



EtherCAT Force Control

Open-loop current control / Closed-loop force control



Application Note



www.agito-akribis.com

Member of Akribis Systems group



Revision History

Version	Description	Date
1.0	Initial Release	17 October 2025

Contact Information

Manufacturer Agito Akribis Systems Ltd., Member of Akribis Systems Group
Address 56 Serangoon North Avenue 4, Singapore 555851
Telephone +65 6484 3357
Website www.agito-akribis.com

Copyright Notice

©2025 Agito Akribis Systems Ltd.

All rights reserved. This work may not be edited in any form or by any means without written permission of Agito Akribis Systems Ltd.

Products Rights

AGDx, AGCx, AGMx, AGAx, AGIx, and AGLx are products designed by Agito Akribis Systems Ltd. in Israel. Sales of the products are licensed to Akribis Systems Pte Ltd. under intercompany license agreement.

Agito Akribis Systems Ltd. has full rights to distribute above products worldwide.

Disclaimer

This product documentation was accurate and reliable at the time of its release.

Agito Akribis Systems Ltd. reserves the right to change the specifications of the product described in this manual without notice at any time.

Trademarks

Agito PCSuite is a trademark of Agito Akribis Systems Ltd..

Warranty

This product is warranted to be free of defects in material and workmanship and conforms to the specifications listed in this manual, for a period of 12 months from the shipment date from factory.

Contents

1	Introduction	4
1.1	General Information	4
1.2	Functional Description (CST)	4
2	AGenData User Program	5
2.1	Description of Related EtherCAT Objects	5
2.1.1	Object 2900 _h : AGenData[1-10]	5
2.1.2	Object 2904 _h : AGenDataStatus[1-10]	5
2.1.3	Object 6071 _h : Target Torque	6
2.1.4	Object 6077 _h : Torque Actual Value	6
2.2	Verification & Creating User Program in PCSuite	6
2.2.1	Slow Approach	6
2.2.2	Mode Switching	7
2.2.3	Command Source	10
2.2.4	User Program (Example)	11
2.3	TwinCAT / PLC Programming Example	13
2.3.1	Modifying PDO List	13
2.3.2	Creating Global Variable List	13
2.3.3	Writing POU	14
2.3.3.1	Customized Function Block	16
3	Using PLCOpen Torque Control (CST Mode)	19
3.1	Description of Related EtherCAT Objects	19
3.1.1	Object 2410 _h : Analog Input 1 Value	19
3.1.2	Object 2810 _h : CST Force Loop Enable	19
3.1.3	Object 2811 _h : CST Command Interpolation	19
3.1.4	Object 6071 _h : Target Torque	20
3.1.5	Object 6077 _h : Torque Actual Value	20
3.2	TwinCAT / PLC Programming Example	21
3.2.1	CST Current Loop	21
3.2.1.1	Modifying PDO List	21
3.2.1.2	Writing POU	21
3.2.1.3	Verification	23
3.2.2	CST Force Loop	24
3.2.2.1	Modifying PDO List	24
3.2.2.2	Startup Parameters	24
3.2.2.3	Modifying Output Scaling Factor / CST Interpolation Command Factor	25
3.2.2.4	Writing POU	26
3.2.2.5	Verification	29

1 Introduction

1.1 General Information

This application note describes the various force control methods that can be achieved with our driver. There are two different kind of force control that our drive can achieve, one is through “open-loop” method whereby there is no force sensor feedback connected to the system, and it is outputting the target torque/current according to the ramp. The other method is through “closed-loop” method whereby there is force sensor feedback connected to the system and the drive is trying to close the force loop while outputting the target torque/command. There are currently two methods to achieve this application, one is through “AGenData User Program” and the other is through the conventional way being described in “CiA402-2, CANopen device profile for drives and motion control, Part 2: Operation Modes and Application Data, Chapter 20”, which is the Cyclic Synchronous Torque Mode (CST).

Chapter 2 of this application note describes the use of “AGenData User Program” method and how it can be achieved, and Chapter 3 describes the use of CST mode with examples from TwinCAT.

1.2 Functional Description (CST)

Figure 1 shows the defined input objects as well as the output objects. With this mode, the trajectory generator is located in the control device, not in the drive device. The user may specify the desired target torque (6071_h) and rate of desired ramp and also monitor the outputs torque actual value (6077_h), velocity actual value (606C_h) and position actual value (6064_h)

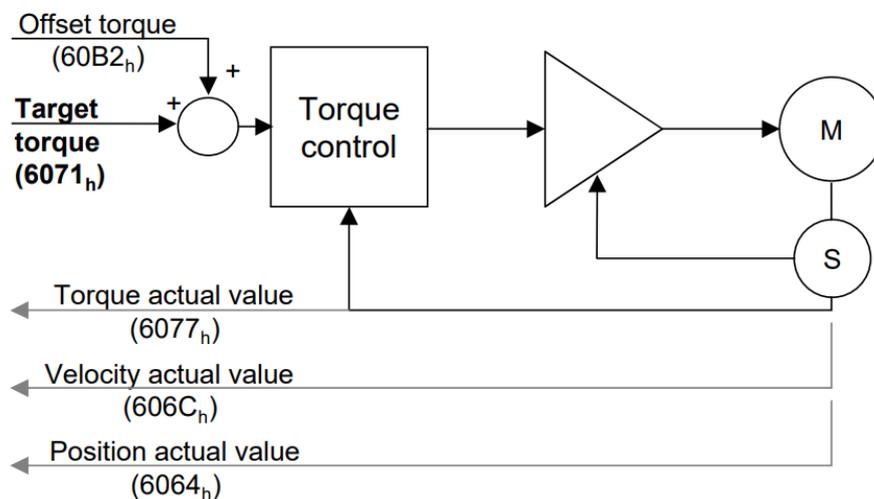


Figure 1 – Cyclic synchronous torque mode overview

2 AGenData User Program

2.1 Description of Related EtherCAT Objects

This section touches on the related EtherCAT Objects (on top of the mandatory ones) that can be / should be mapped to the PDO to ensure cyclic exchanging of information between the master and controller. The mandatory EtherCAT Objects that should have been mapped are “StatusWord”, “Position Actual Value”, “ControlWord” and “Target Position”.

2.1.1 Object 2900_h: AGenData[1-10]

AGenData[1-10] consists of 10 arrays which user can freely use in conjunction with any user program that is running inside the drive itself, this is mainly used for assigning the AGenData value to the drive.

Attribute	Value
Sub-index	00 _h
Description	Highest sub-index supported
Access	rw
PDO Mapping	Yes
Value Range	10
Default Value	10
Sub-index	01 _h
Description	AGenData 1
Access	rw
PDO Mapping	Yes
Value Range	Integer32
Default Value	0
All the subsequent AGenData sub-indices possess the same attribute and value as sub-index 1.	

2.1.2 Object 2904_h: AGenDataStatus[1-10]

AGenDataStatus[1-10] consists of 10 arrays which user can freely use in conjunction with any user program that is running inside the drive itself, this is only used for reading back the AGenData value.

Attribute	Value
Sub-index	00 _h
Description	Highest sub-index supported
Access	rw
PDO Mapping	Yes
Value Range	10
Default Value	10
Sub-index	01 _h
Description	AGenDataStatus 1
Access	rw

PDO Mapping	Yes
Value Range	Integer32
Default Value	0
All the subsequent AGenDataStatus sub-indices possess the same attribute and value as sub-index 1.	

2.1.3 Object 6071_h: Target Torque

This object shall indicate the configured input value for the torque controller in profile torque mode/cst mode. The value shall be given per thousand of rated torque.

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Integer16
Default Value	0000 _h

2.1.4 Object 6077_h: Torque Actual Value

This object shall provide the actual value of the torque. It shall correspond to the instantaneous torque in the motor. The value shall be given per thousand of rated torque. Currently, there is no torque feedback in our drive and since $T = kt * I$ and assuming that the motor constant is static, the calculation of the torque feedback will be in terms of the percentage of the continuous current that is currently used.

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Integer16
Default Value	0000 _h

2.2 Verification & Creating User Program in PCSuite

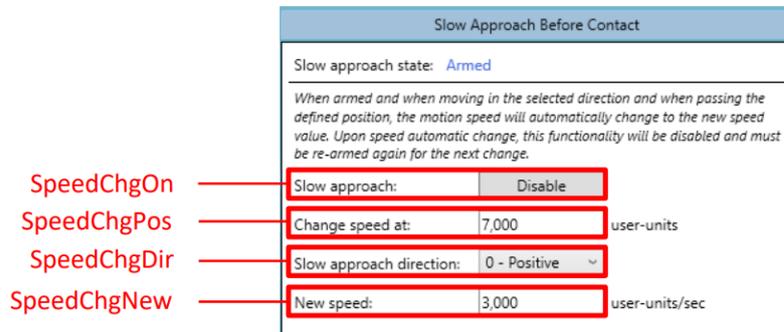
Users can refer to “Agito AN – Force Control” with regards to configuration of force sensor (if any) and tuning of current loop / force loop. This section will be an abstract from the “Agito AN – Force Control” to describe the portion of the UI that the user can use to do their force control verification before writing moving on to writing the force control user program.

2.2.1 Slow Approach

In force control applications, it is common to have the tool move towards the object at a high speed and then slowing down when it approaches so as to reduce the kinetic energy and impulse upon contact.

Verification & Creating User Program in PCSuite

To configure the slow approach, navigate to the “Slow Approach Before Contact” window and set the relevant parameters.



SpeedChgOn — Slow approach: Disable

SpeedChgPos — Change speed at: 7,000 user-units

SpeedChgDir — Slow approach direction: 0 - Positive

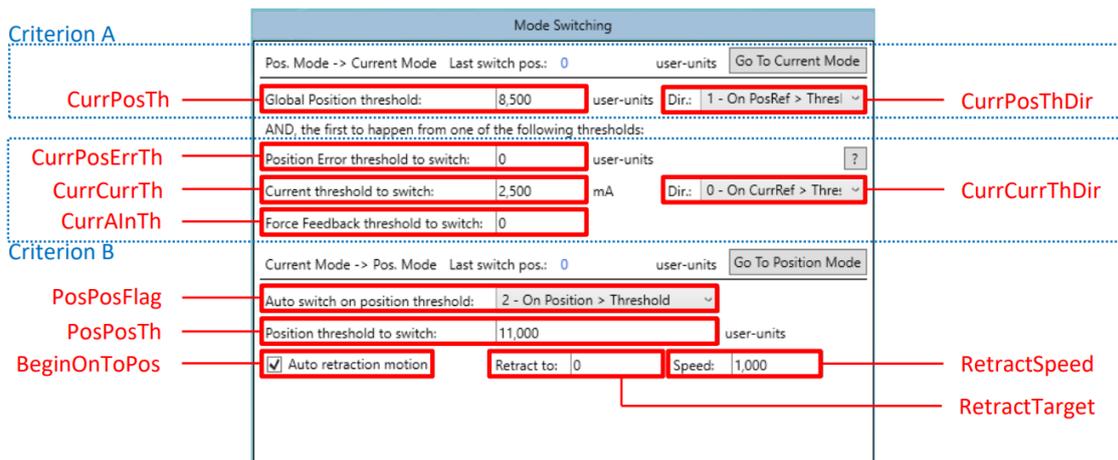
SpeedChgNew — New speed: 3,000 user-units/sec

In Figure x, the parameters are set such that when the motor moves past the position of 7,000 counts in the positive direction, it will start to decelerate (or possibly accelerate) to the new speed of 3000 count/s.

Note: This function must be re-enabled after every cycle of force control. It will automatically disable the function at the end of every force control cycle.

2.2.2 Mode Switching

Navigate to “Mode Switching” window; the trigger parameters to switch from position to force/current mode are configured in this window.



Criterion A

CurrPosTh — Global Position threshold: 8,500 user-units

CurrPosThDir — Dir.: 1 - On PosRef > Thres

CurrPosErrTh — Position Error threshold to switch: 0 user-units

CurrCurrTh — Current threshold to switch: 2,500 mA

CurrCurrThDir — Dir.: 0 - On CurrRef > Thres

CurrAlnTh — Force Feedback threshold to switch: 0

Criterion B

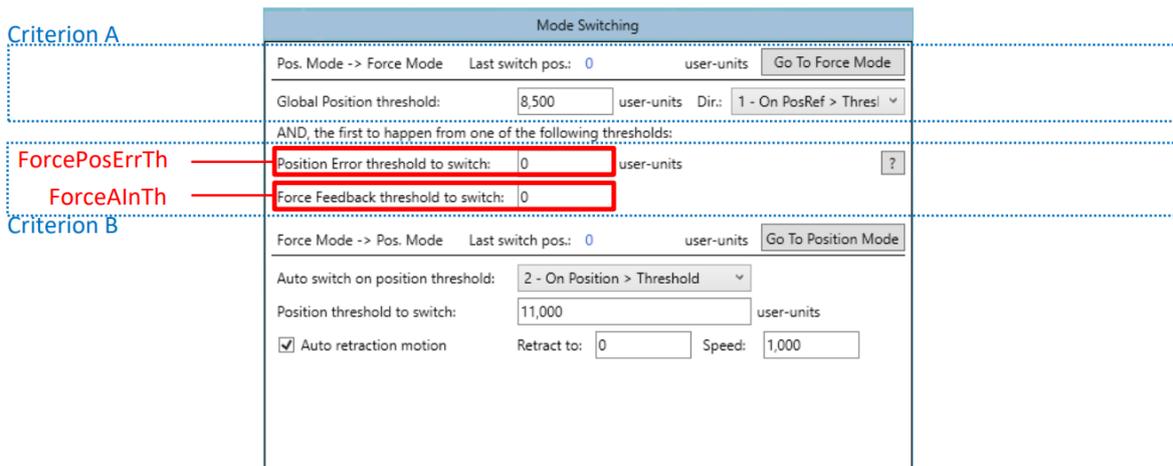
PosPosFlag — Auto switch on position threshold: 2 - On Position > Threshold

PosPosTh — Position threshold to switch: 11,000 user-units

BeginOnToPos — Auto retraction motion

RetractSpeed — Retract to: 0

RetractTarget — Speed: 1,000



Two main criterions have to be fulfilled for switching to occur; in this document it will be referred as Criterion A and Criterion B.

Criterion A is a position criterion which allows for mode switching to occur only when the motor is out of a certain position range. This prevents unintended switching during the acceleration and deceleration phases of the motion profile which tend to have higher currents and higher position errors. Such unintended early switching may cause the motor to accelerate (in current/force mode) over a distance and cause damage to the system or part. See relevant keywords below.

Keyword	Description								
CurrPosTh	<p><u>CurrPosTh</u> defines a position threshold value that <u>PosRef</u> must be greater (or smaller) than, as one of the criteria for switching to Current/Force Mode to occur.</p> <p>Set this value to a position slightly before your contact point. For example, if the starting position is at 0 counts and the contact position is between 9,000 to 10,000 counts, this value can be set to 8,500 counts.</p>								
CurrPosThDir	<p>CurrPosThDir defines if the criteria of CurrPosTh is positive, negative or bypassed when set to the following values.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Criteria</th> </tr> </thead> <tbody> <tr> <td>-1</td> <td>PosRef < CurrPosThDir</td> </tr> <tr> <td>0</td> <td>Bypass</td> </tr> <tr> <td>1</td> <td>PosRef > CurrPosThDir</td> </tr> </tbody> </table> <p>Set this value to 1 if moving in a positive direction towards the contact point. E.g., Starting point is at 0 counts and contact point is at 10,000 counts.</p> <p>Or set the value to -1 if the contact point is in the negative direction. E.g., Starting point is at 20,000 counts and contact point is at 10,000 counts.</p>	Value	Criteria	-1	PosRef < CurrPosThDir	0	Bypass	1	PosRef > CurrPosThDir
Value	Criteria								
-1	PosRef < CurrPosThDir								
0	Bypass								
1	PosRef > CurrPosThDir								

Verification & Creating User Program in PCSuite

Criterion B is composed of sub-criteria (three sub-criteria **[B1, B2, B3]** for current operation mode and two **[B1, B2]** for force operation mode).

