

INTRODUCTION

FORMAT

The format of the Advanced Programming Workshop class is different from the typical Yaskawa training class. In this class you'll write an advanced rotary knife application, following this Application Resource Manual, and working at your own pace. The instructor is available to answer your questions about the material and to check to be sure you are working on track.

DEMO EQUIPMENT

The vast majority of the program can be tested on the desktop demo. At key points in the development of your code it will be necessary to test it out on the "real machine". The real machine is the Rotary Knife / Flying Shear demo. You are welcome to try your code on the real machine at any time. If you need to use the machine but it is in use by another student, please make this known to the instructor, but continue working ahead in this Application Resource Manual.

MATH AND EQUATIONS

This class does require that the student understand quite a few mathematical calculations and motion profile analysis. However, for the most part, the equations and profile analysis is explained so that the student can concentrate on incorporating these calculations in the program, rather than spending time developing them. So save yourself some time by reading ahead a few pages to find these calculations before solving them on your own.

TRAINING ACTIONS

Throughout this document are many headings titled "Training Action". These may consist of questions to answer, procedures to follow, or programming tasks to solve. This class is designed for you to "learn by doing" accompanied by simple, yet accurate descriptions to keep you on track. Please don't skip over the Training Actions. They serve as a checkpoint for the instructor to monitor your progress and to be sure you understand the material before getting too far ahead.

TRAINING ACTION

- Read the application description.
- Ask the instructor to clarify anything that is questionable or unclear.
- Think about how you would use the controller to solve the application

TABLE OF CONTENTS

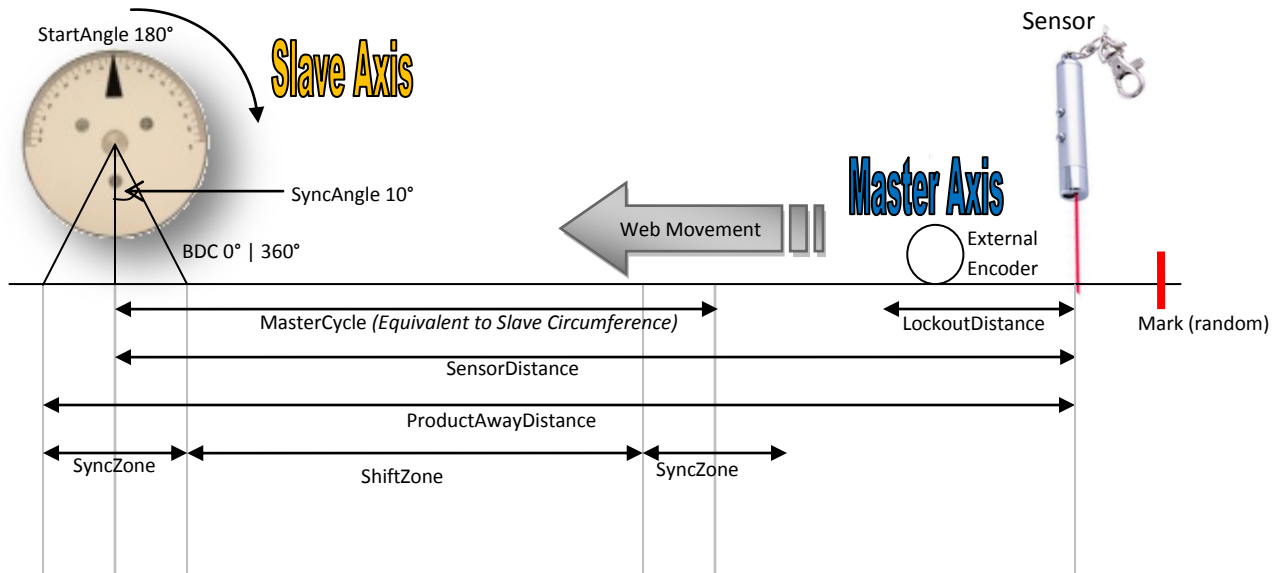
CONTENTS

Introduction.....	1
Application Description: Rotary Knife.....	4
Expected Motion Profile	7
Solution approaches.....	8
Three Incorrect Approaches	8
The Correct Approach: Electronic Cam.....	10
Solution Approach	11
Starter Project	12
Setup.....	12
Homing the demo	14
Cam Engage/Disengage	15
Running Cam Profile	16
Shift Introduction.....	18
Cam Shift Concept overview.....	18
Product Buffer	21
Product Buffer Definitions	23
Product Buffer Quickstart.....	24
ProductBuffer Execution.....	27
Product Away.....	28
Top-Level Sequence.....	29
First Shift and Engage	31
Camming Block Diagram.....	31
First Shift Simplified Example	33
Training Action:.....	33

First Shift Equation	36
Y_CamIn Execution Zone	37
PITFALL: Sensor Location	38
Programming: First Shift and Engage	39
SFC Actions	40
Running Shift	42
Shift Zone.....	42
Shift Execution Timing: “Within Range”	43
Running Shift Calculation.....	45
Program the RunningShift.	46
Last Shift	48
Dead Man condition	48
Exe_LastShift.....	52
Phase 1 Conclusion	53
Phase 2: Internal Cam Table Generation	55
Y_CamStructSelect.....	55
CamGenerator	56
Cam Tool (software) and CamGenerator (function block)	57
Elements of the CamData structure	59
Programming: Internal Cam Table Generation.....	61
Phase 3 – Cam Blend	62
BlendData Structure	64
CamBlend Illustration	66
CamBlend: General Requirements for Cam Tables	68
Programming: CamBlend.....	69

APPLICATION DESCRIPTION: ROTARY KNIFE

MACHINE EXAMPLES: Any machine that performs a process to a continuous “web” of moving material by means of a rotary actuator. This workshop will assume the process to be cutting, but the process could also be printing, forming, sealing, or others.

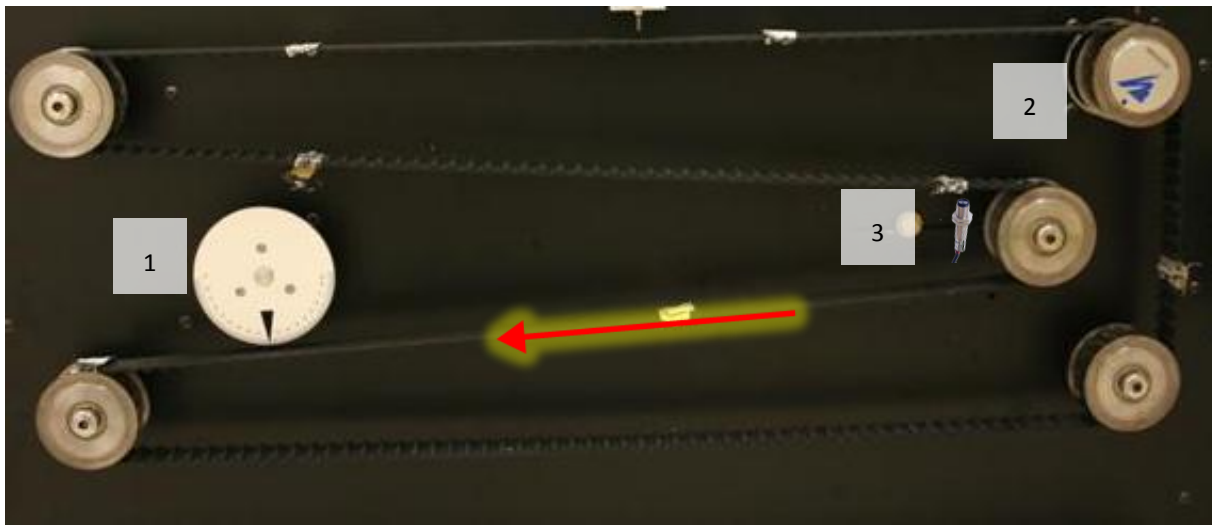


MACHINE DESCRIPTION AND OPERATION

- Material is moving continuously, in the diagram this is from right to left
- The material is printed with a mark and must be cut on the leading edge of this registration mark with a tolerance of $[\pm 0.001 \text{ Inches}]$.
 - Even if there is material stretching or slippage, it will be cut at the correct location relative to the other graphical printing on the material.
- The knife will rotate in only one direction (CW in this diagram) to cut the material while moving.
- The rotary knife is driven by a rotating servo axis, as shown in the diagram, which is controlled by the MP2000iec.
- This knife is to penetrate a single point on the material as it moves, moving synchronously with the material throughout the entire cutting cycle, even in the unlikely case that the material were to speed up or slow down.
- The cutting is “random” in the sense that the registration marks may not be exactly the same distance apart. They are random to within 1.0 [inches]
- The final product, and therefore the product length must be able to change “on the fly”

MECHANICAL SPECIFICATIONS

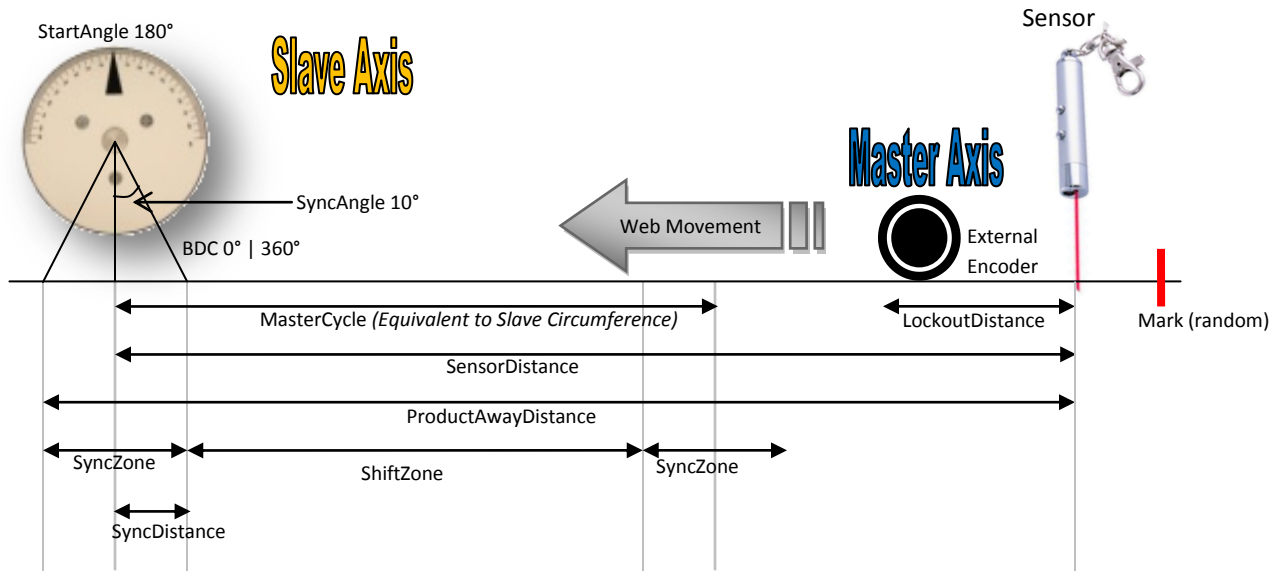
1. The servomotor that directly drives the rotary knife is a Sigma-2 motor (driven by a Sigma-5 amplifier) and has a resolution of 8192 [post-quadrature counts / rev]
 - o Knife diameter is 4.000 [inches], giving a circumference of $4 * \pi = 12.566$ [inches].
2. The controller measures the position of the material using an external encoder. On this demo that encoder is the master axis.
 - o The external encoder on the web of material has a resolution of 8192 [post-quadrature counts / rev]
 - o One revolution of the external encoder represents 8 [inches] of material movement
3. The sensor is located a relatively long way away from the knife cutting point. The distance from the registration sensor to the rotary knife bottom dead center is roughly 28.5 [in].



PERFORMANCE REQUIREMENTS

- Cut any length between 5[in] and 10[ft].
- The conveyor travels up to 400[ft/min] = 80[in/sec]
- The material feeds at a speed that may be adjusted by the operator. The material may suddenly slow or stop altogether due to an unexpected power outage or emergency shutdown. Therefore the knife must move at the exact speed of the material during the cutting cycle, or else the material will be wasted or the mechanism could jam.
- The knife must stop smoothly, pointing out of the material while waiting for product.

IMPORTANT MEASUREMENTS

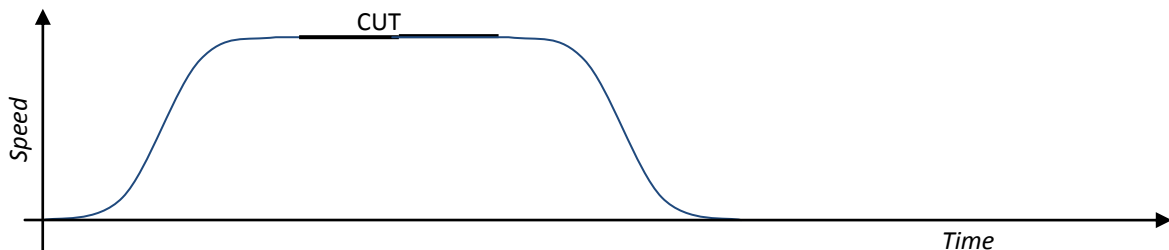


- **StartAngle:** Blade angle of 180 degrees, pointing up. The knife waits here until the sensor is latched. The knife will not stop here if another product has already been sensed.
- **BottomDeadCenter (BDC):** Blade position 360 degrees (= 0 degrees) pointing down. Full cut penetration at this point.
- **SyncAngle:** Blade angle in degrees, \pm BDC during which the knife must remain synchronized with the material. Typical ± 10 degrees. This SyncAngle can be adjusted based on material thickness.
- **SyncDistance:** Linear distance of the angle projected by the knife from BDC to first contact point of blade with material. Will be adjusted based on SyncAngle.
- **MasterCycle:** Linear distance the material moves per knife rotation, if the knife were to remain synchronized all the time. This distance is equal to the circumference of the knife.
- **ShiftZone, SyncZone:** The sync distance defines a zone within the master cycle in which the master can be shifted or must remain synchronized.
- **Sensor Distance:** Linear distance from registration sensor to knife BDC
- **ProductAwayDistance:** Linear distance from sensor to where the knife clears the product. At this distance the knife is free to adjust speed for the next cut mark.
- **LockoutDistance:** Filter to reject false latches after the current latched position.

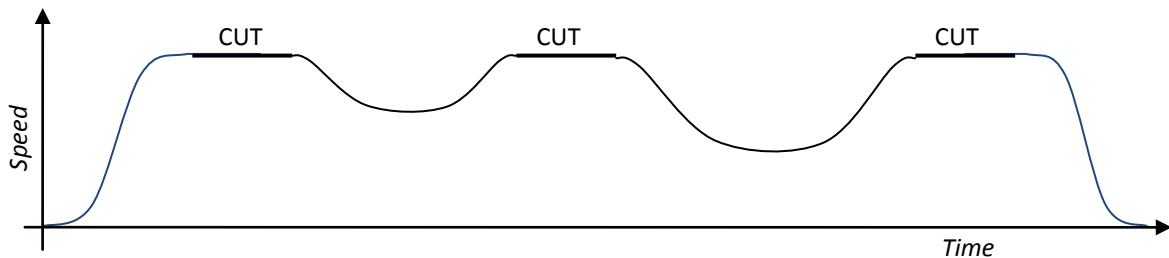
EXPECTED MOTION PROFILE

Before and after the cut, the rotary knife is expected to keep moving, slowing down or speeding up between cuts as needed. It will stop only if the sensor has detected no more marks. The most critical part of the knife motion occurs during the cut. The knife must follow the master during the cut. If the master is moving at a constant speed, then the knife will follow at this same constant speed.

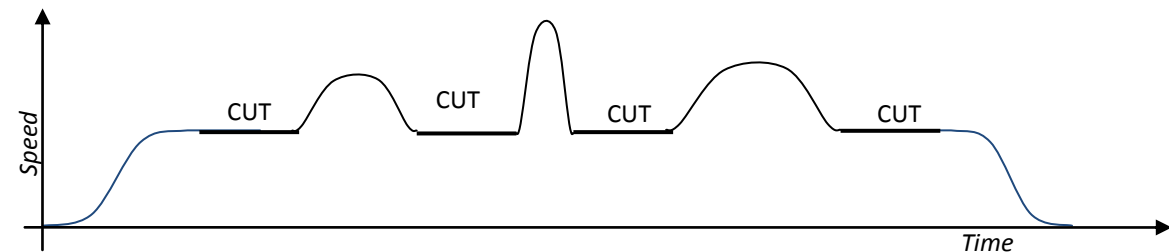
- **Case 1: Very distantly spaced. The knife stops at home position between cuts, waiting for another cut mark to be sensed.**



- **Case 2: Distantly spaced. The knife keeps moving, slowing down between cuts.**



- **Case 3: Closely spaced. The knife keeps moving, speeding up between cuts.**



For a truly random mark, the knife must work for any combination of the above 3 cases. For example, the very first cut will by definition follow the acceleration of Case1. After all, the knife was stopped before power was turned on! Then as marks are sensed, it may alternate randomly between distantly and closely spaced cuts as the marks are sensed. Eventually no more marks are sensed, and the knife will come to rest again as in the end of Case1, waiting for the possibility of another mark.

SOLUTION APPROACHES

THREE INCORRECT APPROACHES

SET THE SPEED ONCE

At first glance, the solution to this application seems simple. Read the speed of the conveyor and move the servomotor forward at that speed for the appropriate amount of time.

This method, however, does not consider the fact that the motor must perfectly align with an exact position of the material on the conveyor in order that the cut be made accurately. So some type of algorithm would have to monitor and match the position of the conveyor with the servomotor. This can be cumbersome, and the effectiveness is limited.

The possibility also exists that the conveyor could change speed while the motor is moving. The customer has stated that the speed of the saw must exactly match the speed of the conveyor for the whole time that cutting is taking place. Therefore, setting the speed once is not sufficient.

UPDATE THE SPEED CONTINUOUSLY

So the simple fix then seems to be to continuously monitor the conveyor speed and update the motor speed to match it during the critical part of the move. This could be done with a simple program loop that reads the speed of the conveyor and sets the same speed to the motor.

This approach does work to a limited degree, but it will soon become apparent that the motor is not following the speed exactly. This is because there is too much delay. The encoder on the conveyor is sending pulses, which must be converted to speed by dividing pulses by time. Time is necessary to calculate speed, so a time delay is unavoidable.

Compare this speed-matching problem to the familiar experience of driving an automobile. To match the speed of another car, a driver naturally tries to maintain a constant distance between cars. It would be far less effective to match speedometer readings, even if a wireless display of the other car's speedometer was available. The same is true in servo applications. If you match position, you will also match speed. But matching speed does not ensure you will match position.

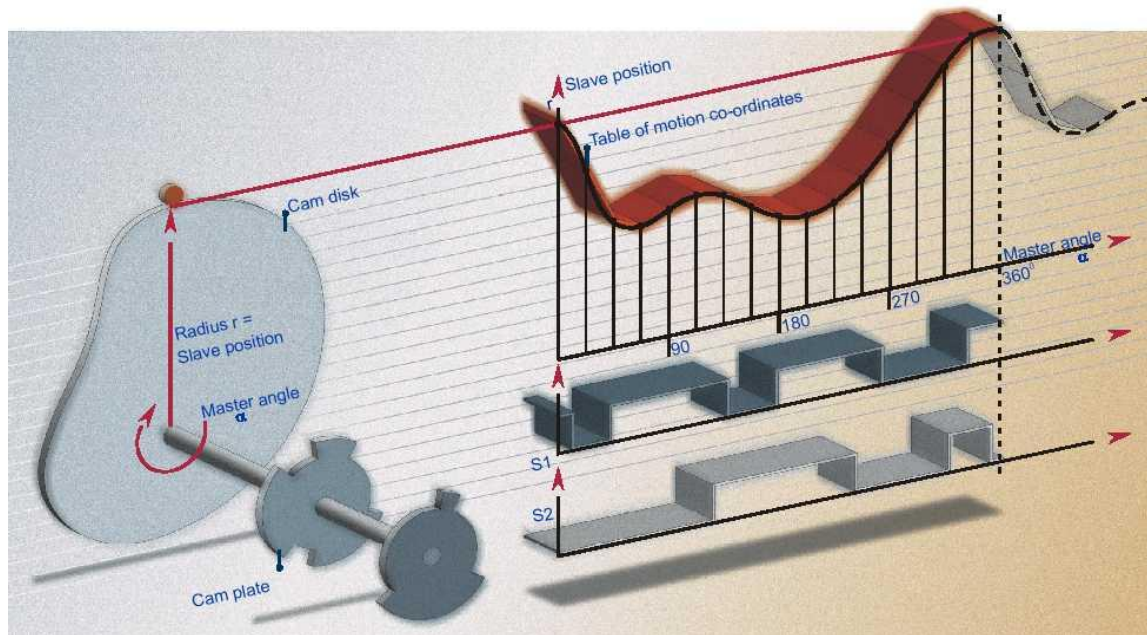
USE ELECTRONIC GEARING

Electronic gearing is designed for just that purpose - to match the motor's position to that of an external axis. This external axis is referred to as the **master axis**. In the case of this application, the encoder on the conveyor is the master axis. The **slave axis** is the motor being controlled. If geared properly, the slave will follow the master exactly.

