Velocity	SetValueVelocity		
	MC	FALSE		
	ProcessData.MC	FALSE		
20	ProcessData. NumberCurrentPosition	NumberCurrentPosition	DriveIsReferenced = TRUE en Actuele positie ≈ Ingestelde positie	0
	ProcessData.StartTask	FALSE		
	StartTask	FALSE		
	ProcessData.MC	TRUE		
	NoPosition	0	Bovenstaande voorwaarden en NoPosition = NoOfPositions	20
	NumberCurrentPosition	NoPosition + 1		
	NoPosition	NoPosition + 1	Niet voldoen aan de bovenstaande voorwaarden	
	NumberCurrentPosition	NoPosition		

Bijlage VI: Verandering variabelen per state One_Dimensional

State	Variabelen	Nieuwe waarde variabelen	Voorwaarden	Volgende state
-100	TotalReset	FALSE	TotalReset = TRUE	-95
	ProcessData.TotalReset	TRUE		
	HC	FALSE		
	ProcessData.HC	FALSE		
	ProcessData.StartTask	FALSE		
	ProcessData.StartTask	TRUE	TotalReset = FALSE	-95
-95	ProcessData.Positions	ERRORPOS (0)		
	ProcessData.Velocities	ERRORVEL (100)		
	ProcessData.StartTask	TRUE		
			ProcessData.HC = TRUE	-90
-90	ProcessData.EnableDrive	FALSE	ProcessData.HC = TRUE	-20
	ProcessData.Stop	FALSE		
	ProcessData.Brake	TRUE		
	ProcessData.Halt	FALSE		
	ProcessData.StartTask	FALSE		
	ProcessData.StartHoming	FALSE		
	ProcessData.HC	FALSE		
	Enabled	FALSE		
	HC	FALSE		
-50	ProcessData.StartTask	FALSE	Reset = TRUE	-40
	ProcessData.StartHoming	FALSE		
	Enabled	FALSE		
	Reset	FALSE		
	ProcessData.Reset	FALSE		
-40	ProcessData.EnableDrive	TRUE	ProcessData.Reset = FALSE	State voordat Reset hoog werd
-20	ProcessData.EnableDrive	TRUE	Init = TRUE of (ProcessData.EnableDrive = TRUE en ProcessData.Stop = TRUE en ProcessData.Halt = TRUE)	-15
	ProcessData.Stop	TRUE		
	ProcessData.Brake	FALSE		
	ProcessData.Halt	TRUE		
	ProcessData.StartTask	FALSE		
	ProcessData.StartHoming	FALSE		
-15	ProcessData.StartHoming	TRUE	Init = TRUE en ProcessData.HC = FALSE	-10
			ProcessData.EnableDrive = TRUE en ProcessData.Stop = TRUE en ProcessData.Halt = TRUE en ProcessData.StartHoming = TRUE en ProcessData.HC = FALSE	
-10	TempCoordinates	Convert(Coordinates)		0
	TempVelocityArray	Convert(Velocities)		
	TempPositions	X_Coordinate (TempCoordinates)		
	TempVelocities	X_Velocities (TempVelocityArray)		
	TempMode	Mode(TempCoordinates)		
	ProcessData.Positions	TempPositions		
	ProcessData.Velocities	TempVelocities		
	ProcessData.Mode	TempMode		

0	HC	TRUE	ProcessData.HC en (Init = TRUE of (ProcessData.EnableDrive = TRUE en ProcessData.Stop = TRUE en ProcessData.Halt = TRUE en ProcessData.StartHoming = TRUE)	10
	ProcessData.StartHoming	FALSE		
	Init	FALSE		
10	ProcessData.Positions	TempPositions	Start of TempMode[ProcessData. NumberCurrentPosition - 1]	20
	ProcessData.Velocities	TempVelocities		
	ProcessData.Mode	TempMode		
	ProcessData.StartTask	TRUE		
	ProcessData.Reset	FALSE		
20	ProcessData.StartTask	FALSE	ProcessData.MC	10
	ProcessData.MC	FALSE		
	Start	FALSE		