Criterion B is fulfilled as long as any single one of the sub-criteria, B1 or B2 or B3 is fulfilled.

Sub-criterion B1: The first sub-criterion checks for position error, if the position error grows bigger than the threshold value, then criterion A is fulfilled. See relevant keywords below.

Keyword	Description
CurrPosErrTh/ ForcePosErrTh	CurrPosErrTh/ForcePosErrTh defines a threshold value for going into Current/Force Operation Mode if position error exceeds the threshold. If the threshold is set to 0, then this trigger is not in use.

Sub-criterion B2: The second sub-criterion checks for analog input, if the analog input grows larger (or smaller) than the threshold value, then criterion A is fulfilled. See relevant keywords below.

Keyword	Description
CurrAlnTh/ ForceAlnTh	CurrAlnTh/ForceAlnTh defines a threshold value for going into Current/Force Operation Mode if analog input is greater (less than, if threshold value is negative) than the threshold. If the threshold is set to 0, then this trigger is not in use.

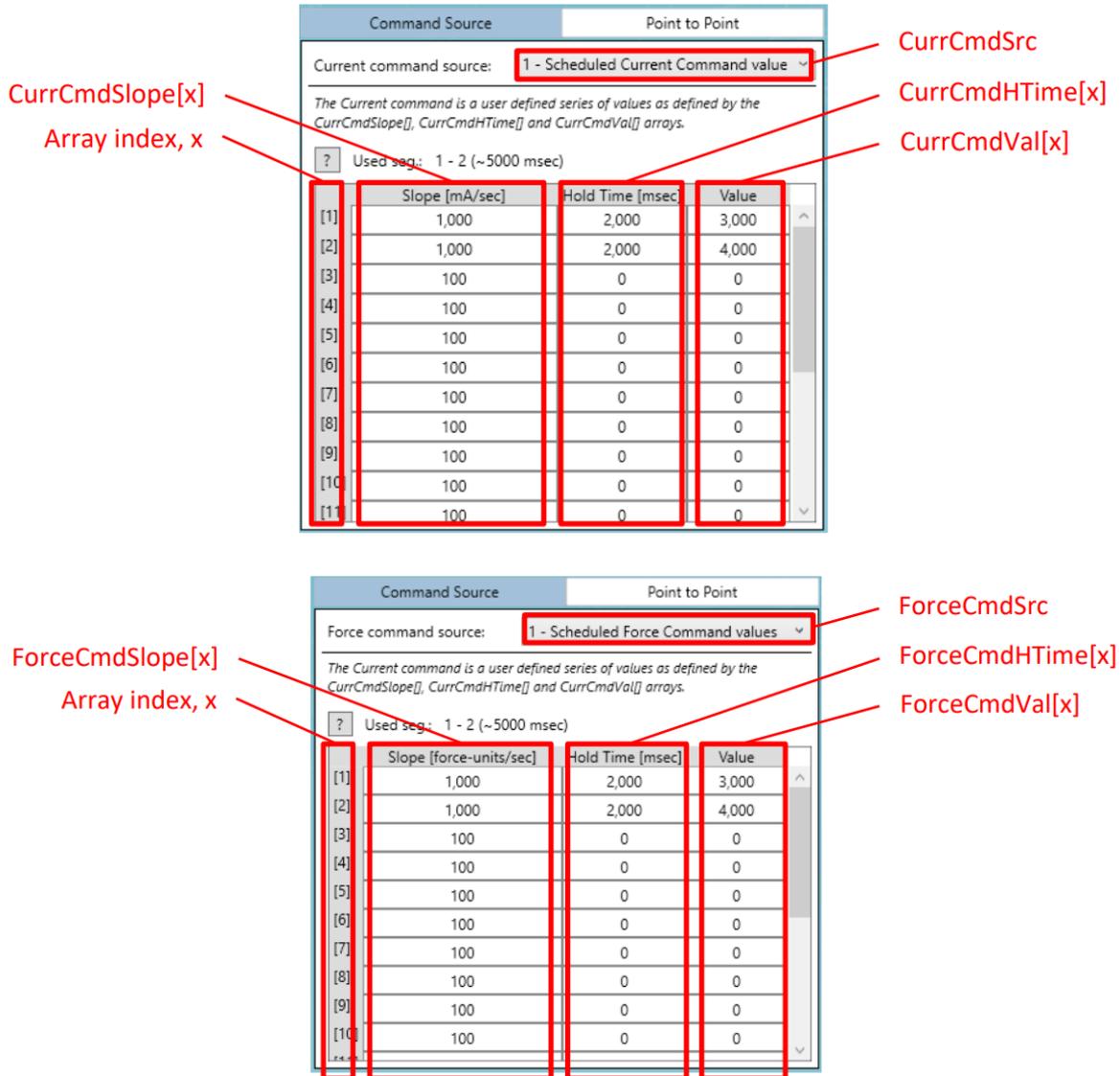
Sub-criterion B3: The third sub-criterion is only applicable for current operation mode and checks for current reference, if current reference is greater than (or less than) the threshold value, then criterion A is fulfilled. See relevant keywords below.

Keyword	Description						
CurrCurrTh	CurrCurrTh defines a position threshold value for going into Current Operation Mode if the motor current is greater (or lesser) than the threshold. If the threshold value is set to 0, then the trigger is not in use.						
CurrCurrThDir	CurrCurrThDir defines if the criteria of CurrCurrTh is positive or negative when set to the following values. <table border="1" data-bbox="459 1621 1401 1727"> <thead> <tr> <th>Value</th> <th>Criteria</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>CurrRef > CurrCurrTh</td> </tr> <tr> <td>1</td> <td>CurrRef < CurrCurrTh</td> </tr> </tbody> </table>	Value	Criteria	0	CurrRef > CurrCurrTh	1	CurrRef < CurrCurrTh
Value	Criteria						
0	CurrRef > CurrCurrTh						
1	CurrRef < CurrCurrTh						

When both **Criterion A and Criterion B** are fulfilled, operation mode switching will be triggered. **Upon triggering, the threshold value of the sub-criterion B that was triggered will be reset to 0, this is to prevent unintended repeated triggering. Users will have to re-apply the value after every cycle.** The remaining sub-criteria will remain unchanged. Although the controller allows flexibility in defining multiple triggers (sub-criteria of B) at the same time, it is recommended to select and just use one of it.

2.2.3 Command Source

The next step would be to configure the current/force profile, navigate to the “Command Source” window. In the relevant table, assign the current/force profile. Alternatively, the command source can also be configured to come from an analog input, but it will not be shown in the example.



Current Command Source Configuration:

Current command source: 1 - Scheduled Current Command value

The Current command is a user defined series of values as defined by the CurrCmdSlope[], CurrCmdHTime[] and CurrCmdVal[] arrays.

Used seg: 1 - 2 (~5000 msec)

Array index, x	Slope [mA/sec]	Hold Time [msec]	Value
[1]	1,000	2,000	3,000
[2]	1,000	2,000	4,000
[3]	100	0	0
[4]	100	0	0
[5]	100	0	0
[6]	100	0	0
[7]	100	0	0
[8]	100	0	0
[9]	100	0	0
[10]	100	0	0
[11]	100	0	0

Force Command Source Configuration:

Force command source: 1 - Scheduled Force Command values

The Current command is a user defined series of values as defined by the CurrCmdSlope[], CurrCmdHTime[] and CurrCmdVal[] arrays.

Used seg: 1 - 2 (~5000 msec)

Array index, x	Slope [force-units/sec]	Hold Time [msec]	Value
[1]	1,000	2,000	3,000
[2]	1,000	2,000	4,000
[3]	100	0	0
[4]	100	0	0
[5]	100	0	0
[6]	100	0	0
[7]	100	0	0
[8]	100	0	0
[9]	100	0	0
[10]	100	0	0
[11]	100	0	0

In the example above, the profile is configured to have two steps. Upon the trigger to switch operation modes, the controller will increase/decrease the force/current command to 3,000 units at a rate of 1,000 units/s. Then, the current/force command will then be held at this level for 2,000ms. After which, the controller will increase the current/force command to 4,000mA at a rate of 1,000 units/s and hold at this level for another 2,000ms. Units will be in mA or force-units depending on the mode selected.

Keyword	Description						
CurrCmdSrc/ ForceCmdSrc	CurrCmdSrc/ForceCmdSrc defines the command source. <table border="1" data-bbox="459 331 1396 616"> <thead> <tr> <th>Value</th> <th>Source</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Analog input (In addition to setting this keyword to 0, it is also required to set the analog input to Force Command in IO Page)</td> </tr> <tr> <td>1</td> <td>Scheduled by table of values</td> </tr> </tbody> </table>	Value	Source	0	Analog input (In addition to setting this keyword to 0, it is also required to set the analog input to Force Command in IO Page)	1	Scheduled by table of values
Value	Source						
0	Analog input (In addition to setting this keyword to 0, it is also required to set the analog input to Force Command in IO Page)						
1	Scheduled by table of values						
CurrCmdVal[x]/ ForceCmdVal[x]	CurrCmdVal[x]/ForceCmdVal[x] defines the current/force command value for step x.						
CurrCmdSlope[x]/ ForceCmdSlope[x]	From the previous step to step x, there 'will' be a difference in the current/force command. Instead of a step jump, the controller will gradually increase/decrease from the current/force command at the rate defined in CurrCmdSlope[x]/ForceCmdSlope[x], in mA/sec or force-unit/sec.						
CurrCmdHTime[x]/ ForceCmdHTime[x]	After increasing/decreasing the current/force reference to the command value, the controller will hold the reference at the commanded value for the period of time defined in CurrCmdHTime[]/ForceCmdHTime[].						

2.2.4 User Program (Example)

This section shows an example of using the IDE to program the various parameters for both current open loop and force close loop in conjunction with the use of AGenData.

```

//
main([10,30],[5,20],[800,1000])
//
// Place here the main code of your program
AGenData[1] = 0 //初始化力控开始下压开关
AGenData[3] = 0 //初始化执行Begin开关
AGenData[4] = 0 //初始化力控按压完成状态
AGenData[5] = 0 //初始化线程#2激活状态
AGenData[6] = 0 //初始化线程#2复位开关
AProgRun[2], 2 //运行开环力控线程

//设定门限值【注意：需要根据实际情况，设定为合理的值】
AGenData[7]=-19000//400//40500 //设定A轴切换到力控模式的全局位置门限值（正数：按压方向为编码器正方向；负数：按压方向为编码器负方向）
AGenData[2]=300 //设定A轴切换到力控模式的压力反馈门限值

//设定力控基本参数【注意：EtherCAT主站控制时，应注释掉，交给EtherCAT主站去赋值】
// AGenData[10]=150 //设定开环压力电流目标值
// AGenData[9]=10 //设定压力电流保持时间
// AGenData[8]=20000 //设定运动速度（正值：按压方向为编码器正方向；负值：按压方向为编码器负方向）
AStop
while(AMotionStat!=0)
end
AMotionMode=19

while (1)

if(AProgStat[2]==1)
AGenData[5]=1
else if(AProgStat[2]==0)
AGenData[5]=0

```

Initialize AGenData values at start up.

Settings of parameters to be use in other task for current/force control.

Initialization process, default motionmode is 19 for CSP mode. Not necessary.

Verification & Creating User Program in PCSuite

```

while (1)
  if(AProgStat[2]==1)
    AGenData[5]=1
  else if(AProgStat[2]==0)
    AGenData[5]=0
  end

  if (AGenData[6]==1) //EtherCAT主站发 "AGenData[6]=1" 重启线程#2
    AGenData[5]=0
    AGenData[1]=0
    AGenData[3] = 0 //初始化执行Begin开关
    AGenData[4] = 0 //初始化力控按压完成状态
    AProgHalt[2]
    AProgReset[2]
    AWaitTime, 10
    AProgRun[2], 2
    if(AProgStat[2]==1)
      AGenData[5]=1
    else if(AProgStat[2]==0)
      AGenData[5]=0
    end

    AStop
    while(AMotionStat!=0)
    end
    AMotionMode=19
    AMotorOn=1

    AGenData[6]=0
  end
end
  
```

Checking if the User Program thread 2 is running and indicate in AGenData.

Resetting of the user program and initialize parameters if user program has an error and indicate that User Program has reset by deasserting AGenData[6].

```

task: OpenLoopFC[2]
  AGoToPosMode //切换到位置模式
  while(AOperationMode!=3)
  end
end
  
```

Running Thread 2 Task 2.

```
while(1)
```

```

//开环力控下压
if(AGenData[1]==1)
  AGenData[4] = 0 //初始化力控按压完成状态
  AGoToPosMode //切换到位置模式
  while(AOperationMode!=3)
  end
end
  
```

If AGenData[1] is set from the master, initialise

```

  AStop //停止运动并切换到JOG速度运动模式
  AMotorOn=1
  AMotionMode=0
  
```

```

//模式切换的阈值设定
ACurrPosThDir=-1 //设定切换到力控模式的全局位置条件方向 (1: 按压方向为编码器正方向; -1: 按压方向为编码器负方向)
ACurrPosTh=AGenData[7] //设定切换到力控模式的全局位置门限值 (正数: 按压方向为编码器正方向; 负数: 按压方向为编码器负方向)
ACurrCurrThDir=1 //设定切换到力控模式的电机电流条件方向 (0: 按压方向为编码器正方向; 1: 按压方向为编码器负方向)
  
```

Relevant threshold settings to switch from position to force/current mode.

```

ACurrCmdSrc=1
ACurrCmdSlope[1]=(abs(AGenData[10]))*100 //开环力控电流变化斜率 (只能是正值)
ACurrCmdHTime[1]=-1//AGenData[9] //开环力控压力电流保持实际
ACurrCmdVal[1]=AGenData[10] //开环力控压力电流指令值
ACurrCmdHTime[2]=0
  
```

Relevant command source and slope profile for force/current mode.

```
//AForceCmdSrc = 1
```

```

//AForceCmdSrc = 1
//AForceCmdSlope[1]=(abs(AGenData[10]))*100
//AForceCmdHTime[1]=-1, if its negative its infinite hold time.
//AForceCmdVal[1]=AGenData[10]
//AForceCmdHTime[2]=0
  
```

```
ASpeedChgOn = 0 //开启或关闭低速功能, 0: 关闭; 1: 开启
```

On the fly speed change option, can input relevant on the fly speed change parameters as well if opted for it.

```

AAccel=(abs(AGenData[8]))*100 //加速度 (只能是正值)
ADecel=(abs(AGenData[9]))*100 //减速度 (只能是正值)
ASpeed=AGenData[8] //设定运动速度 (正数: 按压方向为编码器正方向; 负数: 按压方向为编码器负方向)
  
```

Motion profile to reach contact point.

```

  ABegin
  while (AMotionStat==0)
  end
  
```

```
ACurrCurrTh=AGenData[2] //设定切换到力控模式的电机电流门限值 (正数: 按压方向为编码器正方向; 负数: 按压方向为编码器负方向)
```

Threshold of current to switch, can be put together with the other relevant threshold.

```

while (AOperationMode==3) //首先, 等待位置模式下的运动结束
end
AGenData[4] = 1 // finish force control力控按压完成
//while(AOperationMode==1) //其次, 等待开环力控按压结束
//end
AGenData[1] = 0
end
  
```

Waiting for current/force control to finish as it will auto switch back to position mode. Once done, it is indicated by asserting AGenData[4].

```

//执行Begin (开始运动) 指令
if(AGenData[3]==1)
  AGoToPosMode
  AStop
end
  
```

TwinCAT / PLC Programming Example

```

//执行Begin (开始运动) 指令
if (AGenData[3] == 1)
    AGoToPosMode
    AStop
    while (AMotionStat != 0)
    end
    AMotionMode = 19 // 切换到CSP模式
    while (AMotionMode != 19)
    end

    AwaitStatus[7], 0
    ABegin
    while (AMotionStat == 1)
    end
    AGenData[4] = 0
    AGenData[3] = 0
end

end //end of while(1)
endoftask

```

Waiting for user to give AGenData[3] = 1 to switch back the motion mode to CSP mode in order to move via EtherCAT control.