The approach would then be to smoothly transition into electronic gearing for the time given, and quickly position back for the next cut. This would ensure that the position is followed during the cutting part of the move. The problem is that it is very difficult to synchronize exactly. Additionally, there is always a shock if electronic gearing is engaged while the motor is in motion. Steps can be taken to counteract these problems and make gearing work. But they end up making program development time consuming and cumbersome.

THE CORRECT APPROACH: ELECTRONIC CAM

Gearing and electronic CAM are similar. With gearing, the position of the slave is controlled to be directly proportional to the position of the master, in proportion to a constant value called the gear ratio. With electronic cam, the position of the slave is individually controlled according to specific positions of the master rather than at a constant proportion. These positions are defined in a **CAM TABLE**. As the master position increases, the slave position could increase, decrease, or stay the same. If the master is running at a constant speed, the slave positions can be set to correspond to a desired velocity profile. But if the master were ever to change speed, stop, or reverse, electronic CAM ensures that the slave motor is always in exactly the right position relative to the master, regardless of any velocity profile expected from the slave.



With electronic cam, the issue of conveyor speed disappears. It is totally out of the equation. The only reason to consider the conveyor speed is to calculate an **EXPECTED VELOCITY PROFILE** for the slave motor. The expected velocity profile is the velocity profile you *expect* the slave to have when the master axis moves at a constant speed. But this profile will not be followed if the master does not move as expected.

SOLUTION APPROACH

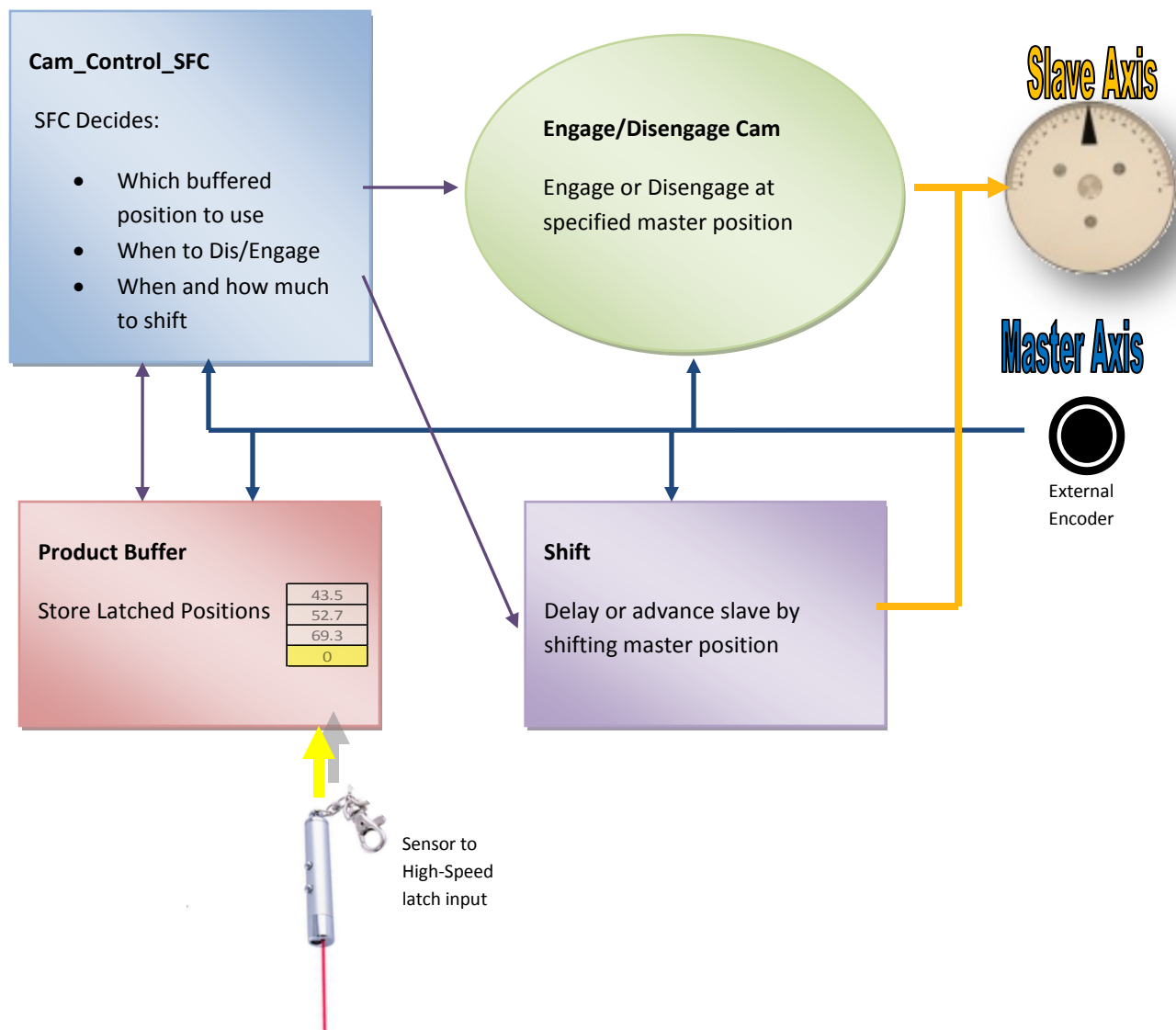
Use a 1:1 straight-line cam profile. Shift the cam when the master cycle is in the shift zone. Adjust the shift amount based on the distance between latched positions

Phase1: Automatically control cam engage and shift using SFC logic

Phase2: Calculate the cam table file internally using CamGenerator

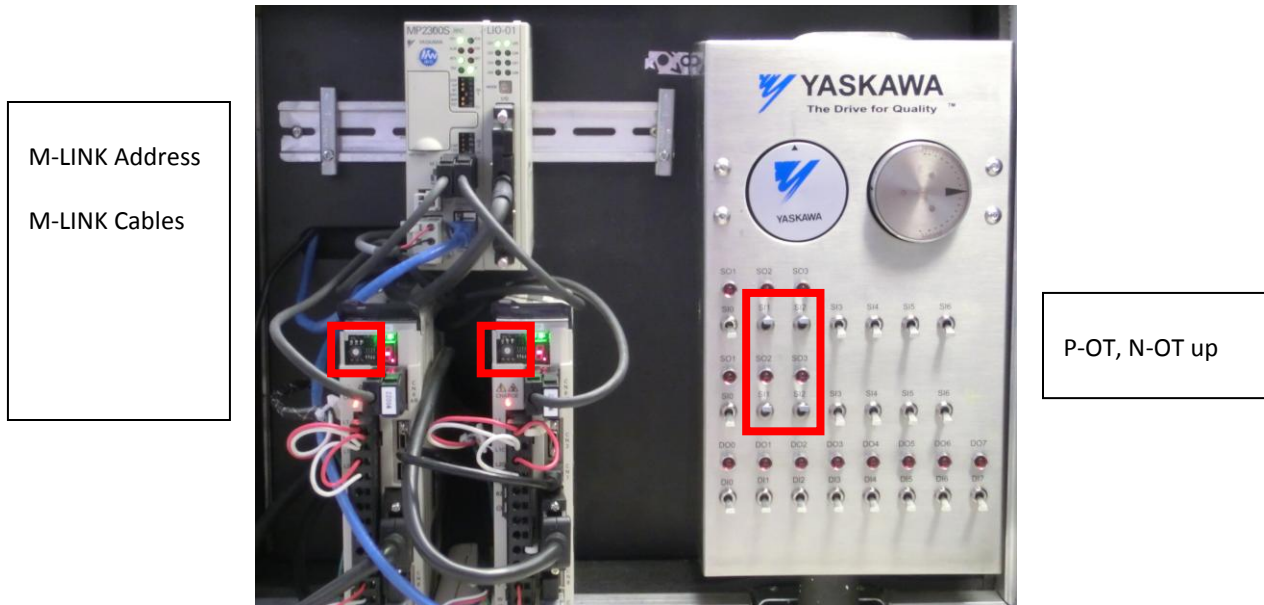
Phase3: Use CamBlend for smooth engage and disengage

Phase4: Adjust and measure performance



STARTER PROJECT

SETUP



Training Action: Prepare Desktop Demo

- All inputs down except P-OT (SI1) and N-OT (SI2) up to simulate normally closed over-travel sensors.
- MECHATROLINK cables securely connected.
- MECHATROLINK address rotary switch set. Left = 1, Right = 2

TRAINING ACTION: STARTER PROGRAM OVERVIEW

It is important to send the archive (and not only the .mwt project) so that all data, including the configuration and the cam table are sent. Servopack Parameters and Absolute Encoder must also receive configuration.

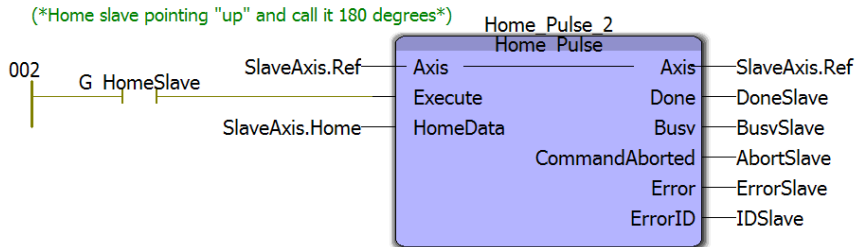
- In Web Server (Login as Admin/MP2300S – case sensitive)
 - Project Archive, Browse for “Archive_APW1... .zip”, Send to Controller
 - Drive Parameters – Write All User Pns
 - Reboot
 - Alarm A.CC0? (1)Machine operations (2)Drive Pn tab (3)Select Axis (4)Multi-turn Reset (5)Reboot
- In MotionWorks IEC – Pro
 - Open/Unzip the APW1Starter .zwt project file to the project folder and RENAME
 - Install the latest “Toolbox Installer” from Yaskawa.com
 - Benefit: Enables right-click help on the toolbox
 - ~~Add the most recent version of CamToolbox and PLCopen Toolbox libraries, delete old~~
 - Open Hardware Configuration and confirm units. Update IP address.
 - Make, Download, and Cold-Start, then turn Debug mode ON
- Open the “Controls” POU to see how to use the different switches and to become familiar with the different POU's of the project.
 - Enable both axes (Switch SI3)
 - Jog the master (Switch DI2)
 - Adjust master speed (View – Watch Window, “Jog” tab)
 - 80.0 is the application maximum. 5.0 is good for testing.
- Open the Init POU and verify the mechanical constants, comparing to the application description.

QUESTIONS:

- The desktop demo has the Left and Right motor. Which motor is the cam master and which is the slave?
- Open the Hardware Configuration. How many inches does one revolution of the master axis represent on this desktop demo?
- How is this different than the physical inches per revolution of the master on the machine (Refer to Mechanical Specifications, p.4)?
- How many degrees does one revolution of the slave represent?

HOMING THE DEMO

While homing is not the focus of this application, it is beneficial to home the axes to start at the correct position. The starter project takes advantage of Yaskawa's PLCopen Toolbox user library and the Home_Pulse function block to home to the encoder's "C-pulse" (sometimes called "marker pulse" or "index pulse"). The relationship to the c-pulse and the arrow on the disc varies from demo to demo, so a home offset must be set to calibrate the home position on the particular demo in front of you.



TRAINING ACTION: HOME THE DESKTOP DEMO

1. Open the Home POU (debug mode)
2. Open the Watch window, Home tab, set SlaveAxis.Home.Offset = 0.0
3. Execute the homing routine (DI1)
4. Disable both servos (SI 3) and move the motors by hand to the calibration **zero position**

Calibration ZERO position



5. Set SlaveAxis.Home.Offset equal to the SlaveAxis.Prm.ActualPosition viewed in the watch window.
6. Enable both servos (SI 3) and home the axes again (DI1) to confirm. The slave axis will stop at the 180 deg position, pointing "away"

Position after successful homing



7. The demo uses absolute encoders, and so it never has to be homed again. The "real machine" uses incremental encoders, and so must be homed after every power cycle.
8. The SlaveAxis global variable is retained during power cycle. SlaveAxis.Home.Offset is initialized to a value of -89.0 (specific to Rotary Knife Demo) only at cold start in the "Cold" system task. Therefore do not do cold start again on the desktop demo or else you will have to calibrate the home position again.

ADDITIONAL INFORMATION ON PLCOPEN TOOLBOX “HOME_PULSE”

The Home_Pulse function block uses the “HomeData” input. This input uses the “HomeStruct” datatype included in the PLCopen Toolbox library. This structure itself is an element to a greater structure called “AxisStruct”. You can see in the global variables that we have a variable SlaveAxis and MasterAxis each as an AxisStruct, which means it includes the HomeStruct. Add from the global variables list both SlaveAxis and MasterAxis to visually see the elements of these data structures. The HomeStruct element is already in the watch window and appears as MasterAxis.Home and SlaveAxis.Home. Expand it to see the elements of the structure. The Init POU “home” worksheet loads default values into the structure.

CAM ENGAGE/DISENGAGE

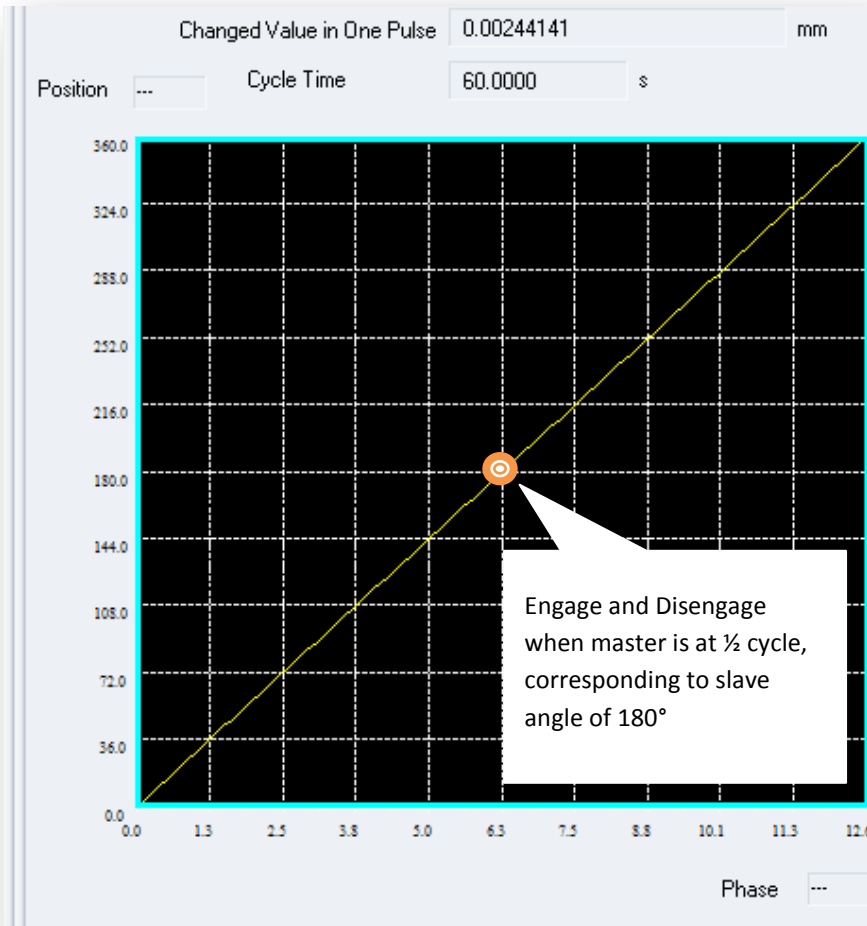
It is critical to understand how the controller engages and disengages the cam. Please do the following on the demo for hands-on experience.

TRAINING ACTION

- Open the CamInOut POU
- Jog the master slowly with DI2 (or disable the master and move by hand)
- Engage and disengage the cam with DI3 which executes the Y_CamIn block
- Open the watch window to see the master and slave position as you engage and disengage
 - Roughly, what is the change in master position over one cam cycle?
 - At what slave position does the slave engage and disengage?

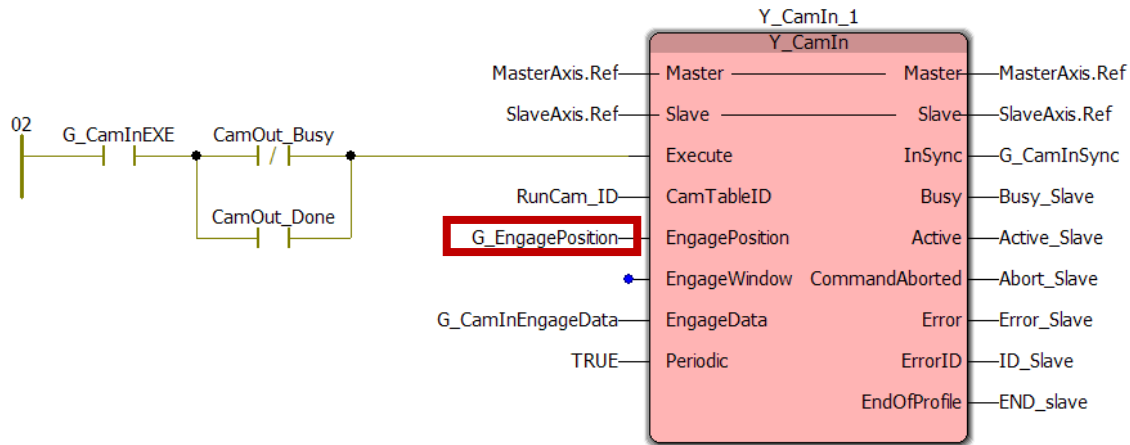
RUNNING CAM PROFILE

When we engage the cam, the controller moves the motor according to a simple 1:1 linear cam. This is a cam profile that behaves in many ways like electronic gearing. The cam table exists on the controller as a CSV file of master and slave positions (created by Yaskawa's Cam Tool software). The cam profile looks like this:

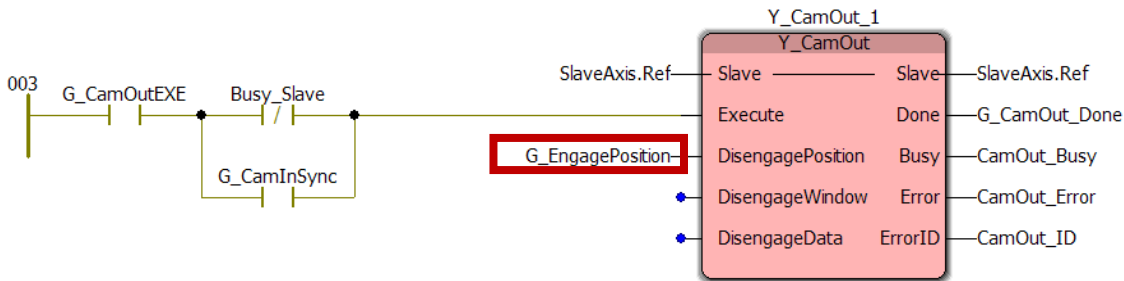


As the master goes through its cycle (zero to 12.566 [in]), the slave does one revolution (zero to 360). Once engaged, you can see the slave is essentially “gearing” to the master at a ratio of 1:1. We call this simple cam profile the “Running Cam”.