Bijlage VII: Verandering variabelen per state Two_Dimensional

State	Variabelen	Nieuwe waarde variabelen	Voorwaarden	Volgende state
-100	TotalReset	FALSE	TotalReset = TRUE	-95
	Start	FALSE		
	HC	FALSE		
	XProcessData.HC	FALSE		
	YprocessData.HC	FALSE		
	XProcessData.MC	FALSE		
	YProcessData.MC	FALSE		
	XprocessData.TotalReset	TRUE		
	YProcessData.TotalReset	TRUE		
	XProcessData. NumberCurrentPosition	0		
	YprocessData. NumberCurrentPosition	0		
	XprocessData.StartTask	TRUE	TotalReset = FALSE	
	YprocessData.StartTask	TRUE		
-95	YProcessData.Positions	ERRORPOS (0)		
	YProcessData.Velocities	ERRORVEL (100)		
	XprocessData.Positions	ERRORPOS (0)		
	XprocessData.Velocities	ERRORVEL (100)		
	YprocessData.StartTask	TRUE		
			YprocessData.HC = TRUE	-90
-90	XprocessData.StartTask	TRUE	XProcessData.HC = TRUE	-20
	XProcessData.EnableDrive	FALSE		
	XProcessData.Stop	FALSE		
	XProcessData.Brake	TRUE		
	XProcessData.Halt	FALSE		
	XProcessData.StartTask	FALSE		
	XProcessData.StartHoming	FALSE		
	XProcessData.HC	FALSE		
	YProcessData.EnableDrive	FALSE		
	YProcessData.Stop	FALSE		
	YProcessData.Brake	TRUE		
	YProcessData.Halt	FALSE		
	YProcessData.StartTask	FALSE		
	YProcessData.StartHoming	FALSE		
	YProcessData.HC	FALSE		
	Enabled	FALSE		
	HC	FALSE		
-50	XProcessData.EnableDrive	TRUE	Reset = TRUE	-10
	XProcessData.Stop	TRUE		
	XProcessData.Brake	FALSE		
	XProcessData.Halt	TRUE		
	XProcessData.MC	FALSE		
	XProcessData.StartTask	FALSE		
	XProcessData.StartHoming	FALSE		
	YProcessData.EnableDrive	TRUE		
	YProcessData.Stop	TRUE		
	YProcessData.Brake	FALSE		
	YProcessData.Halt	TRUE		
	YProcessData.MC	FALSE		
	YProcessData.StartTask	FALSE		

	YProcessData.StartHoming	FALSE		
	Enabled	FALSE		
	Reset	FALSE		
	XProcessData.Reset	TRUE		
	YProcessData.Reset	TRUE		
	Start	FALSE		
-45			XprocessData.Reset = FALSE en YprocessData.Reset = FALSE	-40
	YprocessData.StartTask	TRUE	Actuele positie y-as ≠ Ingestelde positie y-as	
-40	XProcessData.StartTask	FALSE		
			Actuele positie y-as ≈ Ingestelde positie y-as	-30
	XprocessData.StartTask	TRUE	Actuele positie x-as ≠ Ingestelde positie x-as	
-30	XprocessData.StartTask	FALSE	Actuele positie x-as ≈ Ingestelde positie x-as	10
-20	XProcessData.EnableDrive	TRUE	Init = TRUE	-15
	XProcessData.Stop	TRUE		
	XProcessData.Brake	FALSE		
	XProcessData.Halt	TRUE		
	XProcessData.StartTask	FALSE		
	XProcessData.StartHoming	FALSE		
	YProcessData.EnableDrive	TRUE		
	YProcessData.Stop	TRUE		
	YProcessData.Brake	FALSE		
	YProcessData.Halt	TRUE		
	YProcessData.StartTask	FALSE		
	YProcessData.StartHoming	FALSE		
	Waiting	FALSE		
	Enabled	TRUE		
-15	YProcessData.StartHoming	TRUE	Init = TRUE	-10
-10	TempCoordinates	Convert(Coordinates)		-5
	TempVelocityArray	Convert(Velocities)		
	XPositions	X_Coordinates(TempCoordinates)		
	XVelocities	X_Velocities(TempVelocityArray)		
	TempMode	Mode(TempCoordinates)		
	XProcessData.Positions	XPositions		
	XProcessData.Velocities	XVelocities		
	YPositions	Y_Coordinates(TempCoordinates)		
	YVelocities	Y_Velocities(TempVelocityArray)		
	YProcessData.Positions	YPositions		
	YProcessData.Velocities	YVelocities		
-5	YProcessData.StartHoming	FALSE	YProcessData.HC = TRUE en Init = TRUE	0
	XProcessData.StartHoming	TRUE		
0	Init	FALSE	XProcessData.HC en INIT = TRUE	10
	XProcessData.StartHoming	FALSE		
	HC	TRUE		

10	YProcessData.StartTask	TRUE	(Start = TRUE of Mode = TRUE) en(YProcessData.MC = TRUE of YProcessData.HC = TRUE)	15
	YProcessData.HC	FALSE		
	YProcessData.MC	FALSE		
			YProcessData. NumberCurrentPosition ≠ XProcessData. NumberCurrentPosition	
15	YProcessData.StartTask	FALSE		
	XProcessData.StartTask	TRUE	(Start = TRUE of Mode = TRUE) en(XProcessData.MC = TRUE of XProcessData.HC = TRUE)	20
	XProcessData.HC	FALSE		
	XProcessData.MC	FALSE		
20	XProcessData.StartTask	FALSE		
	Start	FALSE	XProcessData.MC = TRUE en YProcessData.MC = TRUE	10
	Init	FALSE		

Bijlage VIII: FHPP Manual

De pagina's die gebruikt zijn bij het maken van de functieblokken zijn bijgevoegd achter deze pagina. De gehele FHPP Manual is te vinden op de site van Festo via onderstaande link en via de link in de literatuurlijst.

http://www.festo.com/net/SupportPortal/Files/319397/555696g1.pdf