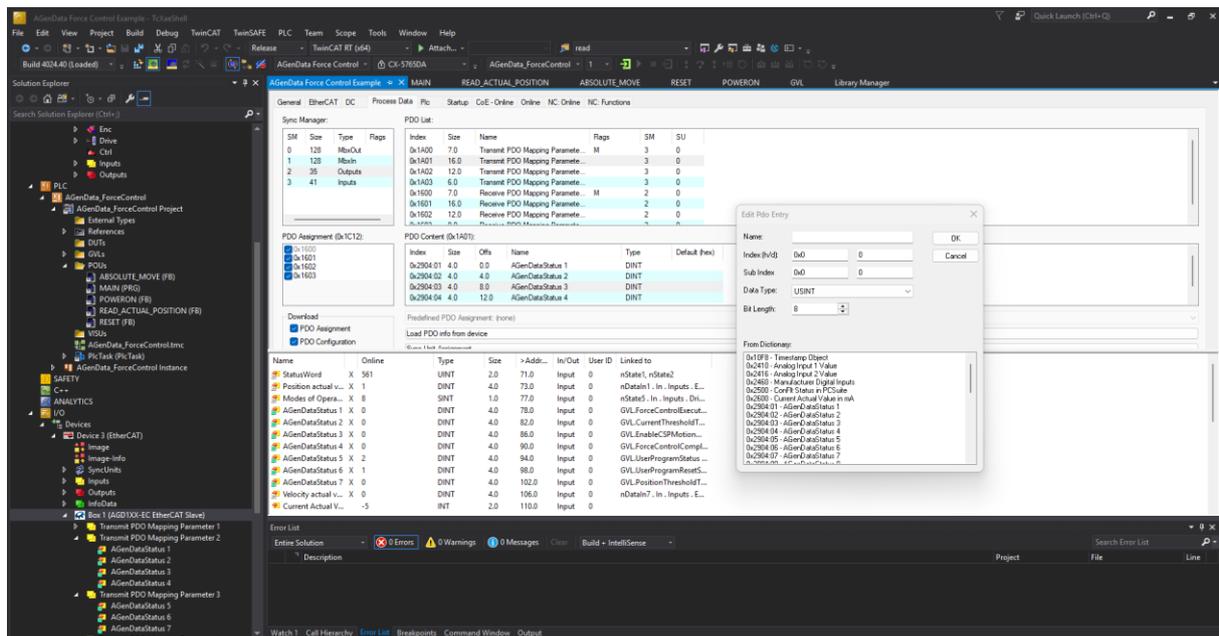
User will have to ensure that the Target Position of the EtherCAT master matches the current actual position of the motor as by using this method we have decoupled the control from the master and will have to realign everything before giving control back to the master.

There will be an example shown in the twincat controls below to show this.

2.3 TwinCAT / PLC Programming Example

2.3.1 Modifying PDO List

It is important to map the AGenData / AGenDataStatus object (current limit is 10 array size for both object) as an PDO to ensure that the cyclic exchange of information between the master and the slave. AGenData is not mapped by default in the PDO list, users have to navigate to the pages whereby they are able to do the PDO configuration to modify the slave PDO list. Inside the TxPDO section, users will be able to find AGenDataStatus as a PDO object, this is generally used for status readback of the particular AGenData that is located under the RxPDO section.



SM	Size	Type	Flags	SM	SU
0	128	MsOut			
1	128	MsIn			
2	35	Outputs			
3	41	Inputs			

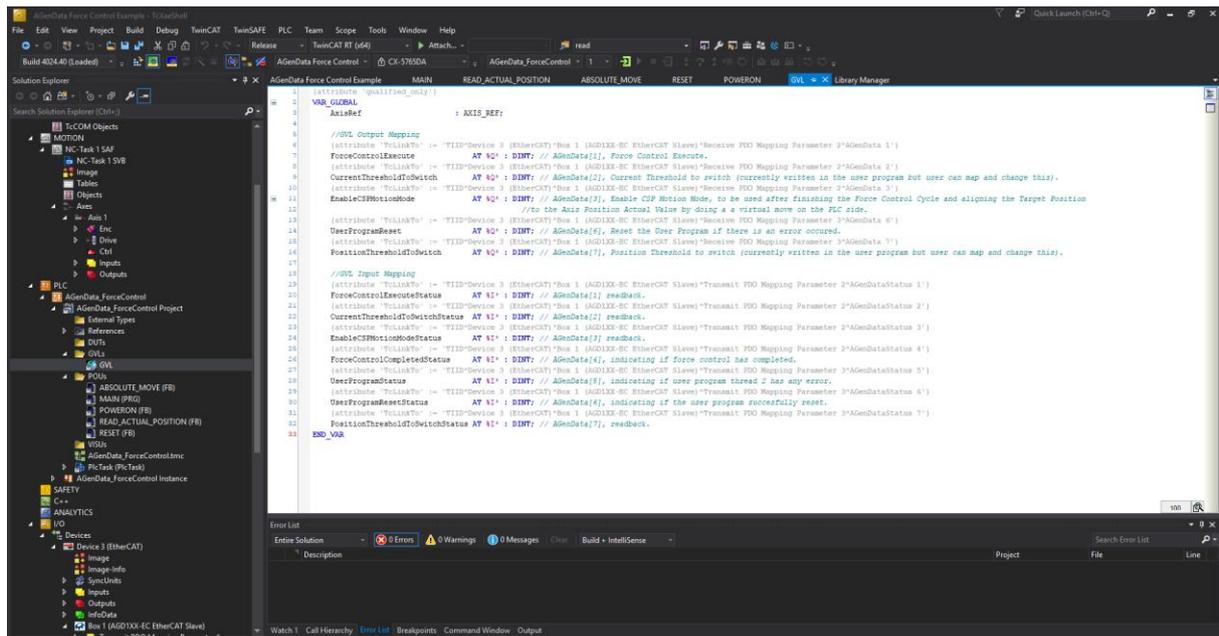
Index	Size	Name	Type	Default (hex)
0x1400	7.0	Transmit PDO Mapping Parameter 1	M	3 0 0
0x1401	16.0	Transmit PDO Mapping Parameter 2	M	3 0 0
0x1402	12.0	Transmit PDO Mapping Parameter 3	M	3 0 0
0x1403	6.0	Transmit PDO Mapping Parameter 4	M	3 0 0
0x1600	7.0	Receive PDO Mapping Parameter 1	M	2 0 0
0x1601	16.0	Receive PDO Mapping Parameter 2	M	2 0 0
0x1602	12.0	Receive PDO Mapping Parameter 3	M	2 0 0
0x1603	6.0	Receive PDO Mapping Parameter 4	M	2 0 0

Index	Size	Name	Type	Default (hex)
0x2004.01	4.0	0.0	DINT	
0x2004.02	4.0	AGenDataStatus 1	DINT	
0x2004.03	4.0	AGenDataStatus 2	DINT	
0x2004.04	4.0	AGenDataStatus 3	DINT	
0x2004.05	4.0	AGenDataStatus 4	DINT	

2.3.2 Creating Global Variable List

Once the PDO list have been modified to include the AGenData/AGenDataStatus, we can create a global variable list and give names to it and link it with the PDO object that we desire so that these variables can be used while creating the customized function block for the force control or any application usage.

TwinCAT / PLC Programming Example



2.3.3 Writing POU

The next step will be writing the whole script to test out the force control application. Inside the example below, a customized function block is created to start / indicate when the force/current control has finished and realigning both the master and slave back to move via CSP mode.

```

1  GVL.JoggingSpeed := 10000;
2
3  (*Power On*)
4  axisPower(axis_Test := GVL.AxisRef, bPowerEnable := bPowerEnable, bPowerStatus => bPowerStatus, bPowerBusy => bPowerBusy, bPowerActive => bPowerActive, bPowerError => bPowerError, nPowerErrorID => nPowerErrorID)
5
6  (*Reset*)
7  axisReset(axis_Test := GVL.AxisRef, bResetExecute := bResetExecute, bResetDone => bResetDone, bResetBusy => bResetBusy, bResetError => bResetError, nResetErrorID => nResetErrorID);
8  (*Move Absolute*)
9  axisMoveAbsolute(axis_Test := GVL.AxisRef, bMoveAbsoluteExecute := bMoveAbsoluteExecute, bAbortDueToCst := bAbortDueToCst, nMoveAbsoluteErrorID => nMoveAbsoluteErrorID, bMoveAbsoluteDone => bMoveAbsoluteDone, bMoveAbsoluteBusy => bMoveAbsoluteBusy, bMoveAbsoluteActive => bMoveAbsoluteActive);
10 (*Read Actual Position*)
11 axisReadPosition(axis_Test := GVL.AxisRef, bReadActualPosEnable := TRUE, bReadActualPosValid => bReadActualPosValid, bReadActualPosBusy => bReadActualPosBusy, bReadActualPosError => bReadActualPosError, nReadActualPosErrorID => nReadActualPosErrorID, nActualPosition => nActualPosition);
12
13 (*Force Control*)
14 axisForceControl(bForceControlExecute := bForceControlExecute, nCurrPosTh := GVL.PositionThresholdToSwitch, nCurrCurrTh := GVL.CurrentThresholdToSwitch, nJoggingSpeed := GVL.JoggingSpeed, bForceControlCompleted => bForceControlCompleted, bForceControlBusy => bForceControlBusy, bForceControlActive => bForceControlActive, bForceControlError => bForceControlError);
15 (*AlignMasterSlave*)
16 axisAlignMasterSlave(bAlignMasterSlaveExecute := bAlignMasterSlaveExecute, bAlignMasterSlaveCompleted => bAlignMasterSlaveCompleted, bAlignMasterSlaveBusy => bAlignMasterSlaveBusy, bAlignMasterSlaveActive => bAlignMasterSlaveActive, bAlignMasterSlaveError => bAlignMasterSlaveError);
17 (*User Program Reset*)
18 axisUserProgramReset(bUserProgramResetExecute := bUserProgramResetExecute, bUserProgramResetCompleted => bUserProgramResetCompleted, bUserProgramResetBusy => bUserProgramResetBusy, bUserProgramResetActive => bUserProgramResetActive, bUserProgramResetError => bUserProgramResetError);
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Instance of function block used

```

CASE seq1State OF
  SEQDEMOSTATE.SEQ1_RESET;
  (*Initialization logic / Reset*)
  bResetExecute := TRUE;
  bUserProgramResetExecute := TRUE;
  bPowerEnable := FALSE;

  IF bResetDone AND bUserProgramResetCompleted THEN
    bResetExecute := FALSE;
    bUserProgramResetExecute := FALSE;

    seq1State := SEQDEMOSTATE.SEQ1_POWER_ENABLE;
  ELSIF bResetError THEN
    bResetExecute := FALSE;
    bUserProgramResetExecute := FALSE;
    nResetErrorID := nResetErrorID;

    seq1State := SEQDEMOSTATE.SEQ1_FAULT;
  END_IF

  SEQDEMOSTATE.SEQ1_POWER_ENABLE;
  (*Axis Power*)
  bPowerEnable := TRUE;

  IF bPowerStatus THEN
    seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE;
  ELSIF bPowerError THEN

```

Reset / Initialization Sequence

TwinCAT / PLC Programming Example

```

SEQDEMO.STATE.SEQ1_POWER_ENABLE:
('Axis Power')
bPowerEnable := TRUE;

IF bPowerStatus THEN
  seq1State := SEQDEMO.STATE.SEQ1_FORCE_CONTROL_ENABLE;
ELSIF bPowerError THEN
  bPowerEnable := FALSE;
  seq1State := SEQDEMO.STATE.SEQ1_FAULT;
END_IF

SEQDEMO.STATE.SEQ1_FORCE_CONTROL_ENABLE:
('Force Control Execute')
bForceControlExecute := TRUE;

IF bForceControlBusy AND bForceControlActive THEN
  seq1State := SEQDEMO.STATE.SEQ1_FORCE_CONTROL_WAIT;
ELSE
  seq1State := SEQDEMO.STATE.SEQ1_FORCE_CONTROL_ENABLE;
END_IF

SEQDEMO.STATE.SEQ1_FORCE_CONTROL_WAIT:

IF bForceControlCompleted THEN
  bForceControlExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE;
  seq1State := SEQDEMO.STATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE;
ELSIF bForceControlError THEN
  bForceControlExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_FAULT;
END_IF

SEQDEMO.STATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE:
('setting virtual move to current position of the axis')
axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
axisMoveAbsolute.nTargetJerk := nTargetJerk_1;

bMoveAbsoluteExecute := TRUE;

seq1State := SEQDEMO.STATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE_WAIT;

SEQDEMO.STATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE_WAIT:
IF NOT bMoveAbsoluteBusy AND bMoveAbsoluteDone THEN
  bMoveAbsoluteExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_ALIGN_MASTER_SLAVE;
ELSIF NOT bMoveAbsoluteBusy AND (bMoveAbsoluteError OR bMoveAbsoluteCommandAborted) THEN
  bMoveAbsoluteExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_FAULT;
END_IF

SEQDEMO.STATE.SEQ1_ALIGN_MASTER_SLAVE:
('This is solely changing the driver motion mode to work in CSP in EtherCAT, can be integrated into the MoveAbsFunction logic as well, as we did a virtual move to align driver position reference and Target Position on EtherCAT')
bAlignMasterSlaveExecute := TRUE;

seq1State := SEQDEMO.STATE.SEQ1_ALIGN_MASTER_SLAVE_WAIT;

SEQDEMO.STATE.SEQ1_ALIGN_MASTER_SLAVE_WAIT:
IF bAlignMasterSlaveCompleted THEN
  bAlignMasterSlaveExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_MOVE_TO_TARGET;
ELSIF bAlignMasterSlaveError THEN
  bAlignMasterSlaveExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_FAULT;
END_IF

SEQDEMO.STATE.SEQ1_MOVE_TO_TARGET:
axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
axisMoveAbsolute.nTargetJerk := nTargetJerk_2;

bMoveAbsoluteExecute := TRUE;

seq1State := SEQDEMO.STATE.SEQ1_MOVE_TO_TARGET_WAIT;