In order for the slave to stop and start at the “up” angle of 180 [deg], the EngagePosition is set to ½ the master cycle. See the Init POU.



The same is done for the Y_CamOut instruction so that the slave stops in the “up” position.



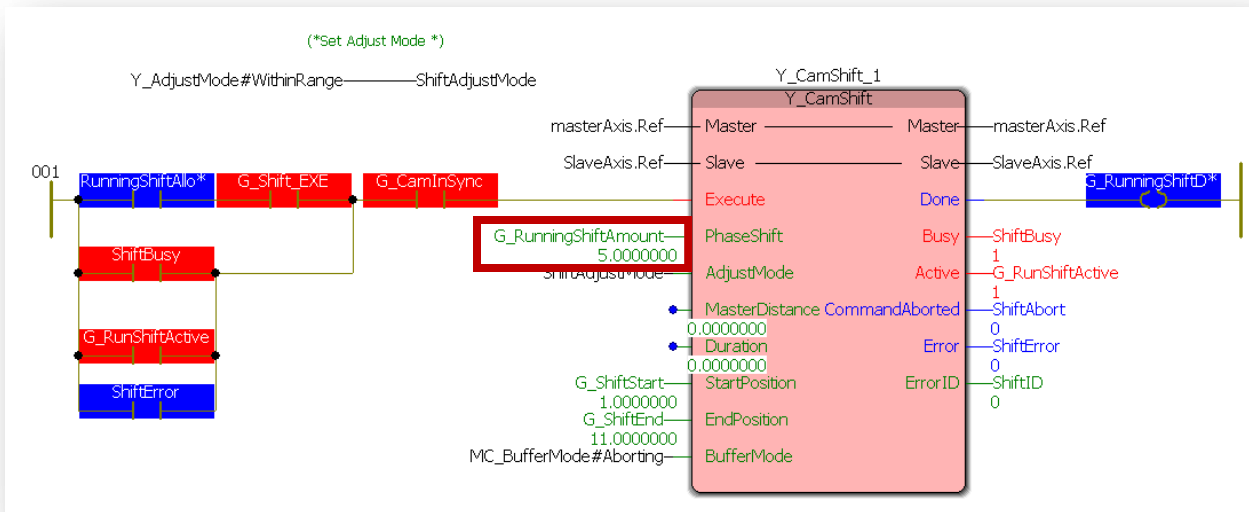
You’ll notice that the slave quite abruptly starts and stops as it engages and disengages. This will be remedied in Phase 3 with CamBlend and two more profiles called “RampIn” and “RampOut”.

SHIFT INTRODUCTION

It is also critical to understand the concept of shifting the cam master.

TRAINING ACTION: EXECUTE "RUNNING CAM SHIFT"

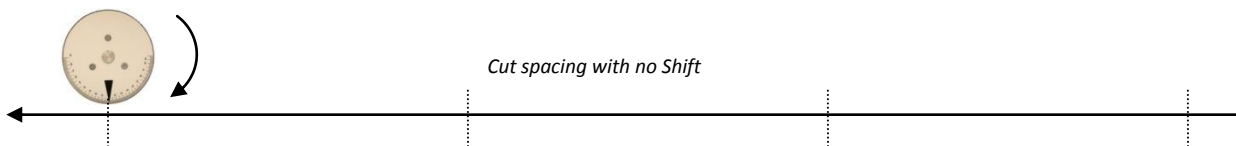
- Open "Shift_Running" POU
- Run the master slowly, engage the slave, and execute the shift (by toggling DI4)
- Adjust G_RunningShiftAmount
 - +9.0, +5.0, +1.0, 0.0, -1.0, -5.0, -15.0, -20.0, etc
- What is the maximum shift allowed?
- Is there a minimum shift? What happens with an extreme negative shift such as -40.0?



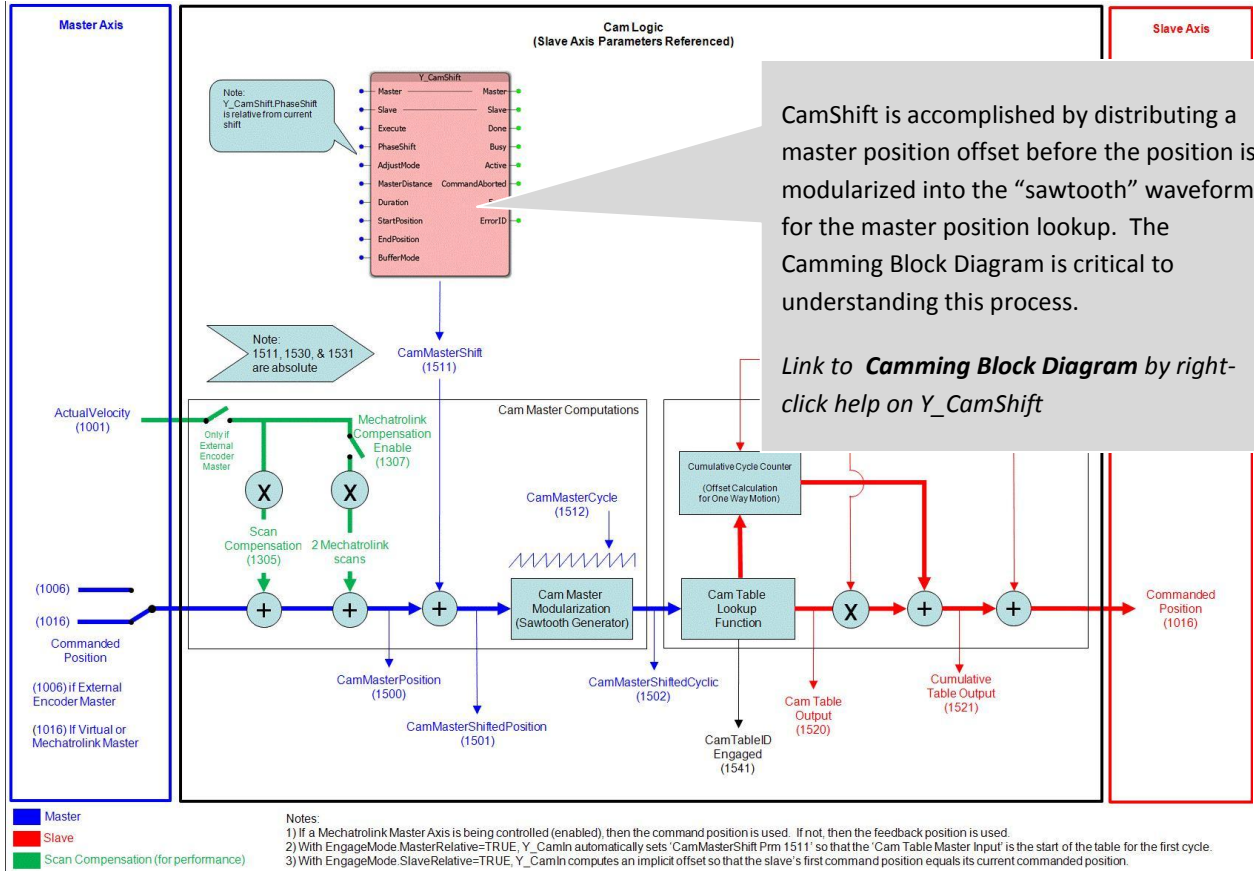
CAM SHIFT CONCEPT OVERVIEW

The concept is to shift the master during the running cam profile, either forward or reverse so that it speeds up or slows down, and returns to the normal cam profile at the required master phase position so that the knife cuts on the mark.

The amount of shift required depends on how much shorter or longer the cut spacing is. Think of it this way. What if there was no shift at all and the knife was simply "gearing" to the linear cam profile? This is the current condition of the project.



Shifting the slave while the cam is running allows us to shrink or grow this cut spacing. The slave and the cam block don't know about this; they just see the master position increasing and respond according to the cam lookup table. But the raw master position is not changed by shift, and neither is the position of the moving product. Instead, an intermediate CamMasterShiftedPosition(1501) is created inside the controller for reference.

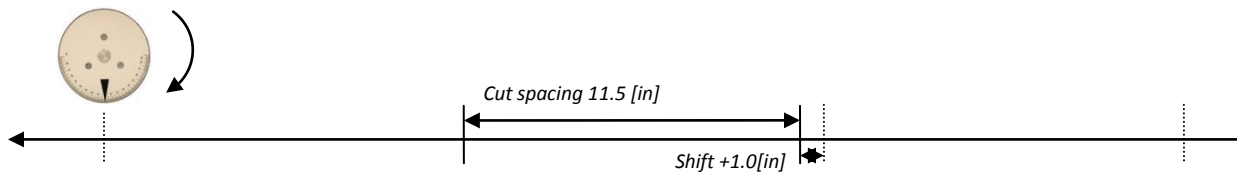


CamShift is accomplished by distributing a master position offset before the position is modularized into the "sawtooth" waveform for the master position lookup. The Camming Block Diagram is critical to understanding this process.

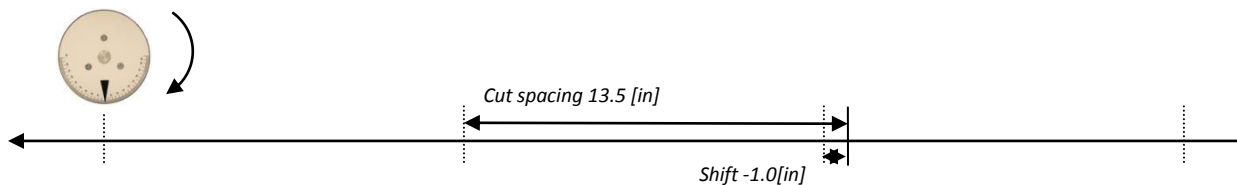
Link to **Camming Block Diagram** by right-click help on Y_CamShift

Consider the following example.

Shift the cam by +1.0 [in] (a forward shift) → Cut spacing will be $12.5 - 1.0 = 11.5$ [in].



If we shift the cam by -1.0[in] (a reverse shift), then the cut spacing will be $12.5 - (-1.0) = 13.5$ [in].



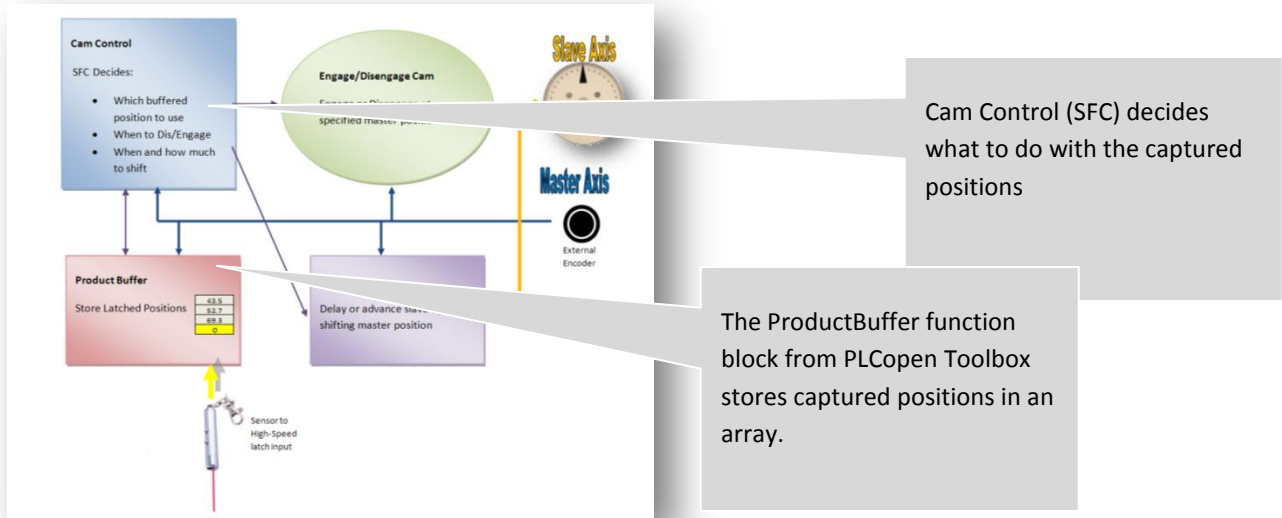
TRAINING ACTION

Observe the effect of positive and negative shifting

- What is the length of the part being cut without executing Y_CamShift ?
- What is the mathematical relationship between shift amount, part length, and master cycle?
- What is the shortest part length possible (with present values of Y_CamShift inputs)?
- What happens to the position of the master relative to the slave after a shift?
- Does the slave return to normal speed at the same position?
- Does the “cut point” relative to the master change with every shift?
- If you execute a positive shift “too late”, does it shift on the next cycle?
 - The Shift_ExecutionRange worksheet is calculating if there is still enough master cycle left to complete the shift. Otherwise Y_CamShift would generate error 4398.

PRODUCT BUFFER

You notice that, you cannot make the slave synchronize with any other point on the master until you shift the master. If you've homed both axes, the slave always engages such that the arrows of the two disks line up... until you shift. Shifting the master allows you to change the synchronize point. If the product positions were known to the controller, then we could use calculations to control the shift amount and synchronize the cut to each position. ProductBuffer from the Yaskawa's PLCopen Toolbox user library stores axis positions in an array so that several marks may be captured before the first mark arrives at the knife.



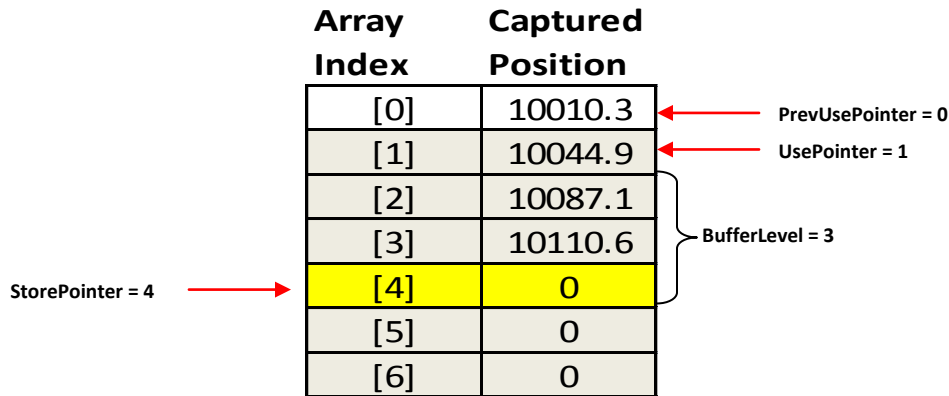
You can see in the above programming concept overview, that Product Buffer is just one component of the program. Cam Control decides what to do with the position in the buffer. Cam Control is another new POU that we will begin to write in this section.

The Cam function does not control the slave based on the raw master position, but rather the `CamMasterShiftedCyclic(1502)` position. Cam Control will intelligently shift the `CamMasterShiftedCyclic(1502)` position when the knife is not cutting, so that the knife rotates slower or faster, and ends up synchronizing with the mark.

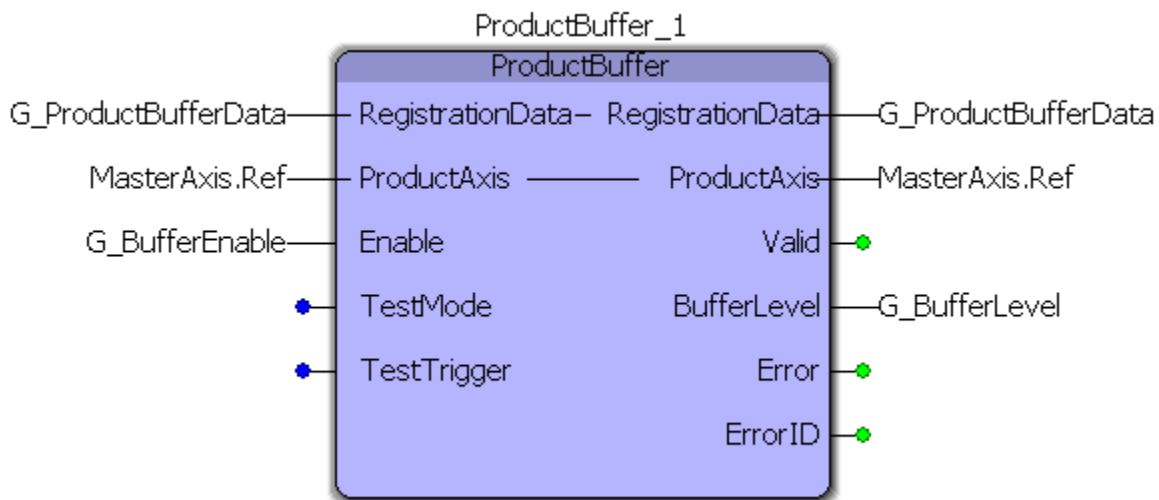
For a 12-minute recorded webinar presentation explaining ProductBuffer, please refer to the following video:

<http://youtu.be/SURwLkaeXeE>

ProductBuffer automatically captures the “raw” position of an axis at every trigger of the high-speed latch input. The high-speed input can be configured, but we will use input SI4 (EXT1). The positions are stored in an array, and works as a FIFO (First In First Out) circular buffer. The “StorePointer” contains the array index (array element number) where the next position will be stored.



The RegistrationData structure sets the array size, filter, position offset, and specification of which hardware input is to be used as the high-speed latch input. The “Cam Control” SFC-language program POU, which we will create, will update the “UsePointer”, used by ProductBuffer to calculate the BufferLevel.



Description of RegistrationData will be explored in the next Training Action.

PRODUCT BUFFER DEFINITIONS

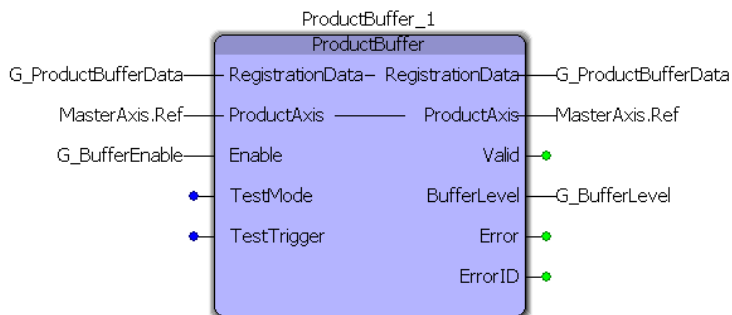
The controller can't do anything with the marks until it senses them. So the first step is to implement the ProductBuffer function block from Yaskawa's PLCOpen Toolbox user library.

ProductBuffer uses the latch input to capture an array of raw master positions. The latch input is connected to the sensor that detects the products as they move by. The sensor is wired to input SI4 to capture the position of a servo axis, or DI1 on an LIO card to capture the position of an external encoder. So every time the sensor sees a mark, the position from that axis is captured into an array.

The product buffer will make it possible to keep track of the master position of the marks on the belt, even if several of them are sensed before the first mark is cut. This is especially important for an application with a large distance between sensor and rotary knife.

TRAINING ACTION:

1. In the "StudentPOUs" folder, insert a new LD program POU named "Buffer", run it in the Fast task, and insert ProductBuffer from the Edit Wizard.
2. Connect variables as indicated below. You may wish to control G_BufferEnable with input DI5 by adding a rung to the Controls POU.
3. What is the data type of the variable "G_ProdBufferData" connected to the RegistrationData Var_IN_OUT?
4. Mark "retain" for G_ProdBufferData so that the values entered retain after warm start.
5. Download Changes.
6. Rename a watch page tab to "buffer" and add G_ProductBufferData so you can see the different elements of the structure.

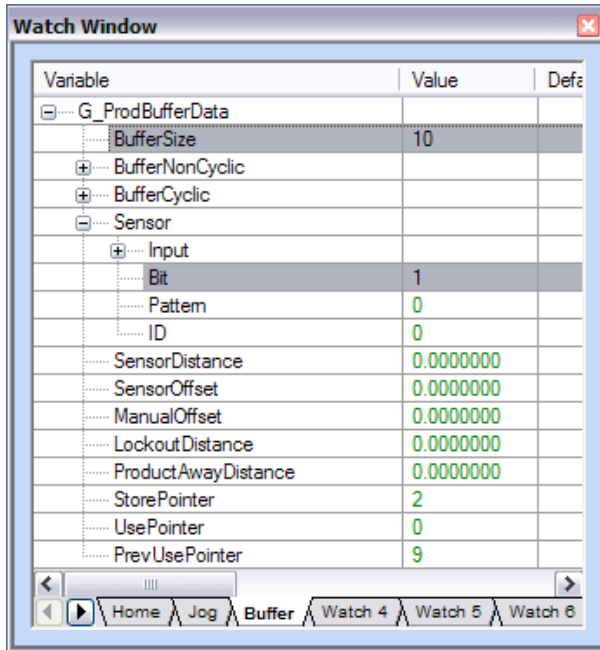


How do you decide what task should run Product Buffer?