SEQDEMO.STATE.SEQ1_MOVE_TO_TARGET_WAIT:
IF NOT bMoveAbsoluteBusy AND bMoveAbsoluteDone THEN
  bMoveAbsoluteExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_FORCE_CONTROL_ENABLE;
ELSIF NOT bMoveAbsoluteBusy AND (bMoveAbsoluteError OR bMoveAbsoluteCommandAborted) THEN
  bMoveAbsoluteExecute := FALSE;

  seq1State := SEQDEMO.STATE.SEQ1_FAULT;
END_IF
END_CASE

```

Axis Power

Force Control & Auto Retract Motion to a certain position (e.g. 2000 counts)

Virtual absolute move of the axis to the position that it was retracted to (e.g. 2000). To align the Target Position on the master and the PosRef of the driver.

Note: The axis will not physically move as the motionmode for the axis is not put to 19 (motionmode used to move in EtherCAT Mode)

Once the TargetPosition and the PosRef of the driver is aligned, we can trigger this function to turn motionmode = 19 and prepare for any movement from the master thereafter.

NOTE: Only call this once the TargetPosition and the posref of the driver is aligned.

Absolute Move to Target Point 2 / datum point.

2.3.3.1 Customized Function Block

The example shown above is the “MAIN Program”. This section will show how the AGenData and AGenDataStatus can be integrated inside the customized function block. Only “AGenDataForceControl”, “AlignMasterSlave” and “UserProgramReset” will be shown in this section. The rest of the function blocks are using standard PLCOpen function blocks.

AGenDataForceControl



```

1 FUNCTION_BLOCK AGenDataForceControl
2 VAR_INPUT
3   bForceControlExecute      : BOOL; // tie with aagenda[1], not physically link them but we can control where we read this booleans and control teh GVL.XX values
4   nCurrPosTh                : DINT; // tie with aagenda[7]
5   nCurrCurrTh               : DINT; // tie with aagenda[2]
6   nJoggingSpeed             : DINT; // tie with aagenda[8]
7 END_VAR
8 VAR_OUTPUT
9   bForceControlCompleted    : BOOL; // tie wi AGenData[4]
10  bForceControlBusy         : BOOL;
11  bForceControlActive       : BOOL;
12  bForceControlError        : BOOL;
13
14 fbExecute (CLK := bForceControlExecute);
15
16 CASE nState OF
17   0: (*INIT*)
18     IF NOT bForceControlExecute THEN
19       bForceControlCompleted := FALSE;
20       bForceControlBusy      := FALSE;
21       bForceControlActive    := FALSE;
22     END_IF
23
24     IF fbExecute.Q THEN
25       bForceControlBusy := TRUE;
26       nState := nState + 1;
27     END_IF
28
29     1: (*Toggling the GVL.ForceControlExecute to turn on Force Control Sequence in User Program*)
30       // Assuming that the PositionThreshold and Current Threshold has been set.
31       GVL.ForceControlExecute := 1;
32
33       IF GVL.ForceControlExecuteStatus = 1 THEN
34         bForceControlActive := TRUE;
35       ELSE
36         bForceControlActive := FALSE;
37       END_IF
38
39       IF bForceControlActive THEN
40         nState := nState + 1;
41
42         bForceControlActive := FALSE;
43       END_IF
44
45       IF bForceControlActive THEN
46         nState := nState + 1;
47
48         bForceControlActive := FALSE;
49       END_IF
50
51       IF bForceControlActive THEN
52         nState := nState + 1;
53
54         bForceControlActive := FALSE;
55       END_IF
56
57       IF bForceControlActive THEN
58         nState := nState + 1;
59
60         bForceControlActive := FALSE;
61       END_IF
62
63       IF bForceControlActive THEN
64         nState := nState + 1;
65
66         bForceControlActive := FALSE;
67       END_IF
68
69       IF bForceControlActive THEN
70         nState := nState + 1;
71
72         bForceControlActive := FALSE;
73       END_IF
74
75       IF bForceControlActive THEN
76         nState := nState + 1;
77
78         bForceControlActive := FALSE;
79       END_IF
80
81       IF bForceControlActive THEN
82         nState := nState + 1;
83
84         bForceControlActive := FALSE;
85       END_IF
86
87       IF bForceControlActive THEN
88         nState := nState + 1;
89
90         bForceControlActive := FALSE;
91       END_IF
92
93       IF bForceControlActive THEN
94         nState := nState + 1;
95
96         bForceControlActive := FALSE;
97       END_IF
98
99       IF bForceControlActive THEN
100        nState := nState + 1;
101
102        bForceControlActive := FALSE;
103      END_IF
104
105      IF bForceControlActive THEN
106        nState := nState + 1;
107
108        bForceControlActive := FALSE;
109      END_IF
110
111      IF bForceControlActive THEN
112        nState := nState + 1;
113
114        bForceControlActive := FALSE;
115      END_IF
116
117      IF bForceControlActive THEN
118        nState := nState + 1;
119
120        bForceControlActive := FALSE;
121      END_IF
122
123      IF bForceControlActive THEN
124        nState := nState + 1;
125
126        bForceControlActive := FALSE;
127      END_IF
128
129      IF bForceControlActive THEN
130        nState := nState + 1;
131
132        bForceControlActive := FALSE;
133      END_IF
134
135      IF bForceControlActive THEN
136        nState := nState + 1;
137
138        bForceControlActive := FALSE;
139      END_IF
140
141      IF bForceControlActive THEN
142        nState := nState + 1;
143
144        bForceControlActive := FALSE;
145      END_IF
146
147      IF bForceControlActive THEN
148        nState := nState + 1;
149
150        bForceControlActive := FALSE;
151      END_IF
152
153      IF bForceControlActive THEN
154        nState := nState + 1;
155
156        bForceControlActive := FALSE;
157      END_IF
158
159      IF bForceControlActive THEN
160        nState := nState + 1;
161
162        bForceControlActive := FALSE;
163      END_IF
164
165      IF bForceControlActive THEN
166        nState := nState + 1;
167
168        bForceControlActive := FALSE;
169      END_IF
170
171      IF bForceControlActive THEN
172        nState := nState + 1;
173
174        bForceControlActive := FALSE;
175      END_IF
176
177      IF bForceControlActive THEN
178        nState := nState + 1;
179
180        bForceControlActive := FALSE;
181      END_IF
182
183      IF bForceControlActive THEN
184        nState := nState + 1;
185
186        bForceControlActive := FALSE;
187      END_IF
188
189      IF bForceControlActive THEN
190        nState := nState + 1;
191
192        bForceControlActive := FALSE;
193      END_IF
194
195      IF bForceControlActive THEN
196        nState := nState + 1;
197
198        bForceControlActive := FALSE;
199      END_IF
200
201      IF bForceControlActive THEN
202        nState := nState + 1;
203
204        bForceControlActive := FALSE;
205      END_IF
206
207      IF bForceControlActive THEN
208        nState := nState + 1;
209
210        bForceControlActive := FALSE;
211      END_IF
212
213      IF bForceControlActive THEN
214        nState := nState + 1;
215
216        bForceControlActive := FALSE;
217      END_IF
218
219      IF bForceControlActive THEN
220        nState := nState + 1;
221
222        bForceControlActive := FALSE;
223      END_IF
224
225      IF bForceControlActive THEN
226        nState := nState + 1;
227
228        bForceControlActive := FALSE;
229      END_IF
230
231      IF bForceControlActive THEN
232        nState := nState + 1;
233
234        bForceControlActive := FALSE;
235      END_IF
236
237      IF bForceControlActive THEN
238        nState := nState + 1;
239
240        bForceControlActive := FALSE;
241      END_IF
242
243      IF bForceControlActive THEN
244        nState := nState + 1;
245
246        bForceControlActive := FALSE;
247      END_IF
248
249      IF bForceControlActive THEN
250        nState := nState + 1;
251
252        bForceControlActive := FALSE;
253      END_IF
254
255      IF bForceControlActive THEN
256        nState := nState + 1;
257
258        bForceControlActive := FALSE;
259      END_IF
260
261      IF bForceControlActive THEN
262        nState := nState + 1;
263
264        bForceControlActive := FALSE;
265      END_IF
266
267      IF bForceControlActive THEN
268        nState := nState + 1;
269
270        bForceControlActive := FALSE;
271      END_IF
272
273      IF bForceControlActive THEN
274        nState := nState + 1;
275
276        bForceControlActive := FALSE;
277      END_IF
278
279      IF bForceControlActive THEN
280        nState := nState + 1;
281
282        bForceControlActive := FALSE;
283      END_IF
284
285      IF bForceControlActive THEN
286        nState := nState + 1;
287
288        bForceControlActive := FALSE;
289      END_IF
290
291      IF bForceControlActive THEN
292        nState := nState + 1;
293
294        bForceControlActive := FALSE;
295      END_IF
296
297      IF bForceControlActive THEN
298        nState := nState + 1;
299
300        bForceControlActive := FALSE;
301      END_IF
302
303      IF bForceControlActive THEN
304        nState := nState + 1;
305
306        bForceControlActive := FALSE;
307      END_IF
308
309      IF bForceControlActive THEN
310        nState := nState + 1;
311
312        bForceControlActive := FALSE;
313      END_IF
314
315      IF bForceControlActive THEN
316        nState := nState + 1;
317
318        bForceControlActive := FALSE;
319      END_IF
320
321      IF bForceControlActive THEN
322        nState := nState + 1;
323
324        bForceControlActive := FALSE;
325      END_IF
326
327      IF bForceControlActive THEN
328        nState := nState + 1;
329
330        bForceControlActive := FALSE;
331      END_IF
332
333      IF bForceControlActive THEN
334        nState := nState + 1;
335
336        bForceControlActive := FALSE;
337      END_IF
338
339      IF bForceControlActive THEN
340        nState := nState + 1;
341
342        bForceControlActive := FALSE;
343      END_IF
344
345      IF bForceControlActive THEN
346        nState := nState + 1;
347
348        bForceControlActive := FALSE;
349      END_IF
350
351      IF bForceControlActive THEN
352        nState := nState + 1;
353
354        bForceControlActive := FALSE;
355      END_IF
356
357      IF bForceControlActive THEN
358        nState := nState + 1;
359
360        bForceControlActive := FALSE;
361      END_IF
362
363      IF bForceControlActive THEN
364        nState := nState + 1;
365
366        bForceControlActive := FALSE;
367      END_IF
368
369      IF bForceControlActive THEN
370        nState := nState + 1;
371
372        bForceControlActive := FALSE;
373      END_IF
374
375      IF bForceControlActive THEN
376        nState := nState + 1;
377
378        bForceControlActive := FALSE;
379      END_IF
380
381      IF bForceControlActive THEN
382        nState := nState + 1;
383
384        bForceControlActive := FALSE;
385      END_IF
386
387      IF bForceControlActive THEN
388        nState := nState + 1;
389
390        bForceControlActive := FALSE;
391      END_IF
392
393      IF bForceControlActive THEN
394        nState := nState + 1;
395
396        bForceControlActive := FALSE;
397      END_IF
398
399      IF bForceControlActive THEN
400        nState := nState + 1;
401
402        bForceControlActive := FALSE;
403      END_IF
404
405      IF bForceControlActive THEN
406        nState := nState + 1;
407
408        bForceControlActive := FALSE;
409      END_IF
410
411      IF bForceControlActive THEN
412        nState := nState + 1;
413
414        bForceControlActive := FALSE;
415      END_IF
416
417      IF bForceControlActive THEN
418        nState := nState + 1;
419
420        bForceControlActive := FALSE;
421      END_IF
422
423      IF bForceControlActive THEN
424        nState := nState + 1;
425
426        bForceControlActive := FALSE;
427      END_IF
428
429      IF bForceControlActive THEN
430        nState := nState + 1;
431
432        bForceControlActive := FALSE;
433      END_IF
434
435      IF bForceControlActive THEN
436        nState := nState + 1;
437
438        bForceControlActive := FALSE;
439      END_IF
440
441      IF bForceControlActive THEN
442        nState := nState + 1;
443
444        bForceControlActive := FALSE;
445      END_IF
446
447      IF bForceControlActive THEN
448        nState := nState + 1;
449
450        bForceControlActive := FALSE;
451      END_IF
452
453      IF bForceControlActive THEN
454        nState := nState + 1;
455
456        bForceControlActive := FALSE;
457      END_IF
458
459      IF bForceControlActive THEN
460        nState := nState + 1;
461
462        bForceControlActive := FALSE;
463      END_IF
464
465      IF bForceControlActive THEN
466        nState := nState + 1;
467
468        bForceControlActive := FALSE;
469      END_IF
470
471      IF bForceControlActive THEN
472        nState := nState + 1;
473
474        bForceControlActive := FALSE;
475      END_IF
476
477      IF bForceControlActive THEN
478        nState := nState + 1;
479
480        bForceControlActive := FALSE;
481      END_IF
482
483      IF bForceControlActive THEN
484        nState := nState + 1;
485
486        bForceControlActive := FALSE;
487      END_IF
488
489      IF bForceControlActive THEN
490        nState := nState + 1;
491
492        bForceControlActive := FALSE;
493      END_IF
494
495      IF bForceControlActive THEN
496        nState := nState + 1;
497
498        bForceControlActive := FALSE;
499      END_IF
500
501      IF bForceControlActive THEN
502        nState := nState + 1;
503
504        bForceControlActive := FALSE;
505      END_IF
506
507      IF bForceControlActive THEN
508        nState := nState + 1;
509
510        bForceControlActive := FALSE;
511      END_IF
512
513      IF bForceControlActive THEN
514        nState := nState + 1;
515
516        bForceControlActive := FALSE;
517      END_IF
518
519      IF bForceControlActive THEN
520        nState := nState + 1;
521
522        bForceControlActive := FALSE;
523      END_IF
524
525      IF bForceControlActive THEN
526        nState := nState + 1;
527
528        bForceControlActive := FALSE;
529      END_IF
530
531      IF bForceControlActive THEN
532        nState := nState + 1;
533
534        bForceControlActive := FALSE;
535      END_IF
536
537      IF bForceControlActive THEN
538        nState := nState + 1;
539
540        bForceControlActive := FALSE;
541      END_IF
542
543      IF bForceControlActive THEN
544        nState := nState + 1;
545
546        bForceControlActive := FALSE;
547      END_IF
548
549      IF bForceControlActive THEN
550        nState := nState + 1;
551
552        bForceControlActive := FALSE;
553      END_IF
554
555      IF bForceControlActive THEN
556        nState := nState + 1;
557
558        bForceControlActive := FALSE;
559      END_IF
560
561      IF bForceControlActive THEN
562        nState := nState + 1;
563
564        bForceControlActive := FALSE;
565      END_IF
566
567      IF bForceControlActive THEN
568        nState := nState + 1;
569
570        bForceControlActive := FALSE;
571      END_IF
572
573      IF bForceControlActive THEN
574        nState := nState + 1;
575
576        bForceControlActive := FALSE;
577      END_IF
578
579      IF bForceControlActive THEN
580        nState := nState + 1;
581
582        bForceControlActive := FALSE;
583      END_IF
584
585      IF bForceControlActive THEN
586        nState := nState + 1;
587
588        bForceControlActive := FALSE;
589      END_IF
590
591      IF bForceControlActive THEN
592        nState := nState + 1;
593
594        bForceControlActive := FALSE;
595      END_IF
596
597      IF bForceControlActive THEN
598        nState := nState + 1;
599
600        bForceControlActive := FALSE;
601      END_IF
602
603      IF bForceControlActive THEN
604        nState := nState + 1;
605
606        bForceControlActive := FALSE;
607      END_IF
608
609      IF bForceControlActive THEN
610        nState := nState + 1;
611
612        bForceControlActive := FALSE;
613      END_IF
614
615      IF bForceControlActive THEN
616        nState := nState + 1;
617
618        bForceControlActive := FALSE;
619      END_IF
620
621      IF bForceControlActive THEN
622        nState := nState + 1;
623
624        bForceControlActive := FALSE;
625      END_IF
626
627      IF bForceControlActive THEN
628        nState := nState + 1;
629
630        bForceControlActive := FALSE;
631      END_IF
632
633      IF bForceControlActive THEN
634        nState := nState + 1;
635
636        bForceControlActive := FALSE;
637      END_IF
638
639      IF bForceControlActive THEN
640        nState := nState + 1;
641
642        bForceControlActive := FALSE;
643      END_IF
644
645      IF bForceControlActive THEN
646        nState := nState + 1;
647
648        bForceControlActive := FALSE;
649      END_IF
650
651      IF bForceControlActive THEN
652        nState := nState + 1;
653
654        bForceControlActive := FALSE;
655      END_IF
656
657      IF bForceControlActive THEN
658        nState := nState + 1;
659
660        bForceControlActive := FALSE;
661      END_IF
662
663      IF bForceControlActive THEN
664        nState := nState + 1;
665
666        bForceControlActive := FALSE;
667      END_IF
668
669      IF bForceControlActive THEN
670        nState := nState + 1;
671
672        bForceControlActive := FALSE;
673      END_IF
674
675      IF bForceControlActive THEN
676        nState := nState + 1;
677
678        bForceControlActive := FALSE;
679      END_IF
680
681      IF bForceControlActive THEN
682        nState := nState + 1;
683
684        bForceControlActive := FALSE;
685      END_IF
686
687      IF bForceControlActive THEN
688        nState := nState + 1;
689
690        bForceControlActive := FALSE;
691      END_IF
692
693      IF bForceControlActive THEN
694        nState := nState + 1;
695
696        bForceControlActive := FALSE;
697      END_IF
698
699      IF bForceControlActive THEN
700        nState := nState + 1;
701
702        bForceControlActive := FALSE;
703      END_IF
704
705      IF bForceControlActive THEN
706        nState := nState + 1;
707
708        bForceControlActive := FALSE;
709      END_IF
710
711      IF bForceControlActive THEN
712        nState := nState + 1;
713
714        bForceControlActive := FALSE;
715      END_IF
716
717      IF bForceControlActive THEN
718        nState := nState + 1;
719
720        bForceControlActive := FALSE;
721      END_IF
722
723      IF bForceControlActive THEN
724        nState := nState + 1;
725
726        bForceControlActive := FALSE;
727      END_IF
728
729      IF bForceControlActive THEN
730        nState := nState + 1;
731
732        bForceControlActive := FALSE;
733      END_IF
734
735      IF bForceControlActive THEN
736        nState := nState + 1;
737
738        bForceControlActive := FALSE;
739      END_IF
740
741      IF bForceControlActive THEN
742        nState := nState + 1;
743
744        bForceControlActive := FALSE;
745      END_IF
746
747      IF bForceControlActive THEN
748        nState := nState + 1;
749
750        bForceControlActive := FALSE;
751      END_IF
752
753      IF bForceControlActive THEN
754        nState := nState + 1;
755
756        bForceControlActive := FALSE;
757      END_IF
758
759      IF bForceControlActive THEN
760        nState := nState + 1;
761
762        bForceControlActive := FALSE;
763      END_IF
764
765      IF bForceControlActive THEN
766        nState := nState + 1;
767
768        bForceControlActive := FALSE;
769      END_IF
770
771      IF bForceControlActive THEN
772        nState := nState + 1;
773
774        bForceControlActive := FALSE;
775      END_IF
776
777      IF bForceControlActive THEN
778        nState := nState + 1;
779
780        bForceControlActive := FALSE;
781      END_IF
782
783      IF bForceControlActive THEN
784        nState := nState + 1;
785
786        bForceControlActive := FALSE;
787      END_IF
788
789      IF bForceControlActive THEN
790        nState := nState + 1;
791
792        bForceControlActive := FALSE;
793      END_IF
794
795      IF bForceControlActive THEN
796        nState := nState + 1;
797
798        bForceControlActive := FALSE;
799      END_IF
800
801      IF bForceControlActive THEN
802        nState := nState + 1;
803
804        bForceControlActive := FALSE;
805      END_IF
806
807      IF bForceControlActive THEN
808        nState := nState + 1;
809
810        bForceControlActive := FALSE;
811      END_IF
812
813      IF bForceControlActive THEN
814        nState := nState + 1;
815
816        bForceControlActive := FALSE;
817      END_IF
818
819      IF bForceControlActive THEN
820        nState := nState + 1;
821
822        bForceControlActive := FALSE;
823      END_IF
824
825      IF bForceControlActive THEN
826        nState := nState + 1;
827
828        bForceControlActive := FALSE;
829      END_IF
830
831      IF bForceControlActive THEN
832        nState := nState + 1;
833
834        bForceControlActive := FALSE;
835      END_IF
836
837      IF bForceControlActive THEN
838        nState := nState + 1;
839
840        bForceControlActive := FALSE;
841      END_IF
842
843      IF bForceControlActive THEN
844        nState := nState + 1;
845
846        bForceControlActive := FALSE;
847      END_IF
848
849      IF bForceControlActive THEN
850        nState := nState + 1;
851
852        bForceControlActive := FALSE;
853      END_IF
854
855      IF bForceControlActive THEN
856        nState := nState + 1;
857
858        bForceControlActive := FALSE;
859      END_IF
860
861      IF bForceControlActive THEN
862        nState := nState + 1;
863
864        bForceControlActive := FALSE;
865      END_IF
866
867      IF bForceControlActive THEN
868        nState := nState + 1;
869
870        bForceControlActive := FALSE;
871      END_IF
872
873      IF bForceControlActive THEN
874        nState := nState + 1;
875
876        bForceControlActive := FALSE;
877      END_IF
878
879      IF bForceControlActive THEN
880        nState := nState + 1;
881
882        bForceControlActive := FALSE;
883      END_IF
884
885      IF bForceControlActive THEN
886        nState := nState + 1;
887
888        bForceControlActive := FALSE;
889      END_IF
890
891      IF bForceControlActive THEN
892        nState := nState + 1;
893
894        bForceControlActive := FALSE;
895      END_IF
896
897      IF bForceControlActive THEN
898        nState := nState + 1;
899
900        bForceControlActive := FALSE;
901      END_IF
902
903      IF bForceControlActive THEN
904        nState := nState + 1;
905
906        bForceControlActive := FALSE;
907      END_IF
908
909      IF bForceControlActive THEN
910        nState := nState + 1;
911
912        bForceControlActive := FALSE;
913      END_IF
914
915      IF bForceControlActive THEN
916        nState := nState + 1;
917
918        bForceControlActive := FALSE;
919      END_IF
920
921      IF bForceControlActive THEN
922        nState := nState + 1;
923
924        bForceControlActive := FALSE;
925      END_IF
926
927      IF bForceControlActive THEN
928        nState := nState + 1;
929
930        bForceControlActive := FALSE;
931      END_IF
932
933      IF bForceControlActive THEN
934        nState := nState + 1;
935
936        bForceControlActive := FALSE;
937      END_IF
938
939      IF bForceControlActive THEN
940        nState := nState + 1;
941
942        bForceControlActive := FALSE;
943      END_IF
944
945      IF bForceControlActive THEN
946        nState := nState + 1;
947
948        bForceControlActive := FALSE;
949      END_IF
950
951      IF bForceControlActive THEN
952        nState := nState + 1;
953
954        bForceControlActive := FALSE;
955      END_IF
956
957      IF bForceControlActive THEN
958        nState := nState + 1;
959
960        bForceControlActive := FALSE;
961      END_IF
962
963      IF bForceControlActive THEN
964        nState := nState + 1;
965
966        bForceControlActive := FALSE;
967      END_IF
968
969      IF bForceControlActive THEN
970        nState := nState + 1;
971
972        bForceControlActive := FALSE;
973      END_IF
974
975      IF bForceControlActive THEN
976        nState := nState + 1;
977
978        bForceControlActive := FALSE;
979      END_IF
980
981      IF bForceControlActive THEN
982        nState := nState + 1;
983
984        bForceControlActive := FALSE;
985      END_IF
986
987      IF bForceControlActive THEN
988        nState := nState + 1;
989
990        bForceControlActive := FALSE;
991      END_IF
992
993      IF bForceControlActive THEN
994        nState := nState + 1;
995
996        bForceControlActive := FALSE;
997      END_IF
998
999      IF bForceControlActive THEN
1000       nState := nState + 1;
1001
1002       bForceControlActive := FALSE;
1003     END_IF
1004
1005     2: (*Waiting for GVL.ForceControlCompletedStatus*)
1006       IF (GVL.ForceControlCompletedStatus = 1) AND (GVL.ForceControlExecuteStatus = 0) THEN
1007         bForceControlActive := FALSE;
1008         bForceControlCompleted := TRUE;
1009
1010         nState := nState + 1;
1011       ELSEIF NOT (GVL.UserProgramStatus = 1) THEN // indicating that somewhere in the user program has fault
1012         bForceControlError := TRUE; // think of another way to reset to error
1013
1014         nState := 20;
1015       ELSE
1016         bForceControlCompleted := FALSE;
1017         nState := nState;
1018       END_IF
1019
1020     3: (* Waiting for Falling Edge to reset the parameters*)
1021       IF NOT bForceControlExecute THEN
1022         bForceControlCompleted := FALSE;
1023
1024     3: (* Waiting for Falling Edge to reset the parameters*)
1025       IF NOT bForceControlExecute THEN
1026         bForceControlCompleted := FALSE;
1027         bForceControlBusy := FALSE;
1028         bForceControlActive := FALSE;
1029
1030         GVL.ForceControlExecute := 0;
1031
1032         nState := 0;
1033       ELSE
1034         nState := nState;
1035       END_IF
1036 END_CASE
1037 END_FUNCTION_BLOCK
  
```

VAR_INPUT / VAR_OUTPUT

Rising Edge Trigger

Initialization / Polling for Trigger

Assigning AGenData[1] = 1

Polling for AGenDataStatus[1] to assert "Active"

Polling for AGenDataStatus[4] to determine that the force control has been completed and retracted back to the retracted target else if there is any userprogram error it will enter to fault state.

Waiting for falling edge of "bForceControlExecute" to reset variables and return to state "0" of the function block

TwinCAT / PLC Programming Example

AlignMasterSlave

```

1 FUNCTION_BLOCK AlignMasterSlave
2 VAR_INPUT
3   bAlignMasterSlaveExecute : BOOL; // tie with sdata[3], not physically link them but we can control where we read this booleans and control teh GVL.XX values
4 END_VAR
5 VAR_OUTPUT
6   bAlignMasterSlaveCompleted : BOOL;
7   bAlignMasterSlaveBusy : BOOL;
8   bAlignMasterSlaveActive : BOOL;
9   bAlignMasterSlaveError : BOOL;
10 END_VAR
11 VAR
12   fbExecute : R_TRIG;
13
14 fbExecute(CLK := bAlignMasterSlaveExecute);
15 fbAlignMasterSlaveDone(CLK := bAlignMasterSlaveActive);
16
17 CASE nState OF
18   0: (*EXIT*)
19     IF NOT bAlignMasterSlaveExecute THEN
20       bAlignMasterSlaveCompleted := FALSE;
21       bAlignMasterSlaveBusy := FALSE;
22       bAlignMasterSlaveActive := FALSE;
23     END_IF
24   IF fbExecute.Q THEN
25     bAlignMasterSlaveBusy := TRUE;
26     nState := nState + 1;
27   END_IF
28
29 1: (*Toggling GVL.EnableCSPMotionMode to align back with master , but first master will have to do a virtual move to the axis current position first before calling this function*)
30 // Or give a TargetPosition to move to let say Point A TargetPosition is 1000. Do AbsMove virtually before triggering this function on the master, and in the user program trigger a absolute m
31 // to Point A Target Position
32 GVL.EnableCSPMotionMode := 1;
33
34 IF (GVL.EnableCSPMotionModeStatus = 1) THEN (*Currently switching to drive to CSP Mode*)
35   bAlignMasterSlaveActive := TRUE;
36 ELSE
37   bAlignMasterSlaveActive := FALSE;
38 END_IF
39
40 // Or give a TargetPosition to move to let say Point A TargetPosition is 1000. Do AbsMove virtually before triggering this function on the master, and in the user program trigger a al
41 // to Point A Target Position
42 GVL.EnableCSPMotionMode := 1;
43
44 IF (GVL.EnableCSPMotionModeStatus = 1) THEN (*Currently switching to drive to CSP Mode*)
45   bAlignMasterSlaveActive := TRUE;
46 ELSE
47   bAlignMasterSlaveActive := FALSE;
48 END_IF
49
50 IF fbAlignMasterSlaveDone.Q THEN
51   bAlignMasterSlaveCompleted := TRUE;
52   nState := nState + 1;
53 ELSIF NOT (GVL.UserProgramStatus = 1) THEN
54   bAlignMasterSlaveError := TRUE;
55   nState := 20;
56 ELSE
57   nState := nState;
58 END_IF
59
60 2:
61 IF NOT bAlignMasterSlaveExecute THEN
62   bAlignMasterSlaveCompleted := FALSE;
63   bAlignMasterSlaveBusy := FALSE;
64   bAlignMasterSlaveActive := FALSE;
65   GVL.EnableCSPMotionMode := 0;
66   nState := 0;
67 END_IF
68 END_CASE

```

VAR_INPUT / VAR_OUTPUT

Initialization / Polling for Trigger

Assigning AGenData[3] = 1

Checking for AGenDataStatus[3] to indicate "Active" and also checking the falling edge of "Active" to indicate that it has been completed.

Waiting for falling edge of "bAlignMasterSlaveExecute" to reset variables and return to state "0" of the function block.

UserProgramReset

```

1 FUNCTION_BLOCK UserProgramReset
2 VAR_INPUT
3   bUserProgramResetExecute : BOOL;
4 END_VAR
5 VAR_OUTPUT
6   bUserProgramResetCompleted : BOOL;
7   bUserProgramResetBusy : BOOL;
8   bUserProgramResetActive : BOOL;
9   bUserProgramResetError : BOOL;
10 END_VAR
11 VAR
12   fbExecute : R_TRIG;

```

```

1 fbExecute(CLK := bUserProgramResetExecute);
2 fbUserProgramResetDone(CLK := bUserProgramResetActive);
3
4 CASE nState OF
5   0: (*INIT*)
6     IF NOT bUserProgramResetExecute THEN
7       bUserProgramResetCompleted := FALSE;
8       bUserProgramResetBusy := FALSE;
9       bUserProgramResetActive := FALSE;
10    END_IF
11
12    IF fbExecute.Q THEN
13      bUserProgramResetBusy := TRUE;
14
15      nState := nState + 1;
16    END_IF
17
18    1:
19      GVL.UserProgramReset := 1;
20
21      IF GVL.UserProgramResetStatus = 1 THEN
22        bUserProgramResetActive := TRUE;
23      ELSE
24        bUserProgramResetActive := FALSE;
25      END_IF
26
27      IF bUserProgramResetActive THEN
28        nState := nState + 1;

```

```

        nState := nState + 1;
    ELSE
        nState := nState;
    END_IF
2:
    IF GVL.UserProgramStatus = 1 THEN
        bUserProgramResetCompleted := TRUE;
        nState := nState + 1;
    ELSE
        bUserProgramResetCompleted := FALSE;
        nState := nState;
    END_IF
3: (* Waiting for Falling Edge to reset the parameters *)
    IF NOT bUserProgramResetExecute THEN
        bUserProgramResetCompleted := FALSE;
        bUserProgramResetBusy := FALSE;
        bUserProgramResetActive := FALSE;