Initially you may assume that you need a fast task so that no latches are missed. The latch function however operates in microseconds at the hardware level. Product Buffer only has to run fast enough to arm and re-arm the latch faster than the products come in. If minimum spacing is 5[in] and max speed 80 [in/sec] then the fastest product frequency is 62.5 ms – the 20ms task is fine.

However! We'll be using G_BufferLevel output elsewhere in the code and want to know as fast as possible if another product has been sensed, so the 2 ms fast task is appropriate.

You can see that the important inputs and outputs of ProductBuffer are all wrapped up within the RegistrationData Var_IN_OUT. This datatype and usage facilitates the implementation of ProductBuffer . Some of the elements of this structure act as inputs and other elements act as outputs.



PRODUCT BUFFER QUICKSTART

1. Set BufferSize=10
2. Set Sensor.Bit=1
3. Enable ProductBuffer block
4. Jog the master
5. Flip SI-4 on and off
6. Monitor Result in BufferNonCyclic

Please read the following explanations for the RegistrationData elements so that you can initialize the input elements according to the application specifications.

PRODUCT BUFFER INPUTS

BufferSize – Set the number of latched positions to be stored in the buffer. 5 to 10 is a typical setting, and 100 is the maximum. It is advantageous to use the smallest buffer size necessary so that it is easy to monitor the buffer in the watch window. This element must be set in order for ProductBuffer to run.

Sensor – Defines the high-speed input that will trigger the position latch. The Sensor element of the structure is also a structure itself – the TRIGGER_REF structure. The good news is that only the BIT element of the TRIGGER_REF structure is used.

The chart at the right is from the PLCopen help file for MC_TouchProbe. This shows that the default, Bit=0, latches on the encoder C-pulse signal. For a ServoPack master, Bit=1 corresponds to the EXT1 input (SI-4) as the latch input. For the LIO cards with encoder input, Bit=1 corresponds to DI-01 on the card. So in most cases the BIT element is set to 1.

Initialize the Sensor to use SI-4 of the master axis with the following line of code (in the next training action).

```
G_ProdBufferData.Sensor.Bit:=UINT#1;
```

LockoutDistance – You don't want extraneous marks to register another capture. So give a minimum master distance at which new marks can be sensed again. As a minimum, LockoutDistance can be set to the theoretical minimum product spacing, as calculated based on the MasterSyncDistance. In practice, this value is set based on the application.

UsePointer – Array index of the latch data to be used in the application program, ie Y_CamShift block. The value of UsePointer must be controlled elsewhere in the program. In this program it will be controlled in the Cam_Control_SFC program POU. The initial value is typically zero.

ManualOffset – A small offset may be added to the captured positions. This value is adjusted for the machine based on the output of the machine. You could also simply adjust the sensor distance.

Device	Signal	Hardware Pin #	Software Default Variable Name	Bit
LIO-01	Encoder C Channel	A3/B3	n/a	0
	DI-01	A22	M□□_DI_01	1
LIO-02	Encoder C Channel	A3/B3	n/a	0
	DI-01	A22	M□□_DI_01	1
LIO-06	Encoder C Channel	35	n/a	0
	DI-01	39	M□□_DI_01	1
MP2600	External C Channel	35	n/a	0
	Cn13 DI-01	39	MO1_DI_01	1
SGDH	C Channel	n/a	n/a	0
	EXT1	44	AX□□_SI4_EXT1	1
	EXT2	45	AX□□_SI5_EXT2	2
	EXT3	46	AX□□_SI6_EXT3	3
SGDS	C Channel	n/a	n/a	0
	EXT1	10	AX□□_SI4_EXT1	1
	EXT2	11	AX□□_SI5_EXT2	2
	EXT3	12	AX□□_SI6_EXT3	3
SGDV	C Channel	n/a	n/a	0
	EXT1	10	AX□□_SI4_EXT1	1
	EXT2	11	AX□□_SI5_EXT2	2
	EXT3	12	AX□□_SI6_EXT3	3

GUIDELINES FOR SETTING BUFFERSIZE

What is the maximum number of marks that could possibly be sensed and waiting to be "cut"? This would occur with the closest possible mark spacing. A longer SensorDistance will require a larger BufferSize. If the required BufferSize is larger than 100, then the PLCopen Toolbox user library project must be modified.

Steps to modify:; open the "PLCopen Toolbox" project, Data Types folder, Toolbox_Data Types, and find the line LatchBufferArray: ARRAY[0..100] OF LREAL; Increase "100" to a suitable number. We recommend saving the library project under a new name. Re-MAKE the project. In your application project, delete the existing PLCopen Toolbox library, and re-import the updated library with the new name you have chosen.

PRODUCTBUFFER OUTPUTS

BufferNonCyclic – the raw master position will be recorded in this array. It is parameter 1016

BufferCyclic – the cyclic master position will be recorded in this separate array. This only applies if the master is configured as rotary in the Hardware Configuration, and it uses the master cycle defined in the HWConfig. This is NOT referring to the master cycle of the cam profile. Remember, different slave axes can have different master cam cycles when camming from the same master. We don't need this buffer for this application.

Valid – means the buffer is running and all inputs are ok

BufferLevel – number of latched positions in the buffer. If the BufferLevel grows larger than BufferSize, then the block will stop with an error. $\text{BufferLevel} = \text{StorePointer} - \text{UsePointer}$. The code will increment UsePointer.

StorePointer – Array index in which the next latch position will be stored. Once a position is latched, StorePointer automatically advances to the next element in the array.

REGISTRATION DATA NOT USED BY PRODUCTBUFFER

PrevUsePointer – PrevUsePointer is not used by the ProductBuffer block, but is included so that the same structure can pass all the required information to other blocks provided in Yaskawa's CamToolbox user library. The value of PrevUsePointer must be controlled elsewhere in the program. In this program it will be controlled in the Cam_Control_SFC Program POU. The initial value would typically be set to the last point in the buffer. Use to keep track of the use pointer's previous value, ie to find the position of the previous latch. The distance between latches can be calculated by the difference between the position value at the UsePointer index and the position value at the PrevUsePointer index.

SensorDistance – SensorDistance is not used by the ProductBuffer block, but is included so that the same structure can pass all the required information to other blocks provided in Yaskawa's CamToolbox user library. Even so, SensorDistance is a noteworthy measurement. It is the distance from the sensor to the location of rotary knife BDC. Refer to the diagrams in the application description. When the mark moves this far away from the sensor, the master should have already been shifted by your application code so that the CamMasterShiftedCyclic(1502) position is zero and a new cam cycle is starting. The SensorDistance can be measured on the machine and is roughly 28.5 [in].

ProductAwayDistance – ProductAwayDistance is not used by the ProductBuffer block, but is included so that the same structure can pass all the required information to other blocks provided in Yaskawa's CamToolbox user library. Even so, Product Away Distance is an important measurement. It is the distance from sensor at which the knife has completely finished cutting. Refer to the diagrams in the application description. When the mark moves this far away from the sensor, your application code will increment the UsePointer and PrevUsePointer. The exact ProductAwayDistance can be calculated based on the SensorDistance and the SyncAngle, or can be set arbitrarily to a value greater than SensorDistance.

SensorOffset - SensorOffset is not used by the ProductBuffer block, but is included so that the same structure can pass all the required information to other blocks provided in Yaskawa's CamToolbox user library. Intended use is to calculate the remainder of master cycles. This will be accomplished in this application using SensorDistance instead of SensorOffset.

PRODUCTBUFFER EXECUTION

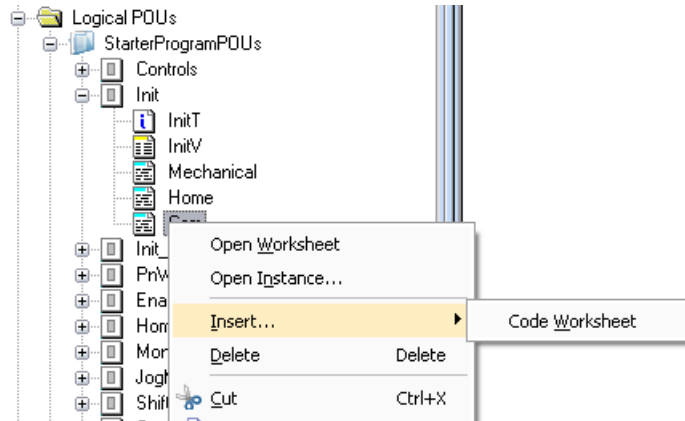
Test the operation of the ProductBuffer function block

TRAINING ACTION:

- Initialize all five of the input elements of “G_ProdBufferData” in a ST POU.

Detail for initializing G_ProdBufferData

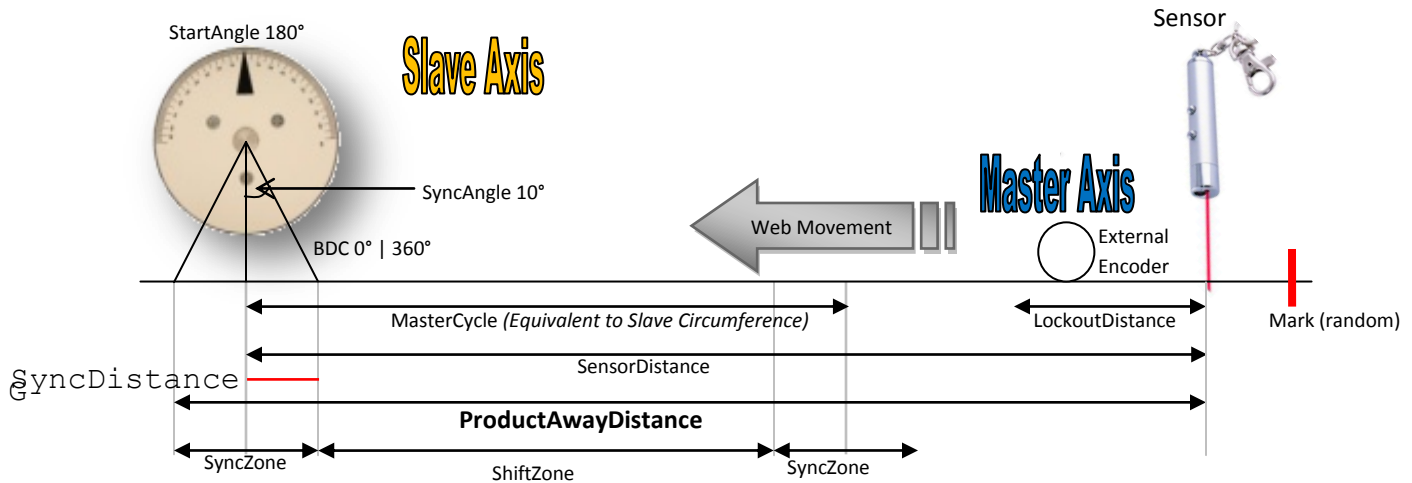
- **Insert a new ST code worksheet named “Buffer” into the Init POU**



- **Initialize the five input elements of G_ProductBufferData using function key F5**
 - Example: `G_ProdBufferData.UsePointer:=INT#0;`
 - Remember to use F5
- Warm start is required for the Init POU to run
- Enable ProductBuffer function block
- Toggle SI4 on the demo and confirm operation
- Add G_ProdBufferData.UsePointer to the watch window. Manually change the value and observe BufferLevel
- Make changes as required to the Init.Buffer worksheet so that the buffer will operate properly after warm start.

PRODUCT AWAY

After the master has jogged a distance of **ProductAwayDistance**, then the current latched “product” is out of the picture and the code can deal with the next product. Every position latched in the ProductBuffer has a corresponding **ProductAwayPosition**, which can be immediately calculated by adding the ProductAwayDistance to the position stored in the buffer.



TRAINING ACTION:

Use the following questions as a guide to generate a line (or 2 lines) of ST code to calculate the ProductAwayPosition for a given position captured in the buffer.

- If the ProductAwayDistance is 30.0 [in] and a position is captured in the buffer at master position 10044.9

$$360 = 12.566 \text{ [in]}, \text{ what is the the ProductAwayPOSITION?}$$

$$10 = \text{syncDist}$$

$$10074.9 \text{ (add 30 in)}$$

Array Index	Captured Position
[0]	10010.3
[1]	10044.9
[2]	10087.1
[3]	10110.6
[4]	0
[5]	0
[6]	0

UsePointer = 1 (points to index [1])

BufferLevel = 3 (bracketed around indices [2], [3], [4])

StorePointer = 4 (points to index [4])

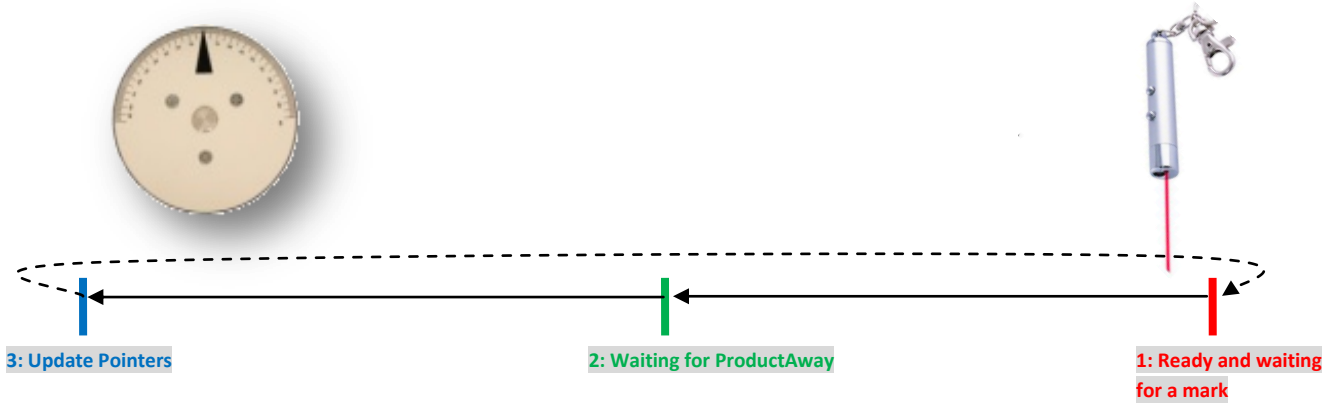
- What is the syntax to access the data stored at index UsePointer=1 for this array?
 - Example: IEC61131-3 Syntax to access the data stored at array index UsePointer=1 for an array named “MyArray” is MyArray[1]. Notice the square brackets. The array used by ProductBuffer is an element of the G_ProdBufferData structure, and it is named “BufferNonCyclic”. The array index for the current position in use (number within the square brackets) can be replaced by a variable. What is the variable used by ProductBuffer?
- Write an equation of pseudo-code to calculate the ProductAwayPOSITION.

$$\text{ProductAwayPOSITION} = \text{MyArray}[1] + \text{ProductAwayDistance}$$

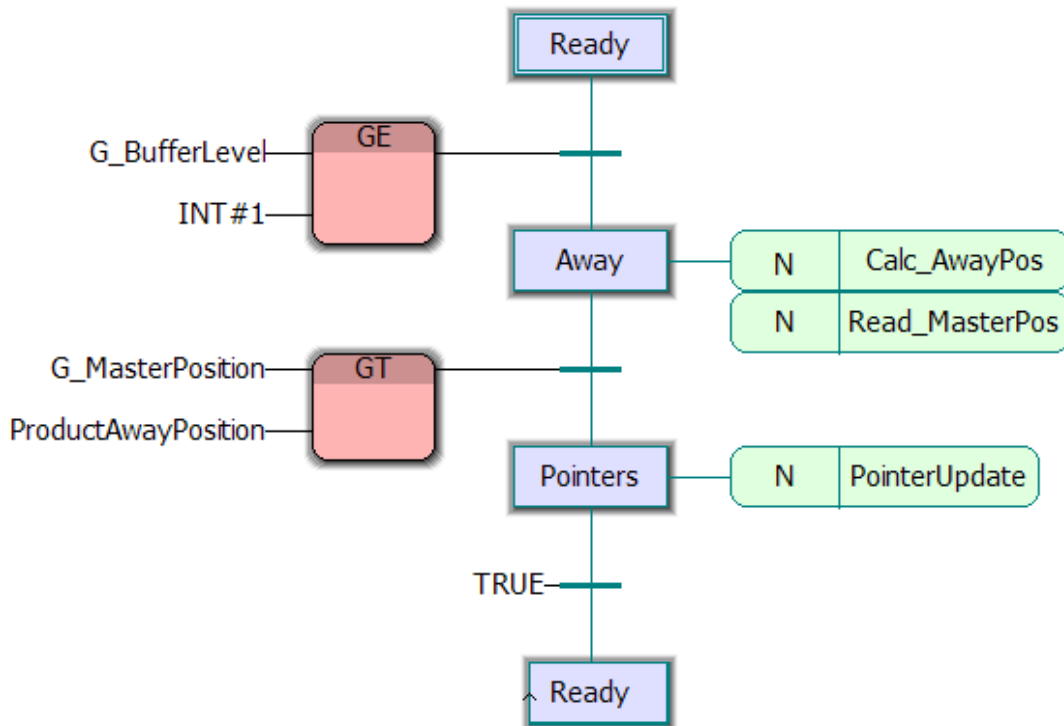
TOP-LEVEL SEQUENCE

While any of the programming languages could be used, the SFC language lends itself well to the sequence involved for controlling pointers, engaging the cam, and shifting. While the SFC controls the top-level logical flow, the working-level code is contained with the individual actions and transitions, programmed in ladder, function block, or structured text.

If you think about the product as they move past the sensor and then away, it is a very sequential process.



This sequence can be expressed in SFC. To the right of each step are the actions which can happen at that step. The transitions that connect the steps define the condition that must be true before progressing to the next step.