        GVL.UserProgramReset := 0;

        nState := 0;
    ELSE
        nState := nState;
    END_IF
END_CASE

```

VAR_INPUT / VAR_OUTPUT

Initialization / Polling for Trigger

Assigning AGenData[6] = 1

Checking for AGenDataStatus[6] to indicate "Active" and also checking the falling edge of "Active" to indicate that it has been completed.

Waiting for falling edge of "bUserProgramResetExecute" to reset variables and return to state "0" of the function block.

3 Using PLCOpen Torque Control (CST Mode)

The conventional way to do force control over EtherCAT is to use the provided Torque Control function block in the PLCOpen. The difference between using this method and the method above is that the sampling of the switching of mode is tied to the cycle time of the PLC task whereas the User Program method is sampling at the driver interrupt which is at 62.5us. There are certain conditions or SDO objects to be set before enabling the torque control to enter current/force control mode in the drive and that will be covered in the description of related EtherCAT Objects.

3.1 Description of Related EtherCAT Objects

This section touches on the related EtherCAT Objects (on top of the mandatory ones) that can be / should be mapped to the PDO to ensure cyclic exchanging of information between the master and controller. The mandatory EtherCAT Objects that should have been mapped are “StatusWord”, “Position Actual Value”, “ControlWord” and “Target Position”.

3.1.1 Object 2410_h: Analog Input 1 Value

This object shall indicate the value that is connected to the analog input port of the driver i.e. force sensor. Calibration should have been done on the PCSuite UI. The value that is reported back will be in terms of force units.

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Integer32
Default Value	00000000 _h

3.1.2 Object 2810_h: CST Force Loop Enable

This object shall indicate if current / force mode is to be used. “0” indicates that current mode shall be used and “1” indicates force mode. By default, the value of the object is set as “0”, if user has analog force feedback connected, this object should set as “1”.

This object can be set through SDO and does not be to be included in the cyclic exchange. If the master allows startup configurations, this object can be set during the start-up parameters.

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Unsigned8
Default Value	00 _h

3.1.3 Object 2811_h: CST Command Interpolation

This object shall indicate the configured value to be used to interpolate the CST Command when force mode is enabled. By default, the value of the object is set as “1”, if the user finds that 1mv/10 force units is too coarse, they can further interpolate by changing this to a value of “2” or any other value. It is scaled down by 1/ (values set in this object).

Homing Methods Example

This object can be set through SDO and does not be to be included in the cyclic exchange. If the master allows startup configurations, this object can be set during the start-up parameters.

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Unsigned8
Default Value	00 _h

3.1.4 Object 6071_h: Target Torque

This object shall indicate the configured input value for the torque controller in profile torque mode/cst mode. The value shall be given per thousand of rated torque.

If current mode (0x2810: “CST Force Loop Enable” = 0) is selected, the Target Torque will be given per thousand of the continuous torque/current set in the driver. (E.g. if the continuous current in the driver is set as 2000mA, 5.0% of Target Torque would indicate a current reference of 100mA and 0.1% of it would indicate 2mA).

If force mode (0x2810: “CST Force Loop Enable” = 1) is selected, the Target Torque will be given per thousand of +- 10V (10000mV) as normally there will be an analog force feedback connected to the driver and during the initial set up it has already done certain calibration to indicate that 1mV represents 1g/5g or 10mv/ represents 1g/5g. The force unit is not necessary to be in g, it will be according to the calibration that was set. Hence, 5% of the Target Torque would give a force reference of 500 force units and 0.1% of it would give 10 force units. It is also possible to scale down even more by changing the scaling factor (0x2810: “CST Command Interpolation”)

Attribute	Value
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Integer16
Default Value	0000 _h

3.1.5 Object 6077_h: Torque Actual Value

This object shall provide the actual value of the torque. It shall correspond to the instantaneous torque in the motor. The value shall be given per thousand of rated torque. Currently, there is no torque feedback in our drive and since $T = k_t \cdot I$ and assuming that the motor constant is static, the calculation of the torque feedback will be in terms of the percentage of the continuous current that is currently used.

If continuous current is set as 2000ma and 500mA is currently being used, a value of 25% would be reported back in this object.

If force mode is used, please refer to object 0x2410 “Analog Input Value 1” to see the force units that is being reported back.

Attribute	Value
-----------	-------

Homing Methods Example

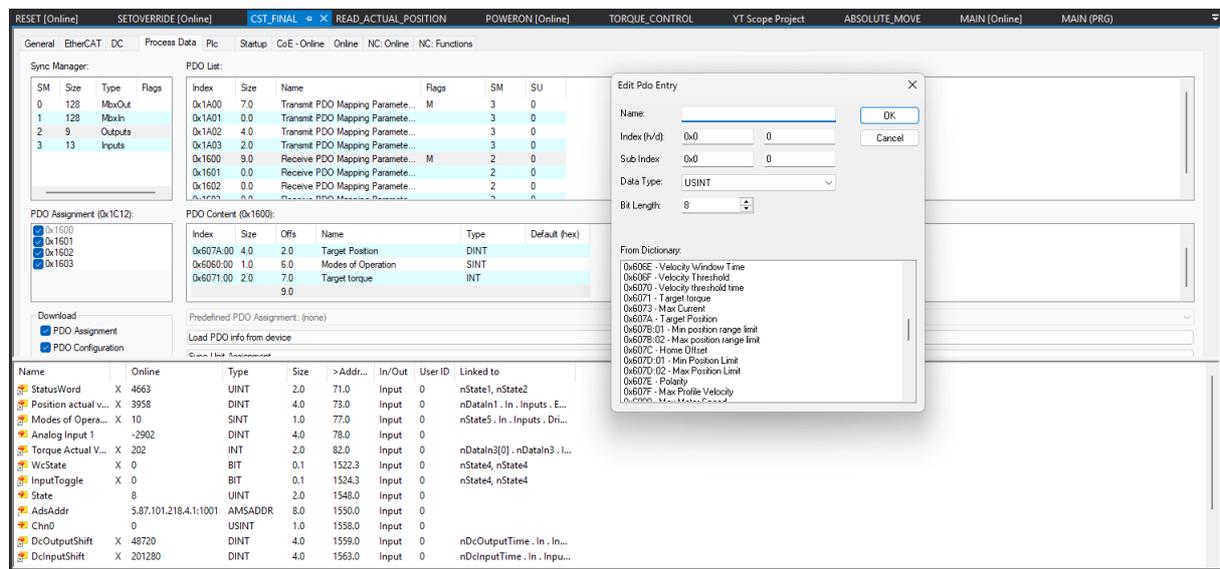
Sub-Index	00 _h
Access	rw
PDO Mapping	Yes
Value Range	Interger16
Default Value	0000 _h

3.2 TwinCAT / PLC Programming Example

3.2.1 CST Current Loop

3.2.1.1 Modifying PDO List

It is important to map the Target Torque (0x6071_h) and Torque Actual Value (0x6077_h) object into the PDO list to ensure the cyclic exchange of information between the master and the slave. Descriptions of the individual objects are described in the section above and how they are being used in different modes. Target Torque (0x6071_h) and Torque Actual Value (0x6077_h) are not mapped in the PDO list by default hence users have to manually add the objects into the PDO configuration list.



3.2.1.2 Writing POU

```

(*Power On*)
axisPower(axis_Test := axis_Test, bPowerEnable := bPowerEnable, bPowerStatus => bPowerStatus, bPowerBusy => bPowerBusy, bPowerError => bPowerError, nPowerErrorID => nPowerErrorID);
(*Reset*)
axisReset(axis_Test := axis_Test, bResetExecute := bResetExecute, bResetDone => bResetDone, bResetBusy => bResetBusy, bResetError => bResetError, nResetErrorID => nResetErrorID);
(*Move Absolute*)
axisMoveAbsolute(axis_Test := axis_Test, bMoveAbsoluteExecute := bMoveAbsoluteExecute, ABORT_DUE_TO_CST := ABORT_DUE_TO_CST, nMoveAbsoluteErrorID => nMoveAbsoluteErrorID);
(*Read Actual Position*)
axisReadPosition(axis_Test := axis_Test, bReadActualPosEnable := TRUE, bReadActualPosValid => bReadActualPosValid, bReadActualPosBusy => bReadActualPosBusy, bReadActualPosError => bReadActualPosError, nReadActualPosErrorID => nReadActualPosErrorID, nActualPosition => nActualPosition);
(*Set Speed Override*)
axisSpeedOverride(axis_Test := axis_Test, bSetOverrideEnable := bSetOverrideExecute);
(*Torque Control*)
axisTorqueControl(axis_Test := axis_Test, bTorqueControlExecute := bTorqueControlExecute, nDesiredTorque := lTorque, nDesiredTorqueRamp := lTorqueRamp);
CASE seq1State OF
  SEQ1STATE_NEW_SEQ1_RESET :
    (*Initialization Logic*)
    bResetExecute := TRUE;
    bPowerEnable := FALSE;
    IF bResetDone THEN
      bResetExecute := FALSE;
      seq1State := SEQ1STATE_NEW_SEQ1_POWER_ENABLE;
    ELSEIF bResetError THEN
      bResetExecute := FALSE;
      nResetErrorID := nResetErrorID;
    END_IF;
  SEQ1STATE_NEW_SEQ1_FAULT :
    END_IF;
END_CASE
  
```

Instance of function blocks used

Reset / Initialization Sequence

Homing Methods Example

```

SEQ1STATE_NEW.SEQ1_POWER_ENABLE:
(*'Axis Power')
  bPowerEnable := TRUE;

  IF bPowerStatus THEN
    seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
  ELSIF bPowerError THEN
    bPowerEnable := FALSE;

    seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
  END_IF

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1:
(*'Setting Target 1 Parameters')
  axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
  axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
  axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
  axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
  axisMoveAbsolute.nTargetJerk := nTargetJerk_1;

  bMoveAbsoluteExecute := TRUE;

  seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1;

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1:
  IF axisReadPosition.nActualPosition > -1000 THEN //-100000 THEN
    axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1:
  IF axisReadPosition.nActualPosition > -1000 THEN //-100000 THEN
    axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;
    bSpeedOverrideExecute := TRUE;
  END_IF

  IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
    seq1State := SEQ1STATE_NEW.SEQ1_CST_MODE;
  END_IF

SEQ1STATE_NEW.SEQ1_CST_MODE:
  IF axis_Test.NcToPlc.ActTorque >= 20 THEN
    bTorqueControlExecute := TRUE;
    ABORT_DUE_TO_CST := TRUE;
    bMoveAbsoluteExecute := FALSE;
    bSpeedOverrideExecute := FALSE;
    seq1State := SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT;
  END_IF

SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT:
  IF axisTorqueControl.bTorqueCommandDone AND NOT axisTorqueControl.bTorqueControlActive THEN
    bTorqueControlExecute := FALSE;
    ABORT_DUE_TO_CST := FALSE;
    seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2;
  END_IF

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2:

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2:
  axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Fast;
  bSpeedOverrideExecute := TRUE;

  IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
    bSpeedOverrideExecute := FALSE;
    seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2;
  END_IF

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2:
(*'Setting Target 2 Parameters')
  axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
  axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
  axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
  axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
  axisMoveAbsolute.nTargetJerk := nTargetJerk_2;

  bMoveAbsoluteExecute := TRUE;

  seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT;

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT:
  IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.hMoveAbsoluteDone THEN
    bMoveAbsoluteExecute := FALSE;

    seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
  ELSIF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteError THEN
    nMoveAbsoluteErrorID := axisMoveAbsolute.nMoveAbsoluteErrorID;

    seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
  END_IF

SEQ1STATE_NEW.SEQ1_FAULT:
  bPowerEnable := FALSE;
  
```

Axis Power

Setting Target Position past the contact point position, will use speed override to slow the speed down after crossing some position threshold.

Reading the axis actual position and trigger speed override upon crossing the position. Users have to set this area to ensure that there is enough stroke to let it travel at slow speed.

Polling the ActTorque that the PLC reads and if its more than 20% then trigger the MC_TorqueControl function block. Inside the MC_TorqueControl, there is a manual enable start option. If it is FALSE, the TargetTorque command will start from 0, else it will start from the point it captured the ActTorque.

Inside the MC_TorqueControl function block, a hold timer was added after the Set Torque have been reached before asserting TorqueCommandDone.

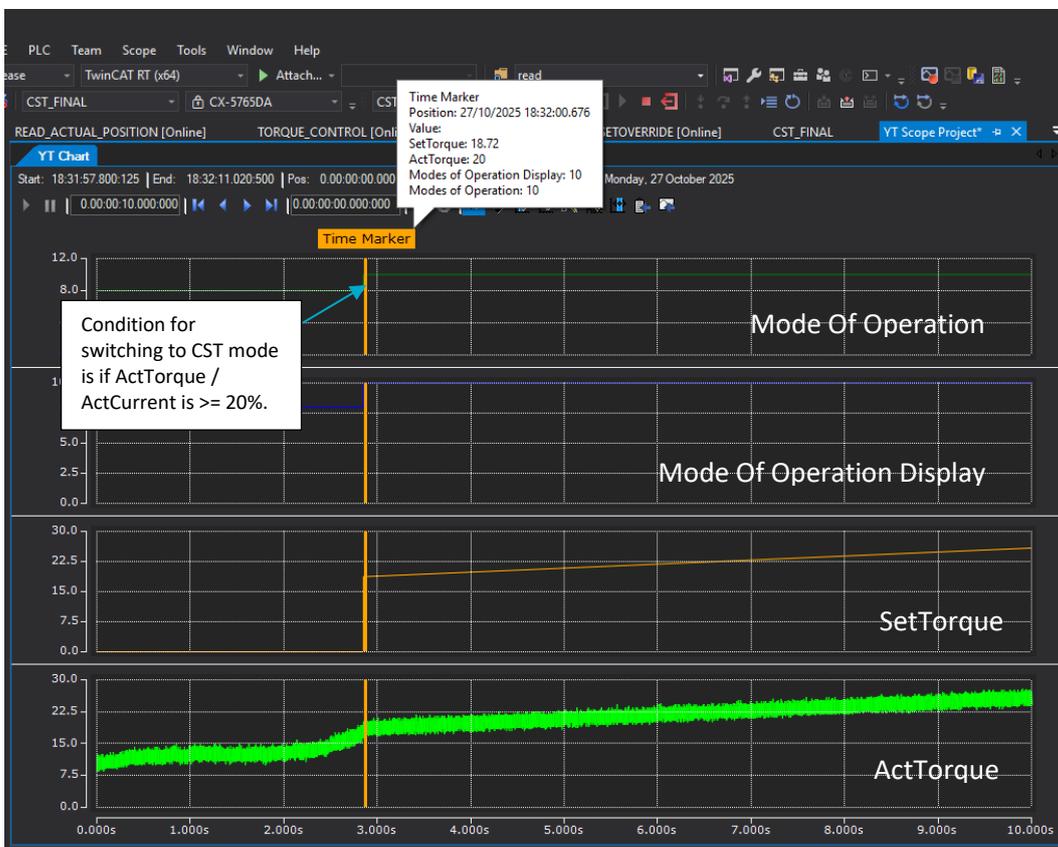
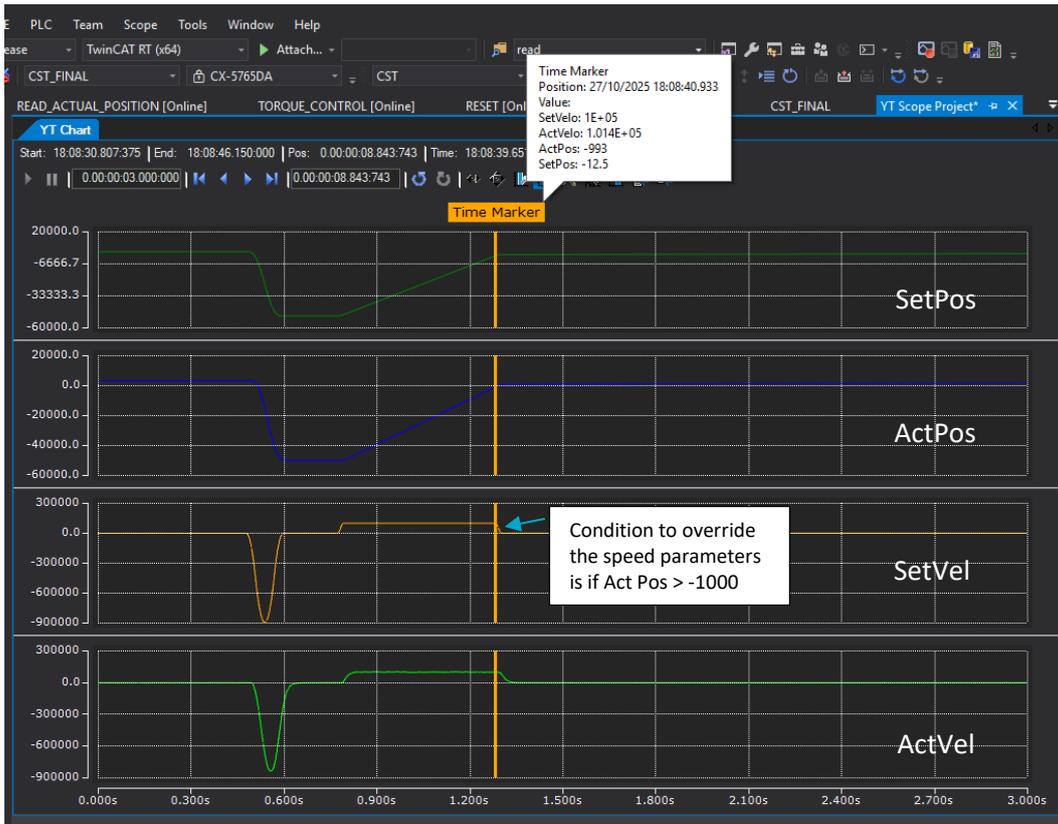
Retracting to Target Position 2.

Resetting the speed override to the original velocity factor to retract the axis at a faster speed.

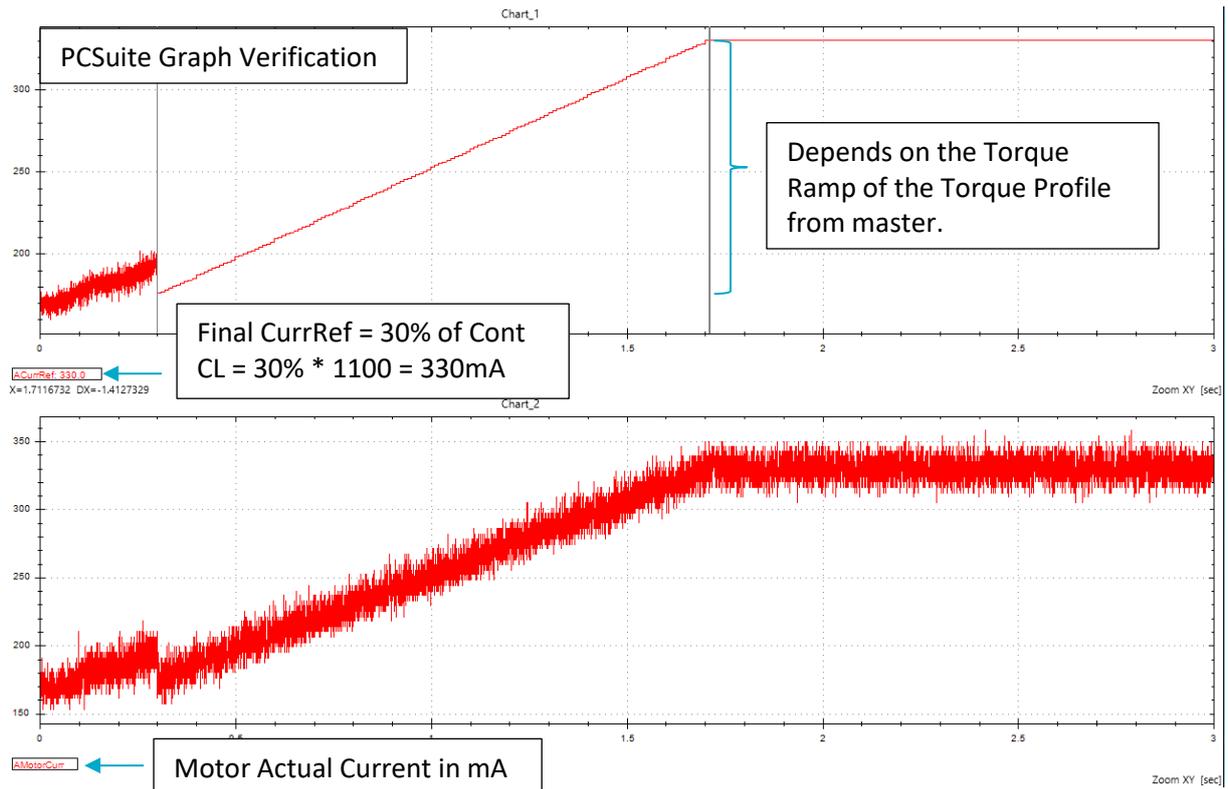
Waiting for done signal.
End of cycle / back to the start of the force control cycle.

END_CASE

3.2.1.3 Verification



Homing Methods Example



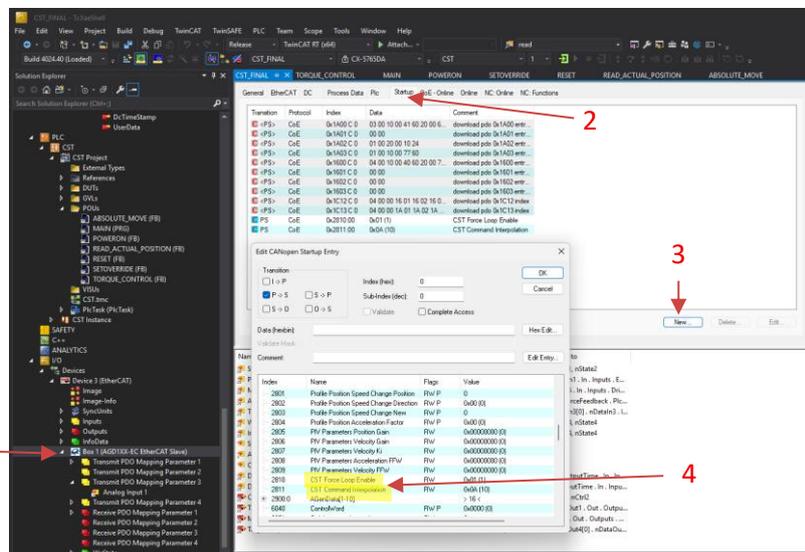
3.2.2 CST Force Loop

3.2.2.1 Modifying PDO List

Similar to CST Current Loop, it is necessary to map the objects that are mentioned above as well as **Analog Input 1 Value (0x2410_h)** into the PDO object list as this object will be used as the force feedback of the force sensor.

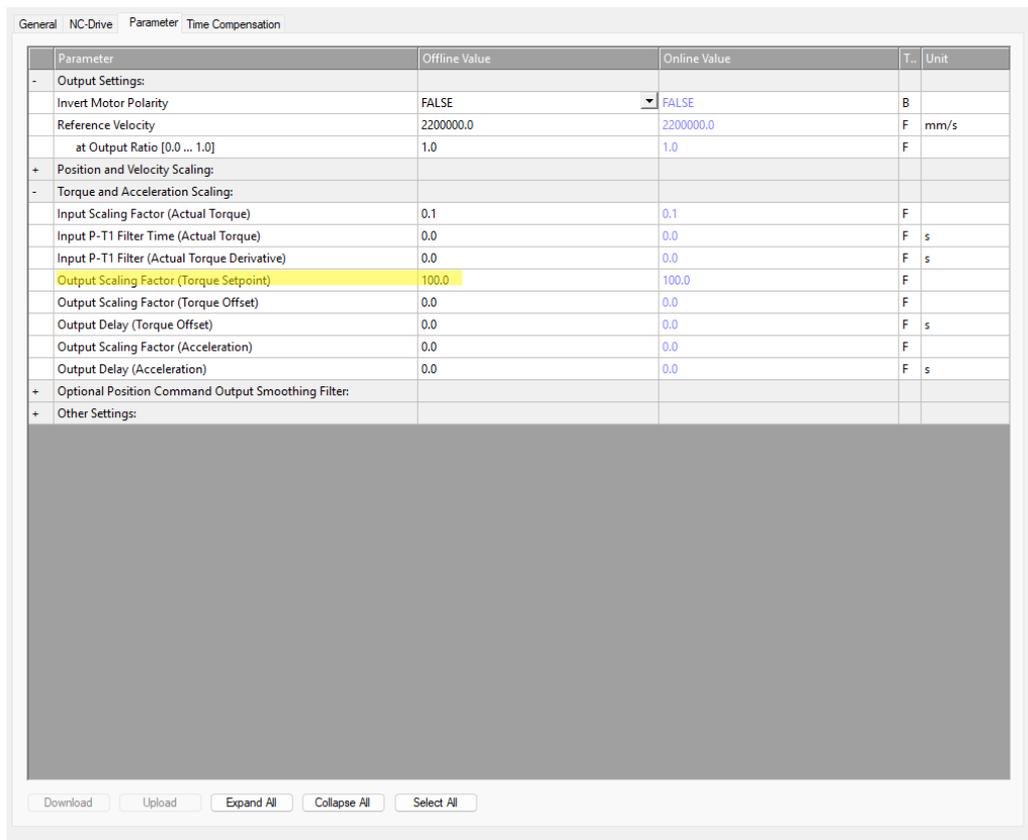
3.2.2.2 Startup Parameters

If the master is able to pre-configure some start up parameters before going into OP mode, they can pre-configure objects **CST Force Loop Enable (0x2810_h)** and **CST Command Interpolation (0x2811_h)** in the startup parameters page shown below.

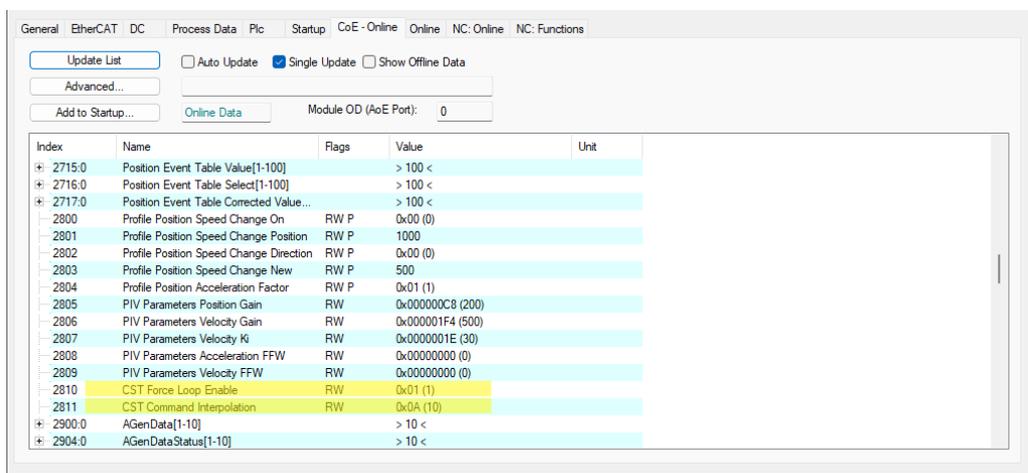


3.2.2.3 Modifying Output Scaling Factor / CST Interpolation Command Factor

By default, the torque output scaling factor is set at 10.0 on the master. This output scaling will in turn use the % command from the MC_TorqueControl function block and scale it to feed into Target Torque Object (0x6071_h). If the user determines that it is too coarse i.e. the force feedback that the user wants to switch at is 50 force units but due to jitter the manual start torque of the MC_TorqueControl function block captured at 48 force units, this will in turn translate to 0.48% and the 0.08% portion will be truncated. Hence, the user can bump up the torque output scaling factor to 100.0 (on the master) and also scale down on the driver side as well by a factor of 10 by inputting “10” in the CST Command Interpolation (0x2811_h) object.



Parameter	Offline Value	Online Value	T...	Unit
- Output Settings:				
Invert Motor Polarity	FALSE	FALSE	B	
Reference Velocity	2200000.0	2200000.0	F	mm/s
at Output Ratio [0.0 ... 1.0]	1.0	1.0	F	
+ Position and Velocity Scaling:				
- Torque and Acceleration Scaling:				
Input Scaling Factor (Actual Torque)	0.1	0.1	F	
Input P-T1 Filter Time (Actual Torque)	0.0	0.0	F	s
Input P-T1 Filter (Actual Torque Derivative)	0.0	0.0	F	s
Output Scaling Factor (Torque Setpoint)	100.0	100.0	F	
Output Scaling Factor (Torque Offset)	0.0	0.0	F	
Output Delay (Torque Offset)	0.0	0.0	F	s
Output Scaling Factor (Acceleration)	0.0	0.0	F	
Output Delay (Acceleration)	0.0	0.0	F	s
+ Optional Position Command Output Smoothing Filter:				
+ Other Settings:				



Index	Name	Flags	Value	Unit
2715:0	Position Event Table Value[1-100]		> 100 <	
2716:0	Position Event Table Select[1-100]		> 100 <	
2717:0	Position Event Table Corrected Value...		> 100 <	
2800	Profile Position Speed Change On	RW P	0x00 (0)	
2801	Profile Position Speed Change Position	RW P	1000	
2802	Profile Position Speed Change Direction	RW P	0x00 (0)	
2803	Profile Position Speed Change New	RW P	500	
2804	Profile Position Acceleration Factor	RW P	0x01 (1)	
2805	PIV Parameters Position Gain	RW	0x000000C8 (200)	
2806	PIV Parameters Velocity Gain	RW	0x000001F4 (500)	
2807	PIV Parameters Velocity Ki	RW	0x0000001E (30)	
2808	PIV Parameters Acceleration FFW	RW	0x00000000 (0)	
2809	PIV Parameters Velocity FFW	RW	0x00000000 (0)	
2810	CST Force Loop Enable	RW	0x01 (1)	
2811	CST Command Interpolation	RW	0x0A (10)	
2900:0	AGenData[1-10]		> 10 <	
2904:0	AGenDataStatus[1-10]		> 10 <	

3.2.2.4 Writing POU