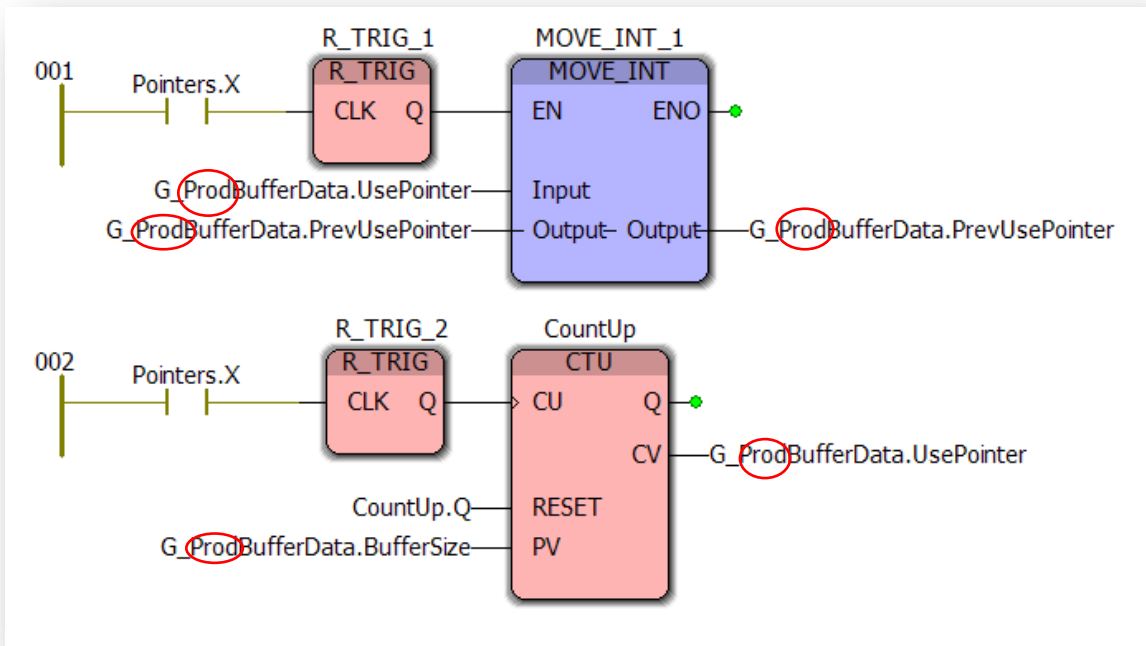


TRAINING ACTION:

Control the increment of UsePointer and PrevUsePointer in the application code

- In the “StudentPOUs” folder, create a new SFC Program POU named “Cam_Control_SFC” and run in Fast task
- Construct the SFC according to the previous illustration
 - **EN/ENO** can be removed in the Objects menu or under Extras-Options-GraphicalEditor
 - Code for the **Calc_AwayPosition** action can be accomplished in ST based on the pseudo-code equation in the previous Training Action
 - The **Read_MasterPos** action a simple one-function-block action using MC_ReadActualPosition
 - The code for the **PointerUpdate** action is given below
- If you don’t know how to get started with an SFC POU, please refer to the pre-requisite for this class, *eLM.MotionWorksIEC.01.ProSFC* eLearning module, available on YouTube at <http://youtu.be/OzAxCNoGbt0>
- Verify operation of the code by jogging the master and toggling the latch input SI4

PointerUpdate

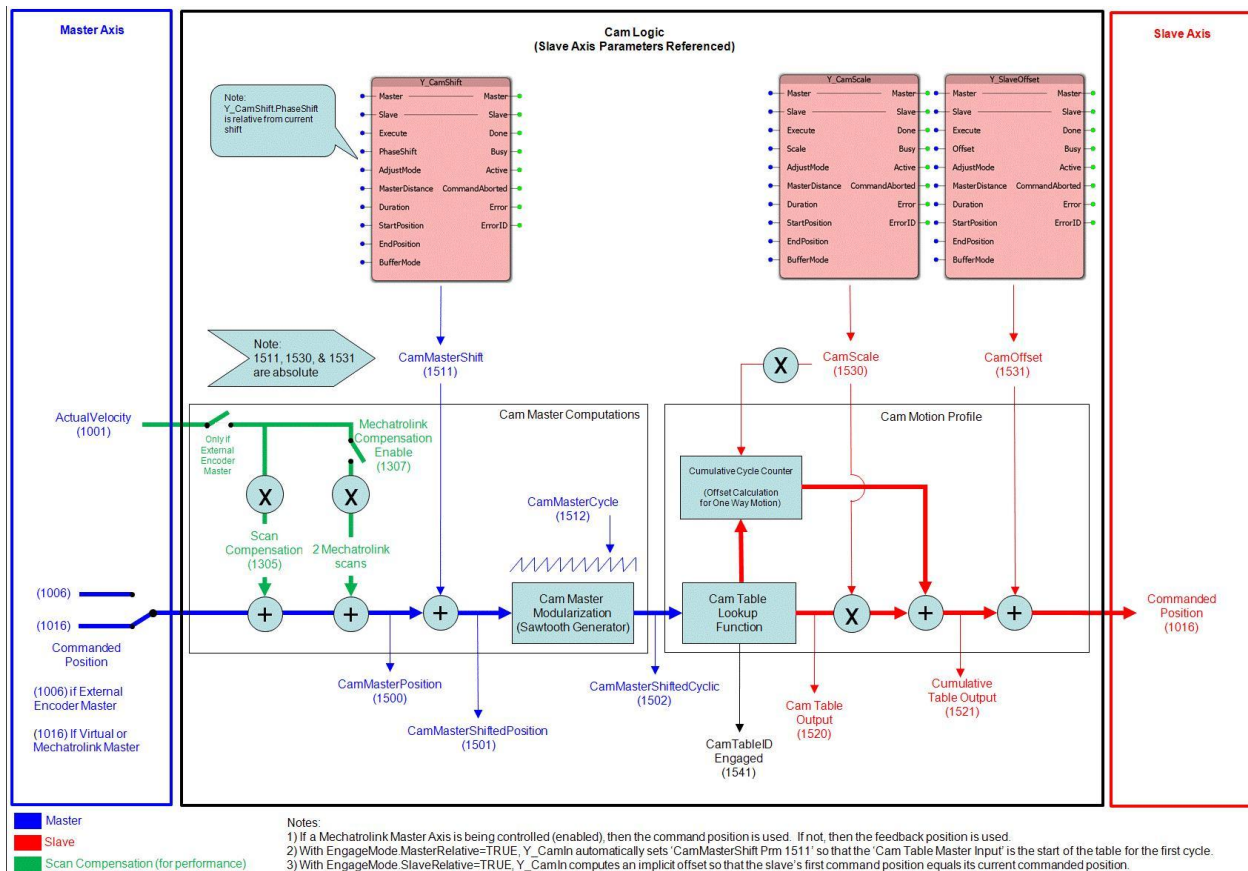


FIRST SHIFT AND ENGAGE

Right now we can engage the cam and it will blindly cut parts of length equal to the knife circumference, 12.566[in]. We can disengage the cam and the knife will stop pointed up at the 180 ° mark. We also have the buffer running and so we're able to keep track of the positions of the products as they pass the sensor, and then move "away" past the sync zone. But these two operations at this point are not connected in any way. We have to make the knife engage such that it will hit the first mark, not just engage at the next master cycle. And then we have to make the knife speed up or slow down in between cuts, depending on the spacing between products. As previously discussed, this can be accomplished by using a separate Y_CamShift block. The first Y_CamShift block will be applied before the cam engages so that when it does engage, it will line up with a mark. The second Y_CamShift block will be applied continuously between cuts while the cam is engaged, thus affecting the observed speed of the rotary knife. Finally, a third Y_CamShift block will "undo" all the previous shifting so that the whole sequence can start over.

Y_CamShift help links to the Camming Block Diagram that describes how Y_CamShift fits into the electronic camming algorithm. It is worthwhile to examine and understand this diagram. In order to do so, please refer to this diagram and answer the following questions.

CAMMING BLOCK DIAGRAM



TRAINING ACTION:

- What is the name and parameter number of the position input used by the Cam Table Lookup Function?
- What is the name and parameter number of the position input captured by ProductBuffer? (Open the block and see the code)
- What is the name and parameter number of the position read by MC_ReadActualPosition?

Notice that the raw axis position, CamMasterPosition(1500) is preserved for reference. But the master position is compensated, shifted, and finally modularized into the cyclic “saw tooth” waveform. The saw tooth waveform, CamMasterShiftedCyclic(1502) is used by the cam table lookup function, which ultimately controls the slave. This means that as we use Y_CamShift in real time, both the saw tooth and the slave itself will be affected in real time. This way, compensation for random spacing between parts can be achieved without disengaging the cam.

While the machine is running products, the concept is to shift the master during the running cam profile, either forward or reverse so that it speeds up or slows down, and returns to the normal cam profile at the required master position so that the knife cuts on the mark. But there is no “spacing” for the first cut. So there has to be an initial shift before the first cut to get everything started off right.

What must be known in order to perform the initial shift be so that the knife will cut on the mark?



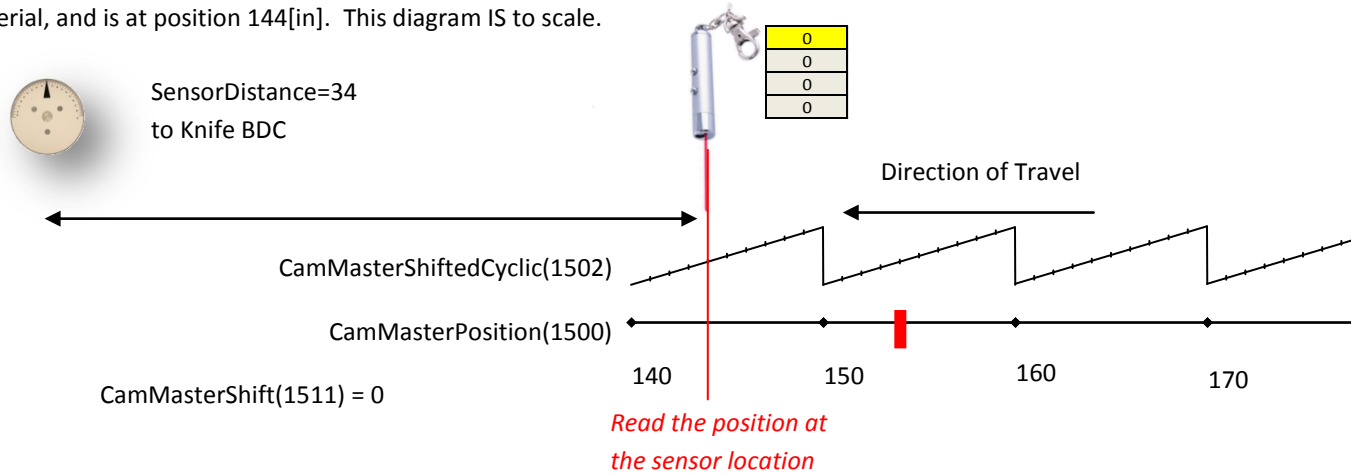
In the end, there will be 3 individual Y_CamShift blocks. We will implement these in the code one by one.

1. First shift (immediate shift before engage)
2. Running shift (during shift zone to compensate for random product spacing)
3. Final shift (immediately unshift after disengage)

FIRST SHIFT SIMPLIFIED EXAMPLE

The first cut is special. The buffer was empty, so the PrevUsePointer data is empty, and it makes no sense to calculate the part spacing. The cam is not engaged; the slave is disengaged from the master. The master is moving, but the slave is not. We need to engage the cam such that the slave hits that first latched position. How? Let's look at what's going on graphically

Consider a simplified version of our application with "easy" integer numbers. Let's say the CamMasterCycle(1512) is 10 [in] and the sensor is measured at 34[in] from the knife BDC. Additionally, the buffer is empty and we're waiting for that first latched position. The cam is a 1:1 profile – as the master moves from 0 to 10[in], the slave moves from 0 to 360 degrees. The slave engages/disengages halfway through the master profile at CamMasterShiftedCyclic(1502) position 5[in], leaving the slave pointing up at 180 degrees. The master is moving the material, and is at position 144[in]. This diagram IS to scale.



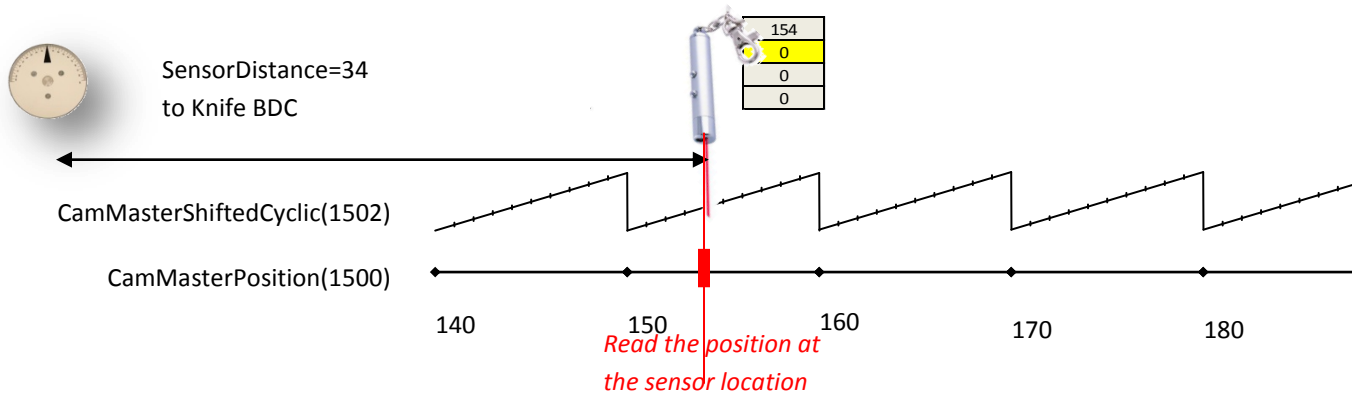
TRAINING ACTION:

It is beneficial to think of the master position as being read at just ONE reference point on the physical machine. In this case, it makes sense to use the latch sensor as that reference point.

- What is the current value of CamMasterPosition(1500) according to this diagram?
- What is the current value of CamMasterShiftedCyclic(1502) position according to this diagram?

In this diagram, you can see that the master position is 144, but the saw tooth CamMasterShiftedCyclic(1502) position used for the cam function block is 4. Refer to the Camming Block Diagram.

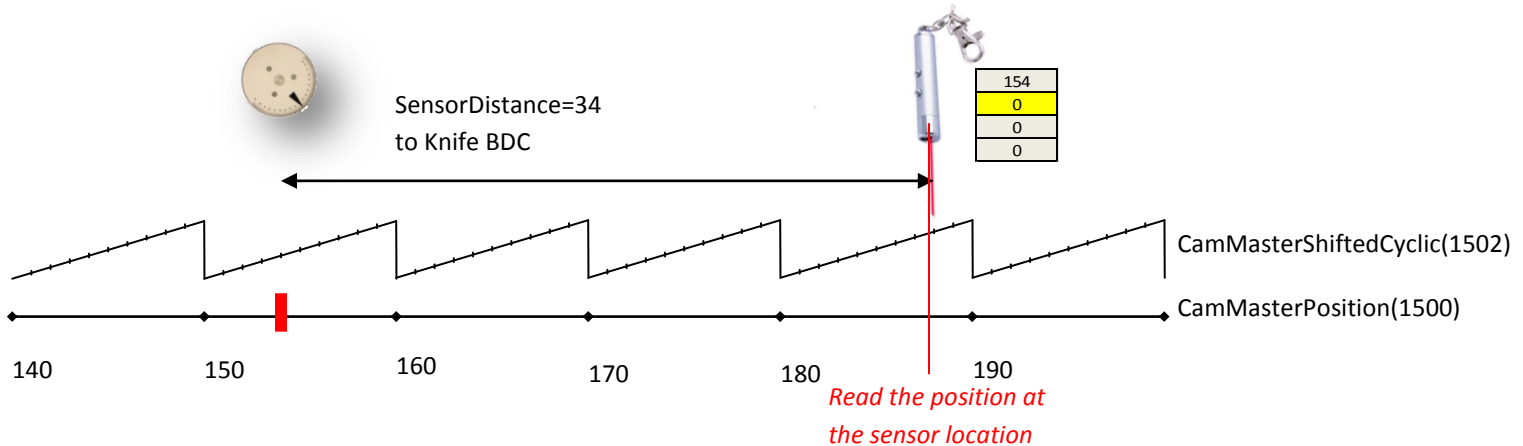
The master keeps moving and latches a product at $\text{CamMasterPosition}(1500) = 154[\text{in}]$



TRAINING ACTION:

- Where will the knife be oriented at the start of the cam cycle ($\text{CamMasterShiftedCyclic}(1502) = 0$)?
- Which values of $\text{CamMasterPosition}(1500)$ correspond to the start of a cam cycle?
- Will the knife cut on the mark?

At first glance, you might think that the mark has to land on one of the multiples of the master cycle in order to cut without a first shift. But that's not true; the sensor distance must be considered. Fast-forward the diagram to the point when the mark reaches BDC. Refer to the diagram and answer the questions.

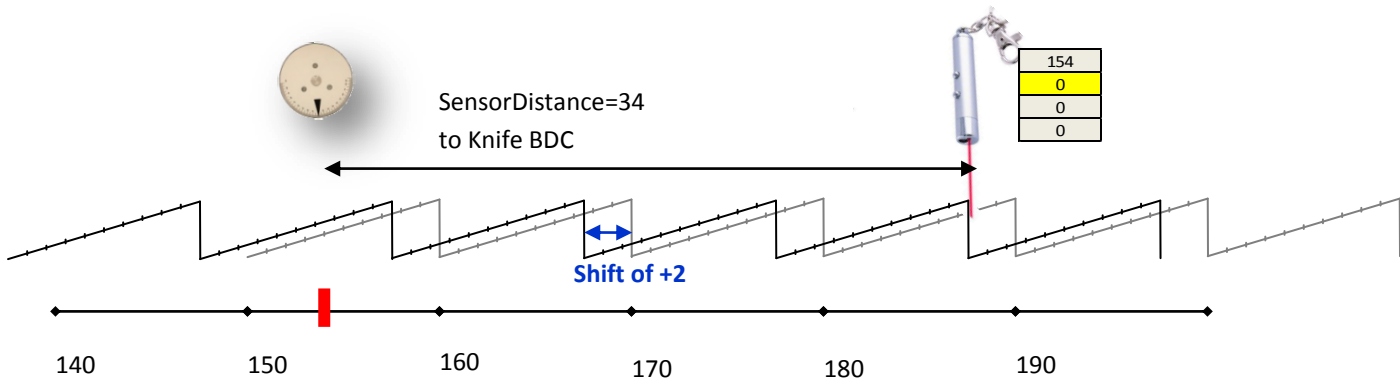


QUESTIONS – LATCH AT BDC:

- What will be the $\text{CamMasterPosition}(1500)$?
- What will be the $\text{CamMasterShiftedCyclic}(1502)$?

From the previous illustration, we can see that the mark will be missed, as the next cam cycle comes at position 190, not 188. “If only the mark had been sensed at 156 or 146 instead of 154; then the knife could have cut it!” But... the cam lookup table uses the CamMasterShiftedCyclic(1502) position. We can “trick” the cam lookup function and first shift the master by +2 [in] or by -8[in] so that the mark lines up with the start of a new master cycle.

The diagram below shows a shift of +2[in] (or -8[in]), with the original **un-shifted CamMasterShiftedCyclic sawtooth in gray**.



If we would shift the CamMasterShiftedCyclic(1502) position by +2[in] (or -8[in]), then the first mark would be at BDC when the CamMasterShiftedCyclic(1502) position = 0, the start of a new cam cycle.

FIRST SHIFT EQUATION

Obviously, the shift of 2 will not work in every case. There is a mathematical way to determine ahead of time how much the shift will be. How did we arrive at a shift of 2? The following mathematical formula describes the above example for the general case.

Latched position + SensorDistance = first BDC master position.

This is what the master position will be in the future, when the mark is directly at BDC of the knife

FirstShiftAmount = $-1 \times \text{REM}[(\text{LatchedPosition} + \text{SensorDistance}) / \text{MasterCycle}]$

“REM” means “remainder”.

Math

REM can be found in the Edit Wizard under the **Yaskawa_Toolbox** user library.

Examples for case of SensorDistance=34, MasterCycle=10

- LatchedPosition=154
 - FirstShiftAmount= $-1 \times \text{REM}[(154+34)/10]$
 - $-1 \times \text{REM}[(18 \text{ with remainder } 8)]$
 - FirstShiftAmount = -8
- LatchedPosition = 156
 - FirstShiftAmount = $-1 \times \text{REM}[(156+34)/10]$
 - $-1 \times \text{REM}[(19 \text{ with remainder } 0)]$
 - FirstShiftAmount = 0
- LatchedPosition = 158
 - FirstShiftAmount = $\text{REM}[(158+34)/10]$
 - $-1 \times \text{REM}[(19 \text{ with remainder } 2)]$
 - FirstShiftAmount = (-2)

TRAINING ACTION

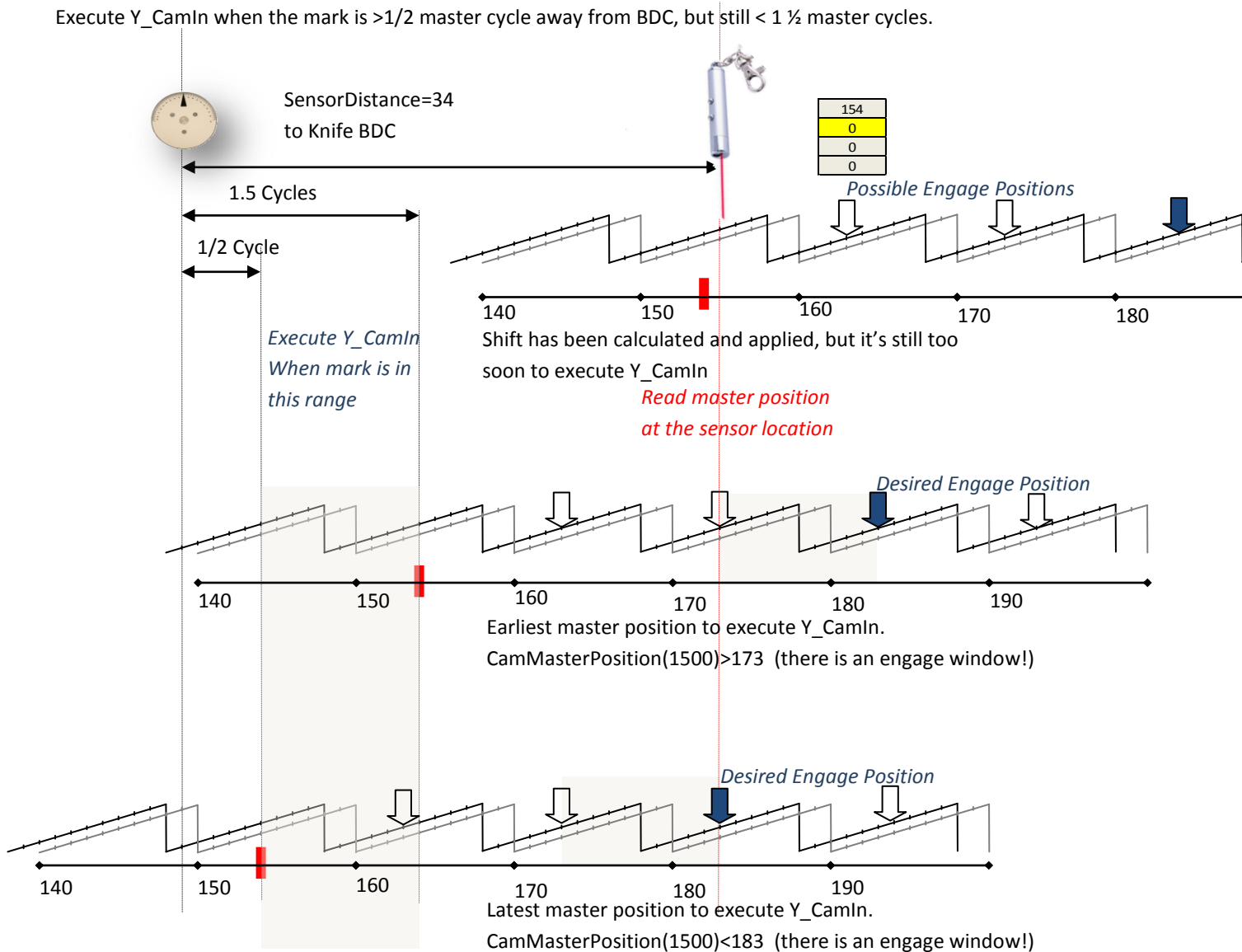
- What conditions must be true in order for the controller to be able to make this calculation?
- What conditions must be true in order for the controller to execute the First Shift?
- What conditions must be true before the cam is engaged?

Y_CAMIN EXECUTION ZONE

If you execute Y_CamIn immediately, what will happen? ANSWER: the cam is set to engage when CamMasterShiftedCyclic(1502) position = (master cycle / 2). In this simplified example, that would be at position 5[in]. The cam would engage the first time that CamMasterShiftedCyclic(1502) position=5. This means we can't execute Y_CamIn immediately because it would engage too soon and miss the mark by one cycle. More cycles have to go by before we tell it to engage. The code has to wait to execute Y_CamIn.

Use the following diagram to answer the questions below.

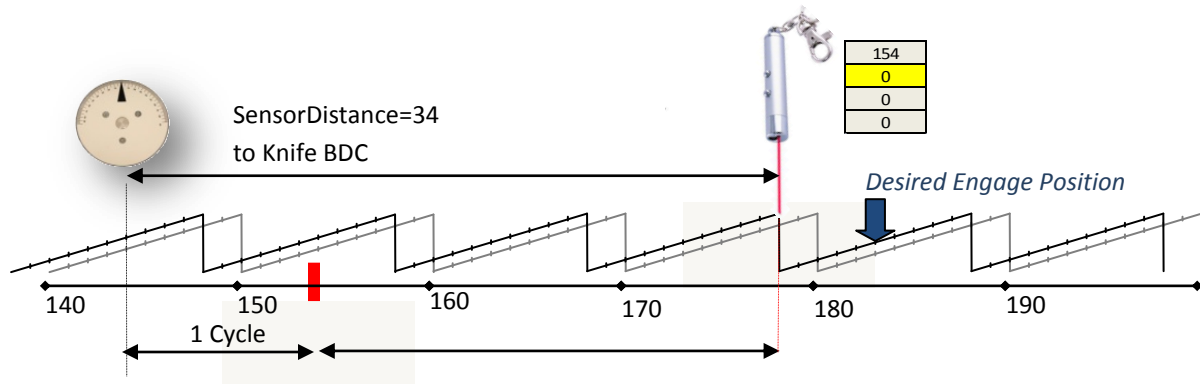
Execute Y_CamIn when the mark is $>1/2$ master cycle away from BDC, but still $< 1 \frac{1}{2}$ master cycles.



QUESTIONS:

- What is the lowest(first) theoretical CamMasterPosition(1500) at which the cam could be engaged?
- What is the highest(last) theoretical CamMasterPosition(1500) at which the cam could be engaged?
- What CamMasterPosition(1500) is in the "middle of the zone"?

Continuing the simple example you see that Y_CamIn could be executed when the CamMasterPosition(1500) is between 173 and 183. You probably wouldn't want to pick 173 because the EngageWindow input on the block may be such that the cam engages a cycle early. You probably wouldn't want to pick 183 either, because at faster speeds, execution delays may cause the mark to be missed and the cam would engage one cycle late. So the sweet spot is probably somewhere in the middle; when the CamMasterPosition(1500) = 178. Execute Y_CamIn at position 178, it will engage at 183, and the next cycle will begin at 188 with knife and mark both at BDC.



Optimal position to execute Y_CamIn is in the "middle of the zone". CamMasterPosition(1500) = 178 in this example

PITFALL: SENSOR LOCATION

This discussion opens up the critical point of Sensor Location. The sensor must NOT be located too close to the knife! A distance of 2.5 master cycles is a good rule of thumb.

If the sensor is too close to the "action", then at high speed any controller doesn't have enough time to react before the fast-moving mark is at the cut point. Classic physics still applies and $Time = Distance / Speed$ cannot be avoided. A short distance and high speed gives very little time for any servo system to react.

The sensor distance must be great enough to allow the cam to both disengage, and then engage again for a new mark. This situation occurs when the last product in the buffer is about to be cut. We have not yet discussed the "Running Shift" yet, but as a little preview, the controller has to make the decision to disengage during the current cycle if there are no more products. When we use CamBlend, it will require two complete master cycles to disengage and engage. But if during that cycle, another product is latched in the buffer, the cam still has to complete the disengage cycle, and must be able to complete a new engage cycle to line up with the new mark. The entire time those two cycles go by, that latched product is getting closer and closer to BDC. So the sensor must be AT LEAST 2 master cycles away. Currently the slave disengages abruptly and could reengage immediately within the same cycle, but once CamBlend is implemented, it will require one master cycle to disengage, and another master cycle to engage again.

On the other hand, if the sensor is too far away from the "action", then inaccuracies (stretch, slippage) may have accumulated once the latched product gets to BDC. Also a very distant sensor means that many products will be buffered, which for some machines implies the possibility of wasted product in an E-stop situation.

PROGRAMMING: FIRST SHIFT AND ENGAGE

TRAINING ACTION:

- The CamIn_Position for the simplified example was 178 based on:
 - Latched Position = 154
 - Sensor Distance = 30
 - Master Cycle = 10
- What is the mathematical equation that defines CamIn_Position?
- Based on the above discussion, what two major transitions occur in the sequence between Ready and Away on the Cam Control SFC in order to implement the first shift?

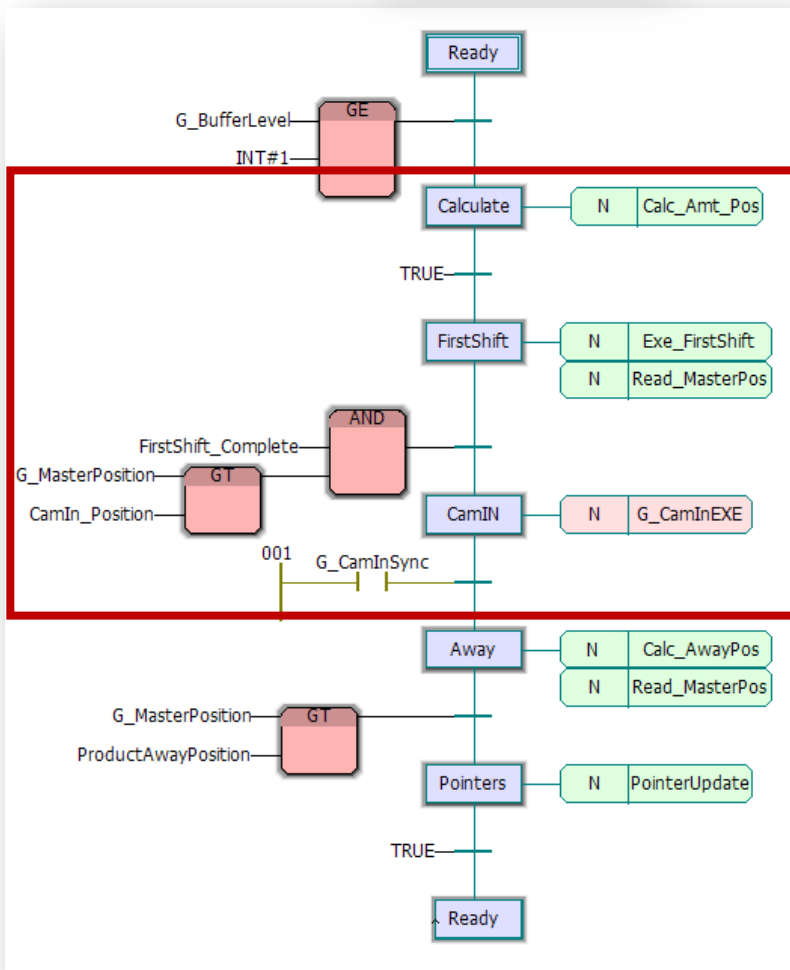
The following bullet items are a specific list of what has to happen between Ready and Away. Each can be represented in the SFC code as either an ACTION or TRANSITION. There is some flexibility with the SFC language regarding the way in which actions and transitions can be used. But generally, the action is the “doing” of something, and the transition is the “waiting” for some other event to happen. According to this guideline, the bullet items that “calculate” or “execute” constitute actions. The “waiting” bullet items constitute transitions.

- Calculate shift amount for FirstShift
- Calculate position at which to execute Y_CamIn (CamIn_Position)
- Execute FirstShift
- Wait for confirmation that FirstShift has completed
- Wait for Y_CamIn execution position to arrive
- Execute Y_CamIn
- Wait for confirmation that cam is engaged

TRAINING ACTION:

- Add functionality to the existing SFC code to execute the First Shift.
- Use this SFC as a guide, and create the required actions and transitions.

SFC EDITING TIP: Click on the transition below the Ready step, then “Create step-transition sequence”



SFC ACTIONS

CALC_AMT_POS:
 $\text{FirstShiftAmount} = -1 \times \text{REM} \left(\frac{\text{LatchedPosition} + \text{SensorDistance}}{\text{Master Cycle}} \right)$

$\text{CamIn_Position} = \text{LatchedPosition} + \text{SensorDistance} - \text{MasterCycle}$

EXE_FIRSTSHIFT:
 Implement Y_CamShift block for immediate shift by calculated amount. See next page.

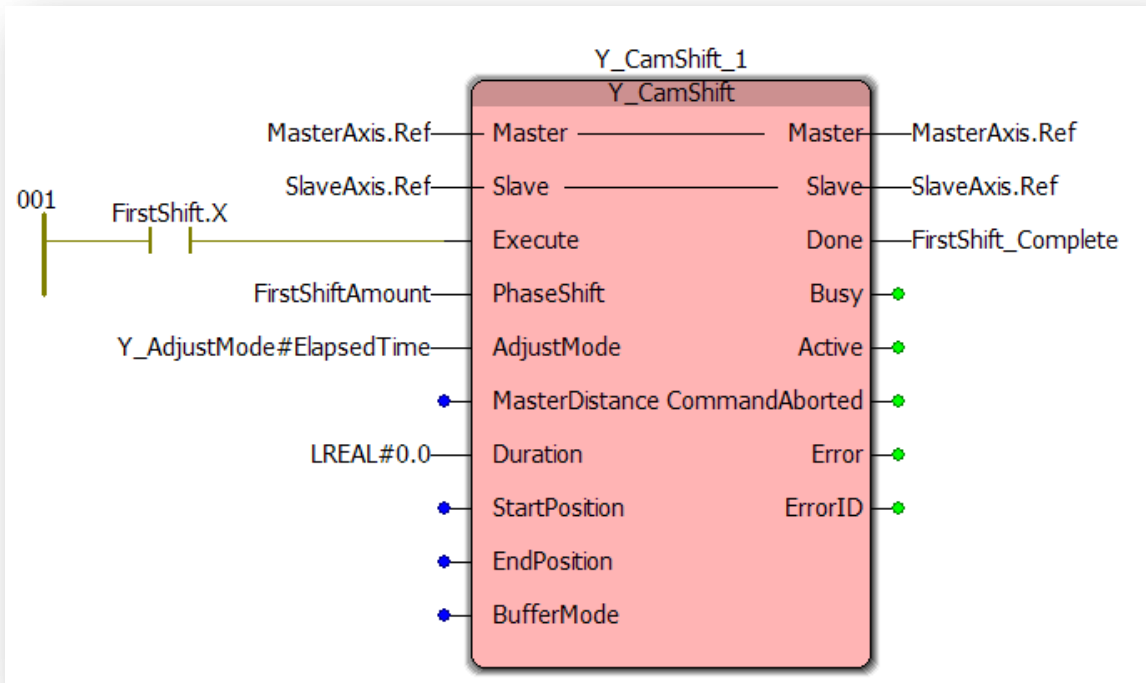
READ_MASTERPOS:
 Same action as Away step

G_CAMINEXE: the existing variable to execute Y_CamIn from CamInOut POU.
 Note: remove control with DI-3 switch.

NOTE: SFC executes all actions associated with a step simultaneously. It does not first evaluate the actions at the top of the step before applying the result to other actions in the same step. Therefore it is important to calculate the values (for FirstShiftAmount and CamIn_Position) in a separate step (Calculate) before they are used as inputs in the following step (FirstShift).

EXE_FIRSTSHIFT ACTION

- The EXE_FirstShift action contains its own Y_CamShift block.
- Do NOT adjust the Y_CamShift block in the Shift_Running POU. It will be used for the running shift in the next section.
- The first shift is to be executed immediately, so set the AdjustMode=ElapsedTime with Duration 0.
- FirstShift.X is the special “step.x” bit which turns on when the step is active and off when not active.



- Run, test, troubleshoot your code. Use debug mode and verify that each step is working as expected.
- It should line up with the first mark. Subsequent marks will be ignored, but you can still manually execute the running shift.
- Warmstart the controller to test the first cut more than once.
- Keep your hand on the jog button so you can stop and start the master, “freezing time” so you can verify and troubleshoot your code.
- Adjust code in “Controls” POU to bypass DI-3 (ie, delete the ladder rung).

RUNNING SHIFT

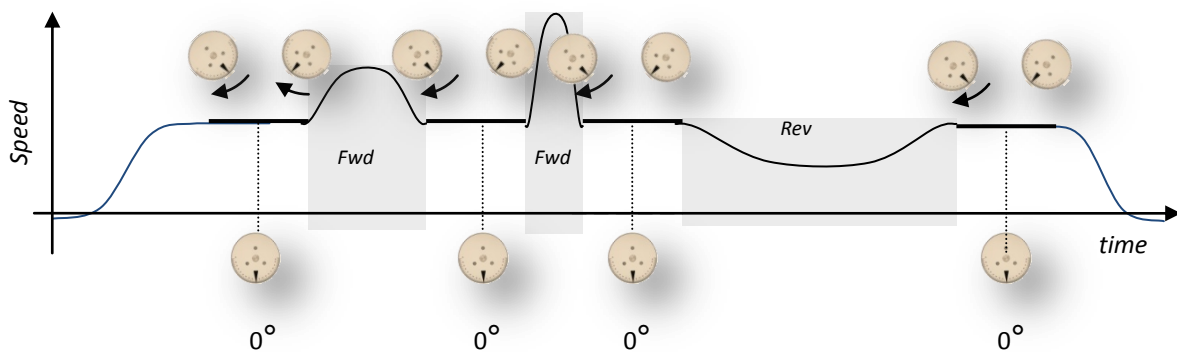
The code as it stands now lines up the first mark, but then continues camming without shifting to accommodate the positions latched in the buffer. The next step is to apply a shift to the cam while it's running – but not while it's cutting.

The concept is to shift the master during the running cam profile, either forward or reverse so that it speeds up or slows down, and returns to the normal cam profile at the required master position so that the knife cuts on the mark.

SHIFT ZONE

Synchronization is lost between master and slave during the shift. Synchronization must be maintained during the cut. So there are effectively two zones in the rotary knife; the “sync zone” and the “shift zone”. The shift must take place inside the shift zone only so that the knife remains synchronized during the cut.

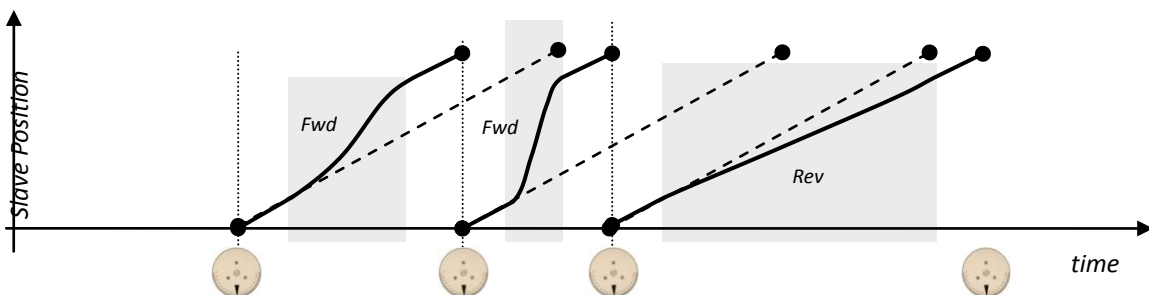
An example of the expected slave speed profile during different running shifts is shown below. The shift zones are highlighted.



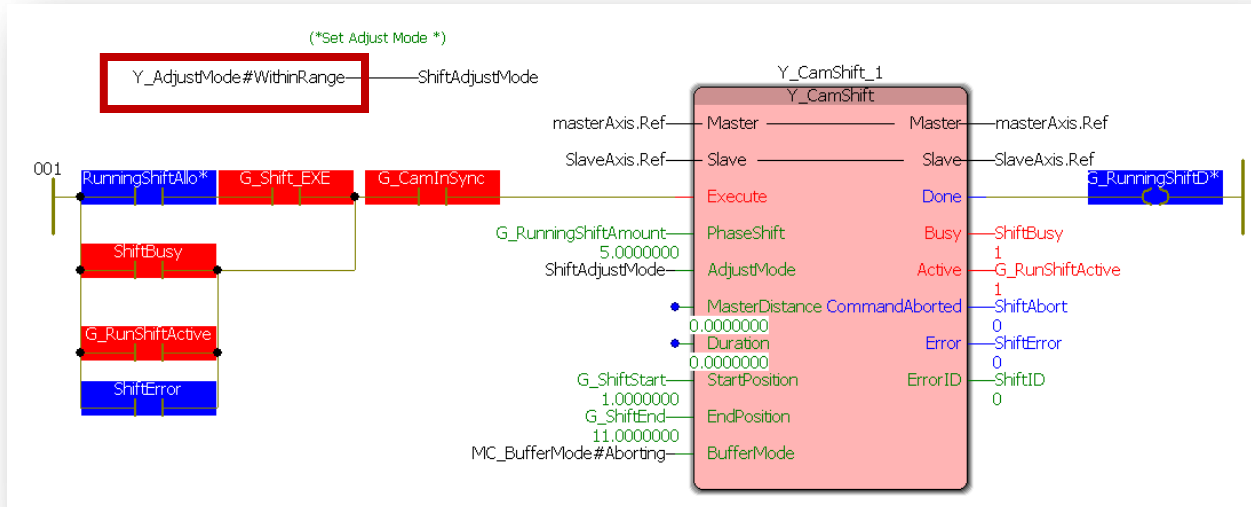
Notice how the expected speed is constant during the cut, but slows down (Fwd Shift) or speeds up (Rev Shift) outside the sync zone.