```

(*Power On*)
axisPower(axis_Test := axis_Test, bPowerEnable := bPowerEnable, bPowerStatus => bPowerStatus, bPowerBusy => bPowerBusy, bPowerError => bPowerError, nPowerErrorID => nPowerErrorID);
(*Reset*)
axisReset(axis_Test := axis_Test, bResetExecute := bResetExecute, bResetDone => bResetDone, bResetBusy => bResetBusy, bResetError => bResetError, nResetErrorID => nResetErrorID);
(*Move Absolute*)
axisMoveAbsolute(axis_Test := axis_Test, bMoveAbsoluteExecute := bMoveAbsoluteExecute, ABORT_DUE_TO_CST := ABORT_DUE_TO_CST, nMoveAbsoluteErrorID => nMoveAbsoluteErrorID);
(*Read Actual Position*)
axisReadPosition(axis_Test := axis_Test, bReadActualPosEnable := TRUE, bReadActualPosValid => bReadActualPosValid, bReadActualPosBusy => bReadActualPosBusy, bReadActualPosError => bReadActualPosError,
nReadActualPosErrorID => nReadActualPosErrorID, nActualPosition => nActualPosition);
(*Set Speed Override*)
axisSpeedOverride(axis_Test := axis_Test, bSetOverrideEnable := bSpeedOverrideExecute);
(*Torque Control*)
axisTorqueControl(axis_Test := axis_Test, bTorqueControlExecute := bTorqueControlExecute, nDesiredTorque := lTorque, nDesiredTorqueRamp := lTorqueRamp);
CASE seq1State OF
  SEQ1STATE_NEW_SEQ1_RESET :
    (*Initialization Logic*)
    bResetExecute := TRUE;
    bPowerEnable := FALSE;
    IF bResetDone THEN
      bResetExecute := FALSE;
    ELSEIF bResetError THEN
      bResetExecute := FALSE;
      nResetErrorID := nResetErrorID;
      seq1State := SEQ1STATE_NEW_SEQ1_POWER_ENABLE;
    END_IF
  SEQ1STATE_NEW_SEQ1_POWER_ENABLE :
    (*Axis Power*)
    bPowerEnable := TRUE;
    IF bPowerStatus THEN
      IF bPowerStatus THEN
        seq1State := SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_1;
      ELSEIF bPowerError THEN
        bPowerEnable := FALSE;
        seq1State := SEQ1STATE_NEW_SEQ1_FAULT;
      END_IF
    END_IF
  SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_1 :
    (*Setting Target 1 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_1;
    bMoveAbsoluteExecute := TRUE;
    seq1State := SEQ1STATE_NEW_SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1;
  SEQ1STATE_NEW_SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1 :
    IF axisReadPosition.nActualPosition > -1000 THEN //--100000 THEN
      axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;
      bSpeedOverrideExecute := TRUE;
    END_IF
    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
      seq1State := SEQ1STATE_NEW_SEQ1_CST_MODE;
    END_IF
  SEQ1STATE_NEW_SEQ1_CST_MODE :
    IF GVL.ForceFeedback >= 50 THEN //axis_Test.NoToPic.ActTorque >= 20.0 THEN
      bTorqueControlExecute := TRUE;
      ABORT_DUE_TO_CST := TRUE;
      ABORT_DUE_TO_CST := TRUE;
      bMoveAbsoluteExecute := FALSE;
      bSpeedOverrideExecute := FALSE;
      seq1State := SEQ1STATE_NEW_SEQ1_CST_FORCE_HOLDING_WAIT;//SEQ1STATE_NEW_SEQ1_CST_FORCE_HOLDING_WAIT;
    END_IF
  SEQ1STATE_NEW_SEQ1_CST_FORCE_HOLDING_WAIT :
    IF axisTorqueControl.bTorqueCommandDone AND NOT axisTorqueControl.bTorqueControlActive THEN
      bTorqueControlExecute := FALSE;
      ABORT_DUE_TO_CST := FALSE;
      seq1State := SEQ1STATE_NEW_SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2;
    END_IF
  SEQ1STATE_NEW_SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2 :
    axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Fast;
    bSpeedOverrideExecute := TRUE;
    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
      bSpeedOverrideExecute := FALSE;
      seq1State := SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_2;
    END_IF
  SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_2 :
    (*Setting Target 2 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_2;
    bMoveAbsoluteExecute := TRUE;
    seq1State := SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT;
  SEQ1STATE_NEW_SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT :
    IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteDone THEN

```

Instance of function blocks used

Reset / Initialization Sequence

Axis Power

Setting Target Position past the contact point position, will use speed override to slow the speed down after crossing some position threshold.

Reading the axis actual position and trigger speed override upon crossing the position. Users have to set this area to ensure that there is enough stroke to let it travel at slow speed.

Polling the Analog Input Feedback and if its more than 50 force units then trigger the MC_TorqueControl function block. Inside the MC_TorqueControl, there is a manual enable start option. If it is FALSE, the TargetTorque command will start from 0, else it will start from the point it captured the in terms of %.

Inside the MC_TorqueControl function block, a hold timer was added before the Set Torque has been reached before asserting TorqueCommandDone.

Resetting the speed override to the original velocity factor to retract the axis at a faster speed.

Retracting to Target Position 2.

Homing Methods Example

```

IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteDone THEN
  bMoveAbsoluteExecute := FALSE;

  seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
  ELSEIF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteError THEN
    nMoveAbsoluteErrorID := axisMoveAbsolute.nMoveAbsoluteErrorID;

    seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
  END_IF

  SEQ1STATE_NEW.SEQ1_FAULT:
    bPowerEnable := FALSE;

END_CASE

```

Waiting for done signal.
End of cycle / back to the start of the force control cycle.

```

fbExecute(CLK:= bTorqueControlExecute):
CASE nState OF
0:
  IF NOT bTorqueControlExecute THEN
    bTorqueCommandDone := FALSE;
    bInTorque := FALSE;
    bTorqueControlBusy := FALSE;
    bTorqueControlActive := FALSE;
    bTorqueControlCommandAborted := FALSE;
    bTorqueControlError := FALSE;
    nTorqueControlErrorID := 0;
    DelayTimer(IN:=FALSE);
  END_IF

  IF fbExecute.Q THEN
    nState := nState + 1;
  END_IF

1:
  Options.EnableManualTorqueStartValue := TRUE;
  Options.ManualTorqueStartValue := (GVL.ForceFeedback * 0.01)//axis_Test.NcToPlic.ActTorque;

  fbTorqueControl(
    Axis           := axis_Test,
    Execute        := bTorqueControlExecute,
    ContinuousUpdate := bTorqueControlContinuousUpdate,
    Relative       := bTorqueControlRelative,
    Torque         := nDesiredTorque,
    TorqueRamp     := nDesiredTorqueRamp,
    VelocityLimitHigh := nVelocityLimitHigh,
    VelocityLimitLow  := nVelocityLimitLow,
    Options       := Options);

  IF fbTorqueControl.Active THEN
    bTorqueControlActive := fbTorqueControl.Active;
    nState := nState + 1;
  ELSEIF fbTorqueControl.Error THEN
    bTorqueControlError := fbTorqueControl.Error;

```

Inside the MC_TorqueControl Function Block

Scaled the Force Feedback depending on "CST Command Interpolation" or the Output Torque Scaling on the master.

On the topic of scaling for Force Feedback, CST command interpolation and also Output Torque Scaling (not all masters support this option).

In the case of default scaling of CST Command Interpolation and Output Torque Scaling:

If CST Command Interpolation is default (value of 1) and Output Target Torque Scaling is default (value of 10)

1% of the commanded "Torque" would give a Target Torque (0x6071) value of 10 due to the default output target torque scaling (default value of 10)

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

Equation of Force Ref = TargetTorque - TargetTorquePrev * 0.001 (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

So a value of 1% equates to Target Torque 10 = Force Ref of 100 Force units Hence, it is needed to scale the force feedback by a factor of 0.01 inside the MC_TorqueControl -> ManualTorqueStartValue

Switching to CST Condition = 50g = 50 * 0.01 = 0.5% on the PLC giving a Target Torque Value of 5. However, due to jitter it could have capture the start value at 0.48% and due to truncation, the Target Torque Value is input as 4 instead of 4.8 and under these circumstances it is advised to do some scaling to acquire the truncated values as well.

In the case of scaling of CST Command Interpolation and Output Torque Scaling by a factor of 10:

If CST Command Interpolation is scaled by 10 (value of 10)
and Output Target Torque Scaling is scaled by 10 (value of 100)

1% of the commanded "Torque" would give a Target Torque (0x6071)
value of 100 after the output target torque scaling.

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

Equation of Force Ref = $\text{TargetTorque} - \text{TargetTorquePrev} * 0.001$ (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

So a value of 1% equates to Target Torque 100 = Force Ref of 100 Force units (assuming that TargetTorquePrev is 0)
Hence, the scaling set for the force feedback is a factor of 0.01 as well inside the MC_TorqueControl -> ManualTorqueStartValue

Switching to CST Condition = 50g = $50 * 0.01 = 0.5\%$ on the PLC giving a Target Torque Value of 50 (due to scaling).
However, due to jitter it could have capture the start value at 0.48% and it will give a value of Target Torque Value of 48 (due to output target torque scaling)

In the case of scaling of CST Command Interpolation (by 10) and default scaling for Output Torque Scaling:

If CST Command Interpolation is scaled by 10 (value of 10)
and Output Target Torque Scaling is default (value of 10)

1% of the commanded "Torque" would give a Target Torque (0x6071)
value of 10 after the output target torque scaling.

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

Equation of Force Ref = $\text{TargetTorque} - \text{TargetTorquePrev} * 0.001$ (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

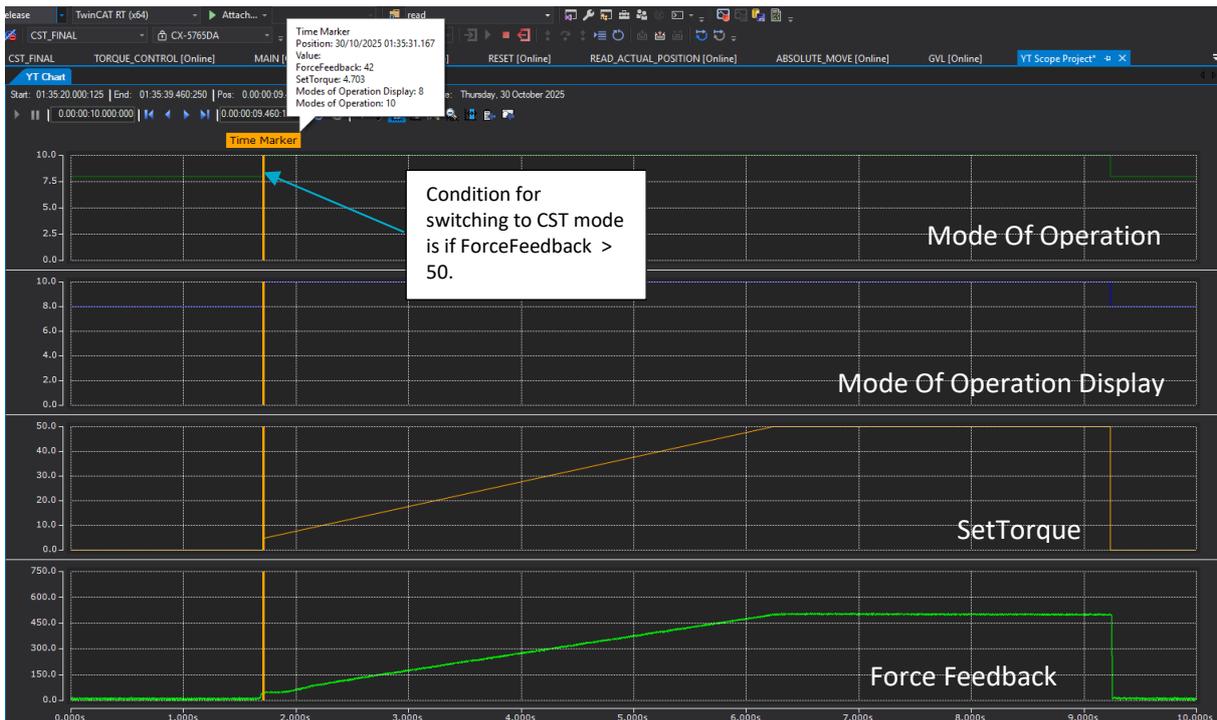
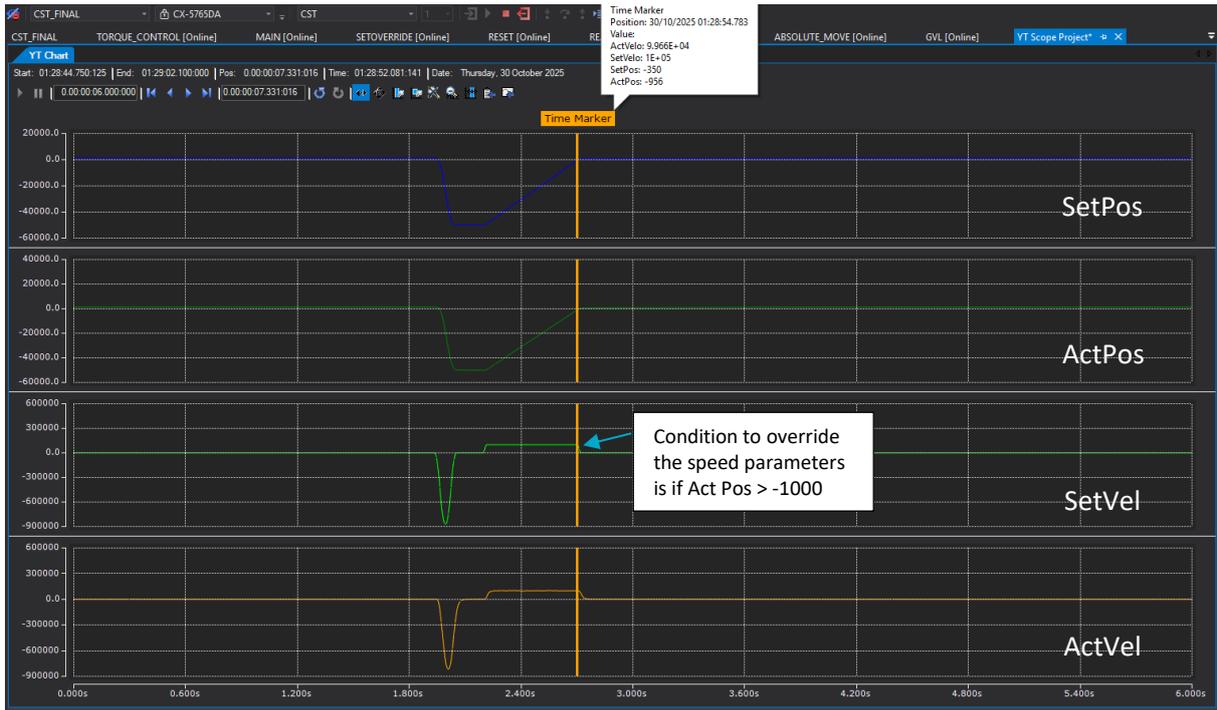
So a value of 1% equates to Target Torque 10 = Force Ref of 10 Force units (assuming that TargetTorquePrev is 0)
Hence, the scaling set for the force feedback is a factor of 0.1 inside the MC_TorqueControl -> ManualTorqueStartValue

Switching to CST Condition = 50g = $50 * 0.1 = 5\%$ on the PLC giving a Target Torque Value of 50 (due to scaling).
However, due to jitter it could have capture the start value at 4.8% and it will give a value of Target Torque Value of 48 (due to output target torque scaling)

In this case, it is necessary to bump the % Target Torque command by a factor of 10 as well.

Homing Methods Example

3.2.2.5 Verification



Homing Methods Example