The diagram below corresponds to the same example, but shows position instead of speed. The dotted line shows the “normal” un-shifted path that the slave would have followed. The solid line shows the path of the slave as it follows the master during a shift.

Notice that the slope of the line returns to the original slope after the shift is complete. This corresponds to the speed matching sync zone.



Y_CamShift “WithinRange” adjust mode is designed for this type of situation. The slave has a Sync Zone and a Shift Zone per slave cycle, and this corresponds to a master sync zone and shift zone per master cycle (in master units). Y_CamShift will shift the given amount during the defined shift zone. The StartPosition and EndPosition inputs refer to the CamMasterShiftedCyclic(1502) position.

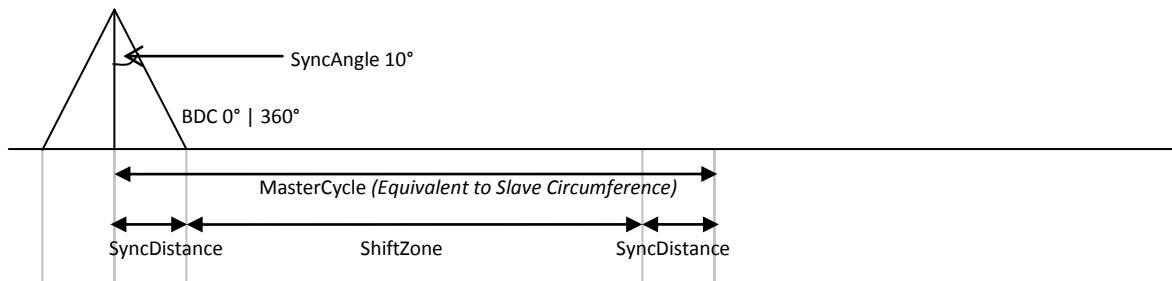


The start and end position of the master could be calculated in different ways. A simple method is to use a relative proportion of the slave sync angle. For example, if the slave sync angle is mechanically ± 10 degrees out of a full rotation of 360 degrees, and the full master cycle is 12.566 [in], then the same proportion is

- Sync Distance = $10/360 * 12.566 = \pm 0.349$ [in].

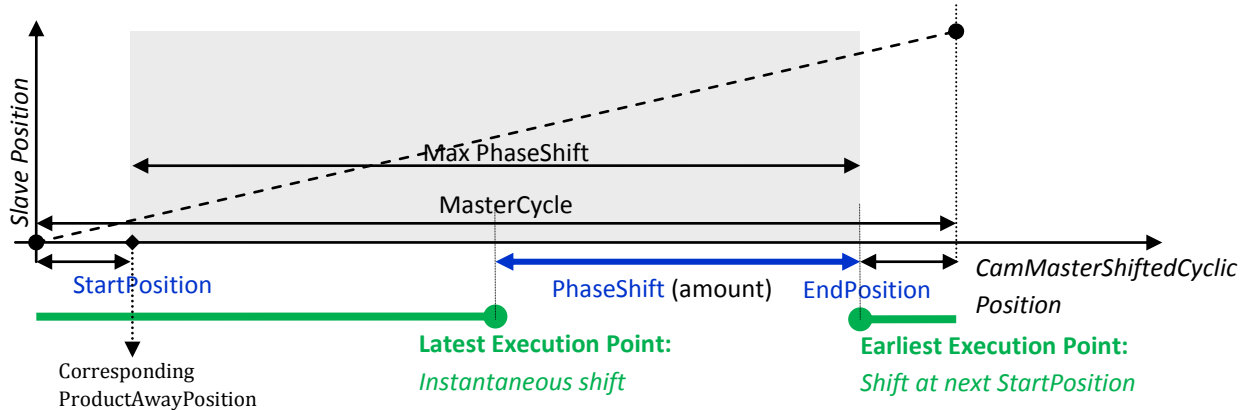
TRAINING ACTION

- Is the Shift Zone at the ends of the master cycle or in the middle? Refer to the diagram below.
- Based on the above proportional calculation for sync distance, what CamMasterShiftedCyclic(1502) positions mark the start and end of the shift zone? Give values.
- Edit the Init.Mechanical POU and update to calculate the following such that they will update if the slave synchronization angle is changed in the future.
 - G_SyncDistance
 - G_ShiftStart
 - G_ShiftEnd

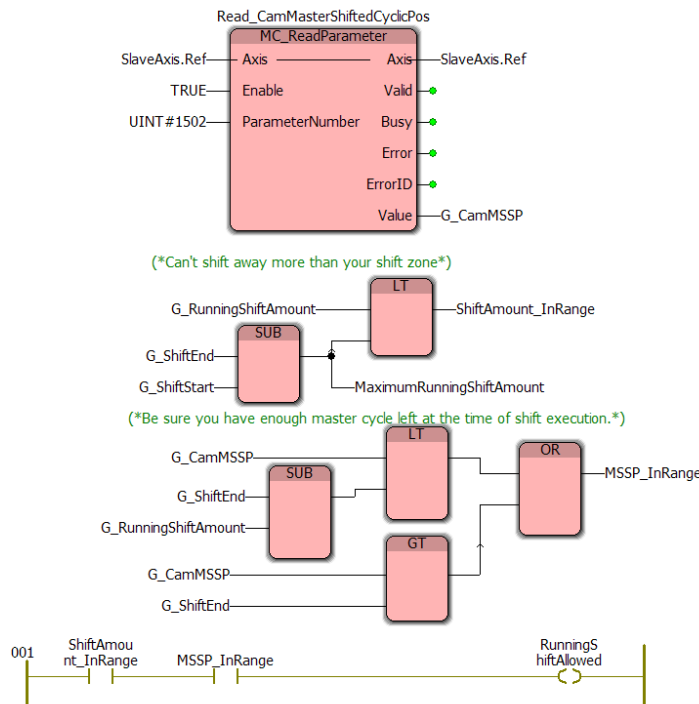


SHIFT EXECUTION TIMING: “WITHIN RANGE”

Y_CamShift set to “WithinRange” performs the shift when CamMasterShiftedCyclic (1502) is between the StartPosition and EndPosition. It is possible, however, to execute the Y_CamShift block when the position is in this range, but it’s too late in the master cycle for the shift to be performed. It is also possible that the PhaseShift amount is simply too high and therefore impossible to be completed within the shift zone. If either of these conditions exists, the block will fault with error 4398.



The starter program has a bit of code that monitors CamMasterShiftedCyclic (1502) and ensures Y_CamShift will execute successfully, preventing the possibility of error 4398 and producing a “RunningShiftAllowed” bit.



At this point in the program, this concept is not critical because the SFC won’t let the running shift execute until the product is away – always corresponding to a CMSSP greater than zero. But at faster speeds the scan time becomes more important, and it may be beneficial to shift at the earliest execution point.

RUNNING SHIFT CALCULATION

After the First Shift has been executed, the cam engages, and the first product is synchronized with the knife, it's time to look at the spacing to the next product and execute the running shift automatically. This can all be calculated and executed immediately while the knife and product are synchronized, because the shift will not begin until the CamMasterShiftedCyclic(1502) position is "WithinRange". It is important to initiate these calculations and executions as soon as possible, because at high speeds, one more controller application scan can translate into a significant distance moved.

Consider the following example to calculate the RunningShift based on the simplified example with 10.000 [in] master cycle "normal" un-shifted cut length

Buffer Element	LatchedMasterPosition	ProductSpacing	RunningShift	Comment
0	154.0	-NA- first cut	-NA- first cut	Use "First Shift" equation
1	163.9	9.9	0.1	Small forward shift
2	171.4	7.5	2.5	Larger forward shift
3	191.9	20.5	-10.5	Part is long, slave slows down
4				

The shift amount is simply how much the product spacing deviates from the default "normal" product spacing. The default product spacing is the master cycle. The master cycle is set equal to knife circumference.

At this point we are going to make one huge assumption – the buffer will always have another product. The next section will deal with disengaging when the buffer is empty.

TRAINING ACTION:

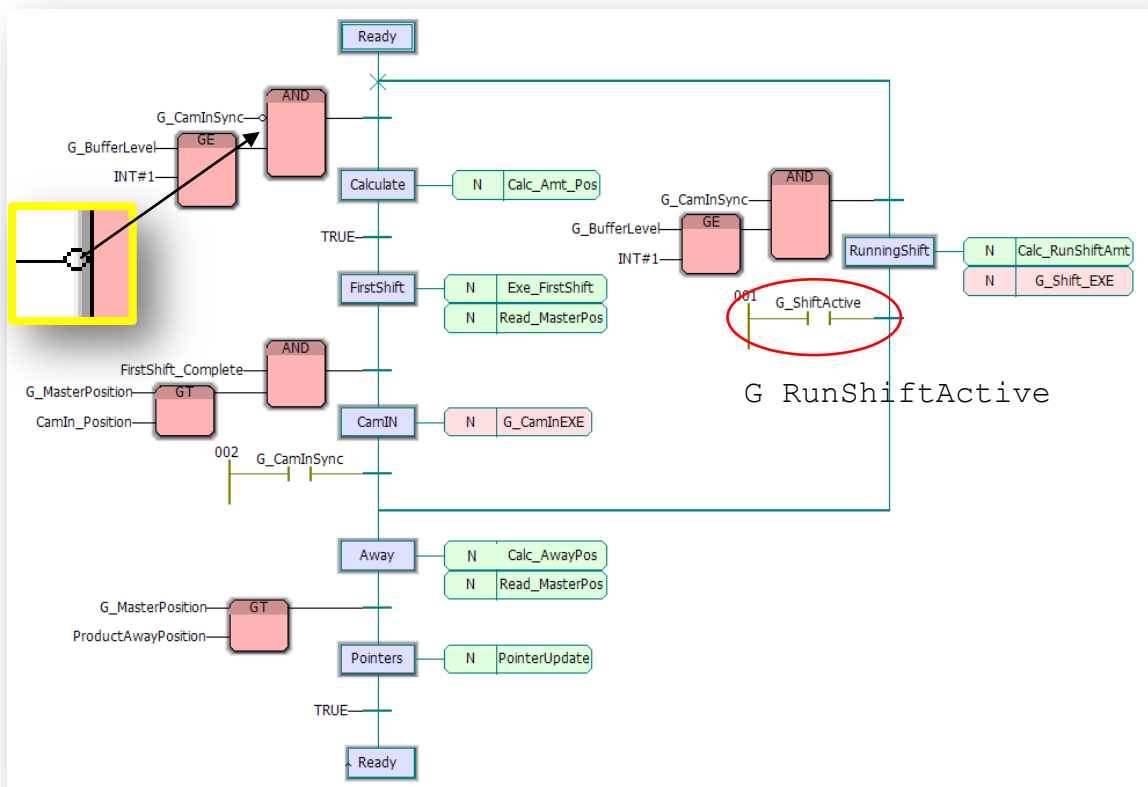
- Express "Product Spacing" as a mathematical equation based on "current latched position" and "previous latched position".
- Express "RunningShift" as a mathematical equation based on Product Spacing and MasterCycle.

PROGRAM THE RUNNINGSHIFT.

Right now the program is a single SFC string that takes us from the top with nothing in the buffer, to the bottom when the product has moved away. Then the SFC jumps back to the beginning. But as it stands now, it will attempt to execute the first shift again. We could keep inserting more steps and attaching some of the same actions, but instead we can make a divergent path.

TRAINING ACTION:

- Remove DI4 from the controls POU
- Update the SFC as illustrated
 - Add a divergent path after READY to a new step called Running Shift
 - Update the transition to FirstShift to be sure the cam is NOT in sync
- Program the Running Shift Calculation in the Calc_RunShiftAmt action in ST
- See the SFC Editing Tips on the next page
- To test the code (warm start as necessary)
 - Enable the buffer and start jogging the master
 - Toggle SI4 at a steady rate
 - Toggle 1x per master revolution, 2x, 3x etc
 - Toggle ever 1.5 revolutions, 2, 3 etc
 - Stop the master and jog slowly to catch the SFC at each step



SFC EDITING TIPS

Create a divergent branch (in Version 2.x)

- Control-click the starting and ending transition
- Click the button “insert SFC branch”,
- Move the new transition to the desired location
- Highlight the new transition
- Click on the button “create new step-transition sequence”



Add Ladder to a transition (in Version 2.x)

- Change transition property to “direct connection”
- Add the contact and power rail in an open white space
- Select the ladder assembly and move to connect to the transition node

LAST SHIFT

At this point the fundamentals are in place and products can successfully go through the process. However, you'll notice that if the product spacing is very great, then the negative shift is very great. A large negative shift causes the knife to move backwards. While a small amount of reverse motion in the knife may be acceptable in some applications, most often it is not.

DEAD MAN CONDITION

So there comes a certain decision point at which it is simply better to disengage the cam and start the sequence over once a new product is latched. This decision point is affectionately referred to as the "dead man position"

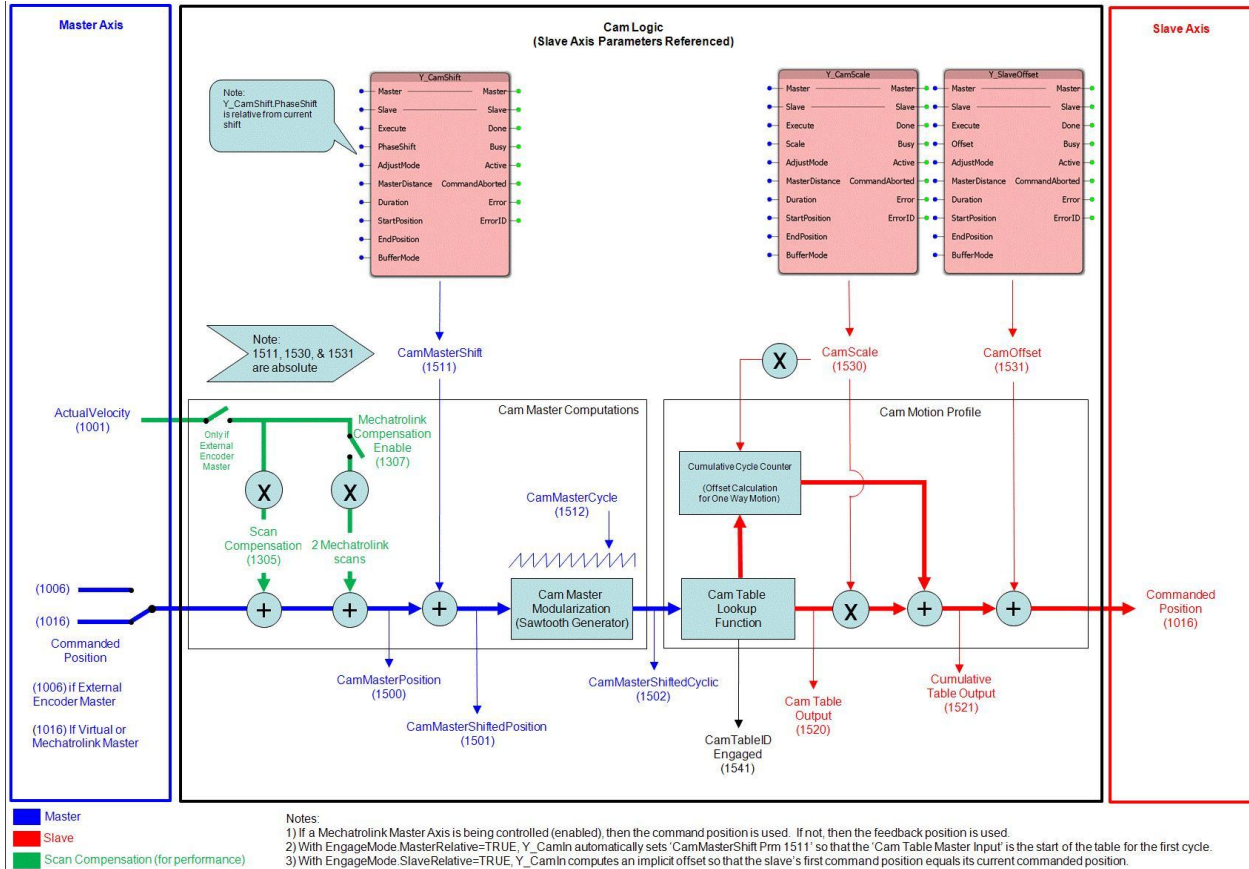
It comes down to two distinct scenarios at which it is better to cam out and start over.

1. Scenario 1: You've started to cut the last product in the buffer; there are no new products yet.
2. Scenario 2: The distance between the latched positions is too large and will cause knife reversal.

Scenario 2 can be avoided by strategically placing the latch sensor close enough to the cut point (but not too close!). As discussed earlier, the sensor "sweet spot" placement is 2.5 master cycles away. This application therefore leaves us with Scenario 1 only.

DISENGAGE AND “UNSHIFT” AFTER DEAD MAN CONDITION

If the Dead Man condition is true, then it's time to CamOut. But there is one clean-up action to take care of before starting over. This is to “undo” the current shift amount that has accumulated. Fortunately, the cumulated shift is stored in the controller as parameter 1511. Remember the Camming Block Diagram.



CHOOSE THE DEAD MAN DISTANCE

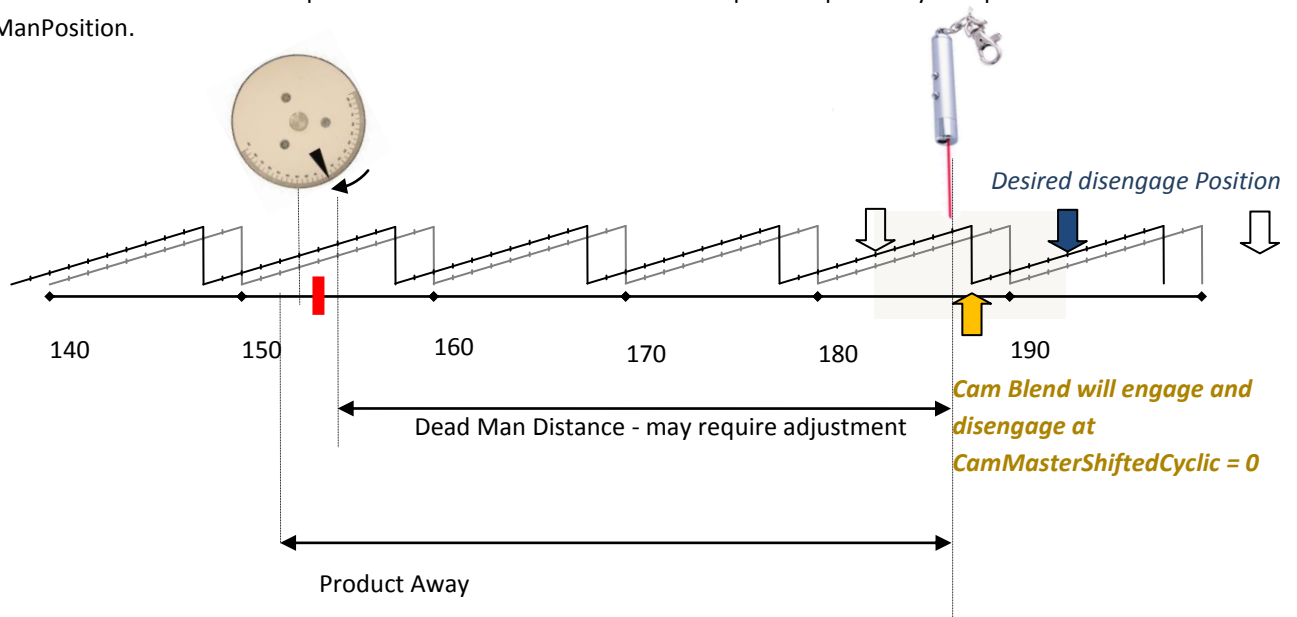
The dead man distance is a “point of no return”. You have to decide to stop the knife even though it is possible that another product is just about to be latched and you could have kept the knife running. But it’s ok if this happens; if the latch sensor is far enough away, then you have time to CamOut and CamIn again right away.

Y_CamOut stops the knife. Y_CamOut disengages at the predefined position (1/2 master cycle). So the block must be executed some distance before the disengage position. Later we’ll use CamBlend to “ramp out”, and this method requires one complete cycle (disengaging at master position zero rather than ½ master cycle). So to accommodate this, execute Y_CamOut before the beginning of the master cycle during which the slave will disengage. How much before depends on factors such as the task interval setting and the speed of the master. A higher task interval and higher master speed means that more distance is covered each application scan, and so if you wait too long you’ll miss it!

This dead man distance can be chosen relative to the CamMasterShiftedCyclic (1502) position, or it can be more directly measured as a raw distance from sensor to point of contact with the knife, or some other arbitrary distance from the sensor. The latter is easier to illustrate, and is the approach used here.

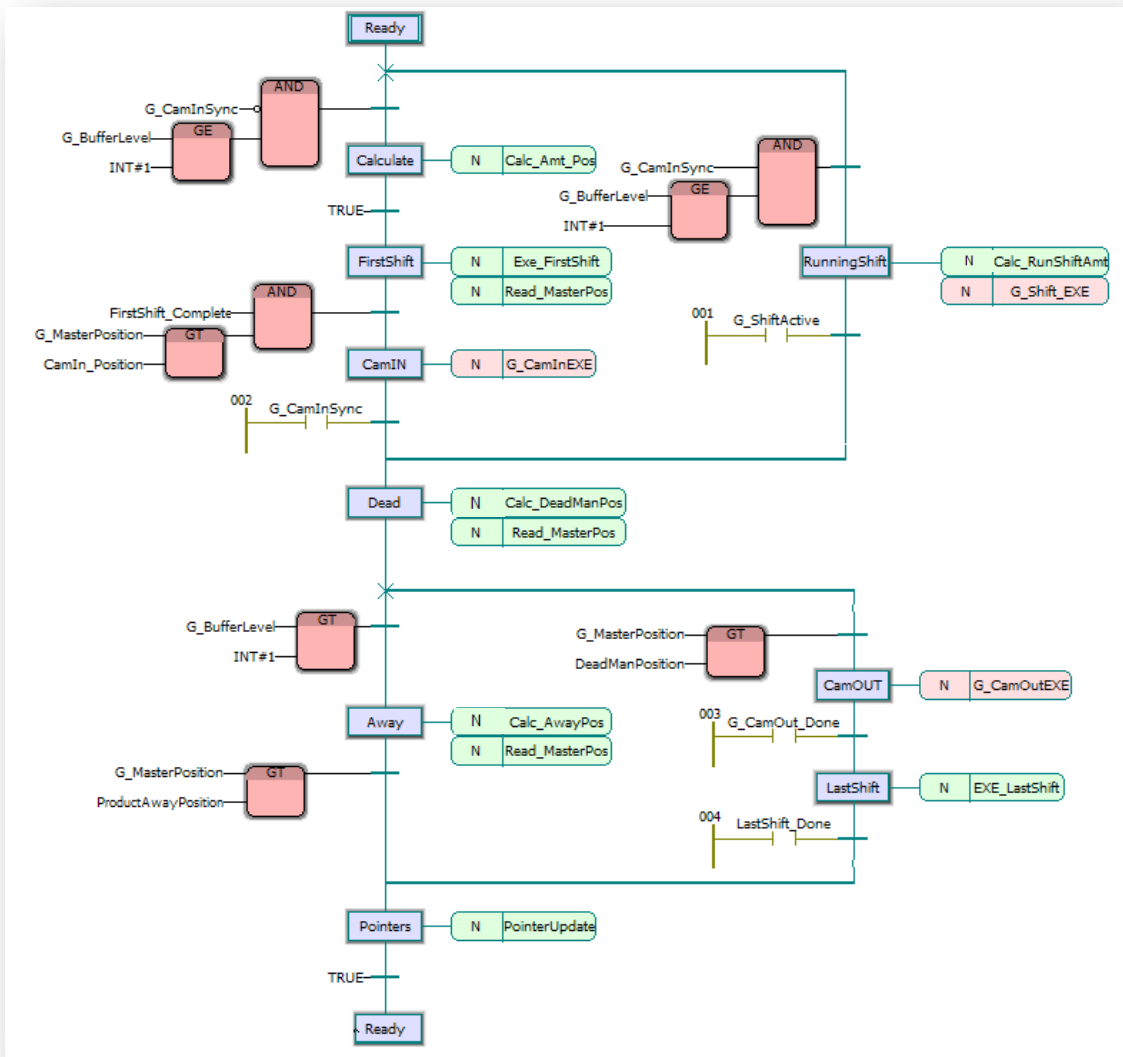
Integration of a specific DeadManPosition for a given position capture in the buffer is very similar to that of ProductAwayPosition. You calculate the future position and just wait for it to pass by.

Each product will first pass the Dead Man position, and then the Product Away position. But the product is only “dead” when there is not another product in the buffer when the current product passes by it’s specific DeadManPosition.



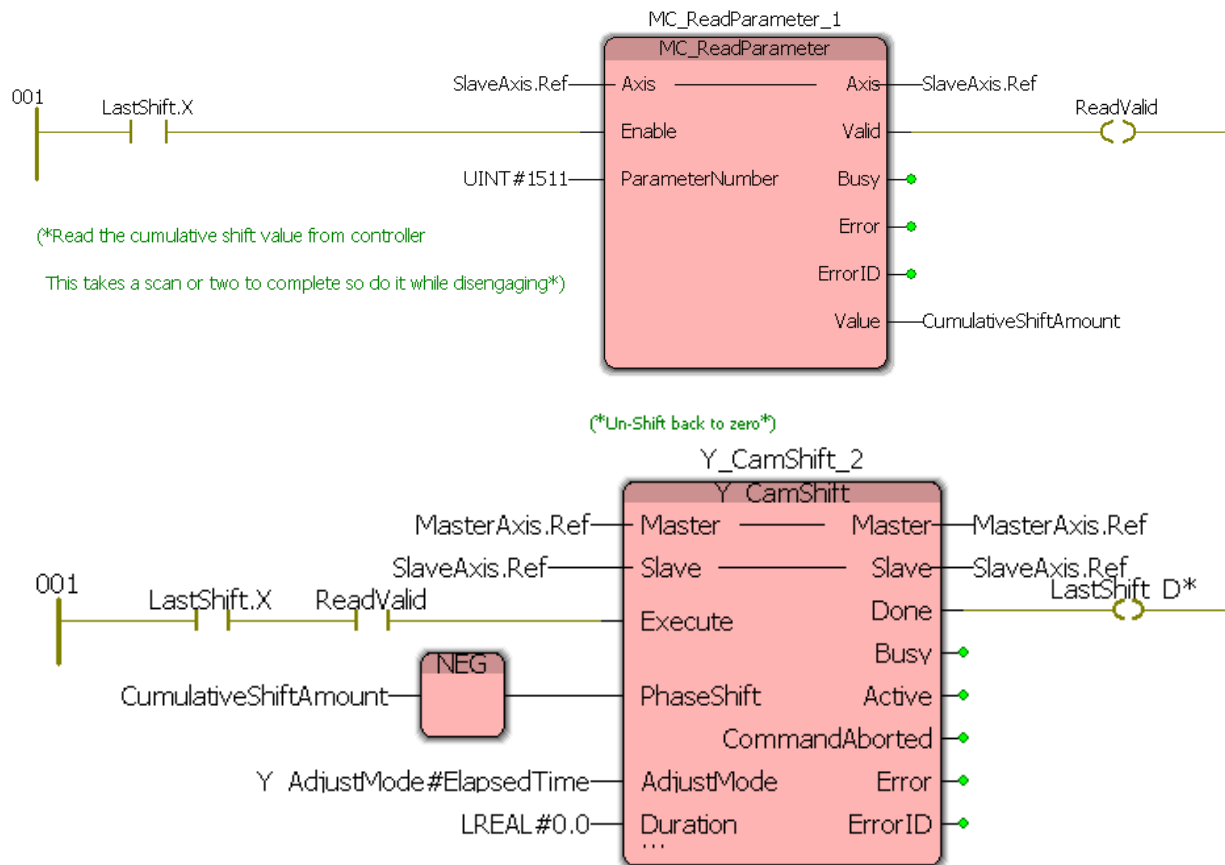
TRAINING ACTION:

- Create and initialize a new variable G_DeadManDistance
- Create the SFC as shown below
 - Tips on next page
 - New Actions
 - Calc_DeadManPos
 - EXE_LastShift
- Use MC_ReadParameter to find the current shift amount in the EXE_LastShift action
- When it looks like it's working, transfer the project to the Rotary Knife Demo for final testing.



EXE_LASTSHIFT

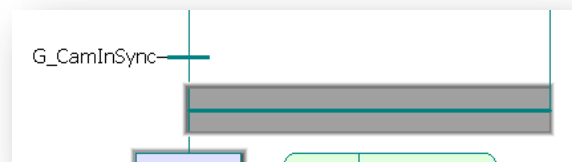
After the cam is disengaged, the cumulative shift amount must be cleared to zero by “un-shifting”.



SFC EDITING TIPS

Insert a Step after an alternate convergence (in Version 2.x)

1. Click on the convergence line
2. Click “Create Step Transition Sequence”



Create a divergent branch (in Version 2.x)

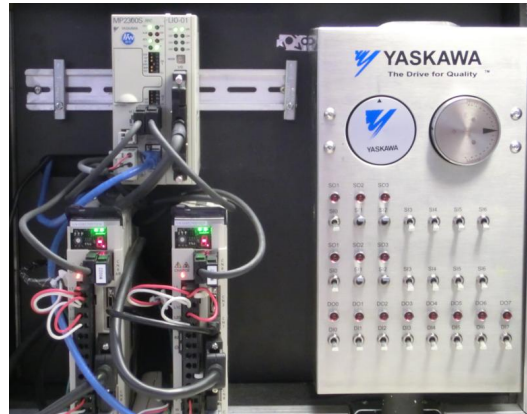
1. Control-click the starting and ending transition
2. Click the button “insert SFC branch”



PHASE 1 CONCLUSION

TEST THE COMPLETED PROGRAM ON THE DESKTOP DEMO UNDER SEVERAL CONDITIONS

- One latch only – does it cut just once?
- Fast latches / short parts? Slow latches / long parts?



TRANSFER PROJECT TO THE ROTARY KNIFE DEMO

- Rename the Project
- Update IP address (project and PC) for the Rotary Knife Demo controller
- Go Online with Hardware Configuration, import configuration from controller, Online Save, Reboot
- Rebuild Project, Stop, Download, Coldstart (cold start resets all retain variables to initial values)
- Verify home offset is -89.0
- Home, Jog, Enable buffer ... and hope it works!



HIGH-SPEED TEST

Increase the jog speed gradually to 80 ipm and even 100 ipm. Use the high-speed camera to see if the cuts are accurate. Eventually the jog speed will be too high and something will stop working.

- We use the Casio EXILIM EX-ZR100 point-and-shoot, up to 1000 FPS (96x240 resolution), \$300 USD <http://www.casio-intl.com/asia-mea/en/dc/lineup/#a0>



PREVIEW REFINEMENTS FOR THE NEXT PHASES OF DEVELOPMENT

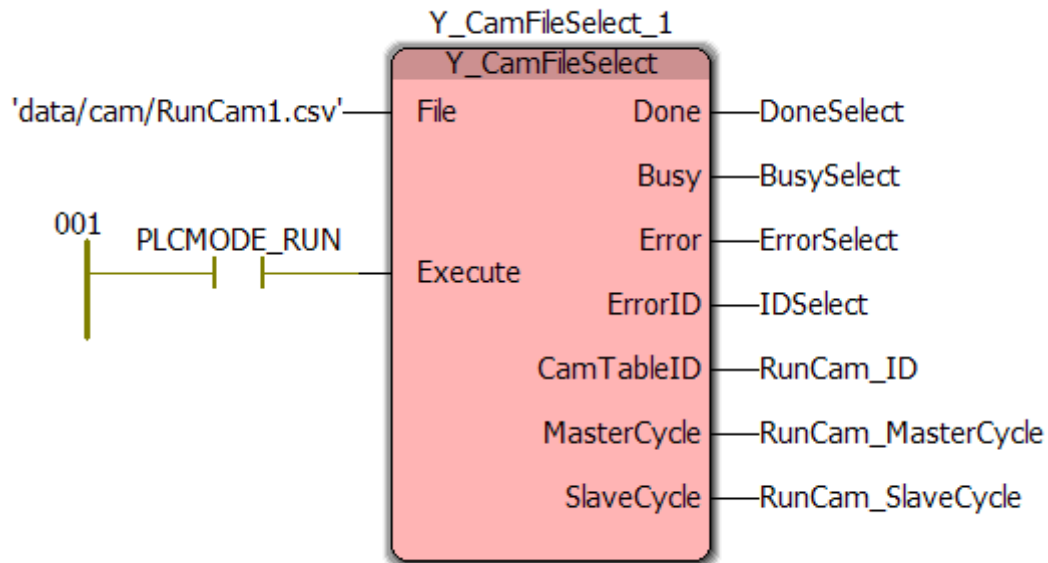
1. Internal Cam table generation
 - a. For a simple linear cam like this one it may not seem like it's worth it to generate the cam internally. But once you see how to do it, it really increases the flexibility. For example, if you think about it, the cam profile technically shouldn't be a simple 1:1 linear cam during the cut. The line speed only matches the knife speed at BDC. Out of this range there is a sinusoidal component to the speed to compensate for the angle of the knife. If the cam table is generated internally, you have full access to an infinite number of custom cam profiles.
2. Cam Blend
 - a. A limitation with the current solution is that the knife quite abruptly starts and stops. While this is not a big deal for a small inertia disc on the training demo, on a real machine this can really be a large mechanical jolt, leading to increased downtime and power consumption. So why not start and stop nice and smooth? Cam blend allows you to do just that by automatically swapping 3 different cam tables (which can be generated internally); one for engaging (RampIn), one while you're running (Running), and another to disengage (RampOut).
3. Adjustments
 - a. You also may notice (using the high-speed camera) that the cuts begin to lag as the machine speed increases. You can adjust the tuning and scan compensation parameters so that this doesn't happen anymore.
 - b. We have not even mentioned what to do if the machine experiences an estop condition. How could you get back in sync without wasting product? The CamBlend function block has an input to make this easy.

PHASE 2: INTERNAL CAM TABLE GENERATION

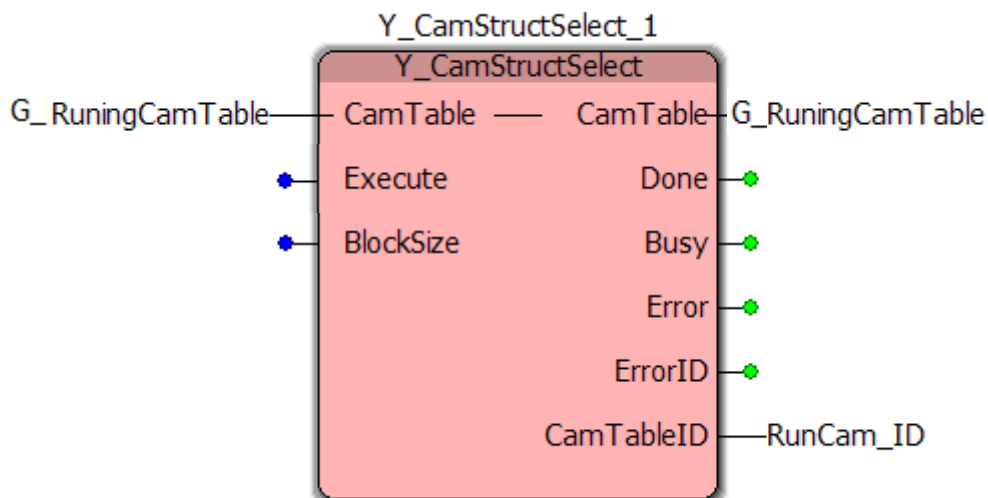
The next step in the refinement of this working application is to generate the cam table internally instead of using a CSV file generated by Yaskawa's CamTool software.

Y_CAMSTRUCTSELECT

You can see in the starter program "Cam In OUT" POU that a static cam table is saved in the controller as a CSV file named "RunCam1.csv", and is given an ID number by Y_CamFileSelect.



The cam data can just as easily be stored in a variable and accessed using Y_CamStructSelect. The variable must have the datatype Y_MS_CAM_STRUCT. The variable will be loaded with master and slave positions using the CamGenerator function block.



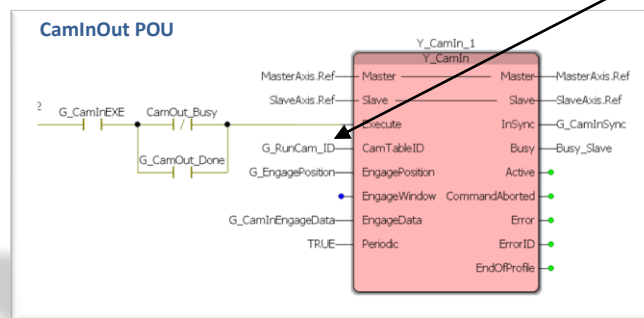
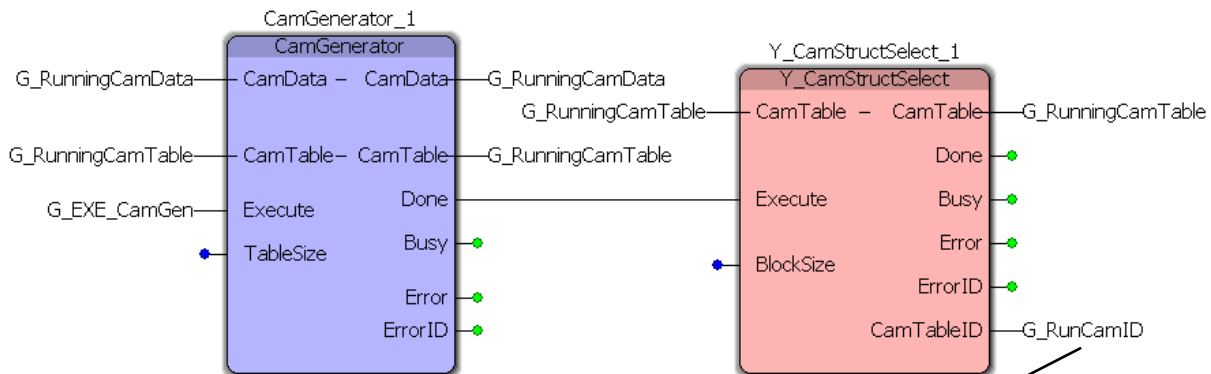
TRAINING ACTION:

- Create a copy of the CamInOut POU (for reference, do not use)
- Delete the Y_CamFileSelect block from the original CamInOut POU
- Create a new LD program POU called “CamCalc” in Student POU folder
- Create a DEFAULT task named “Bckgnd” and insert a program instance of CamCalc
 - The Default task type may be used because the calculation may take a relatively long time and does not have to be completed repeatedly
- Add CamGenerator and Y_CamStructSelect block to the CamCalc POU as illustrated below
- Make , download, warmstart
- Pull up a fresh watch window tab and add G_RunningCamData and G_RunningCamTable to the watch.

CAMGENERATOR

CamGenerator uses the CamData structure to calculate a complete cam table and then loads the table into the CamTable structure. CamData is the input, and CamTable is the output. These variables are implemented in the function block as Var_IN_OUT for efficient memory usage. The CamData input has the datatype “CamSegmentStruct”. This datatype is only available when the project includes Yaskawa’s CamToolbox user library. The CamTable variable has the datatype “Y_MS_CAM_STRUCT”, which is part of PLCopenPlus.

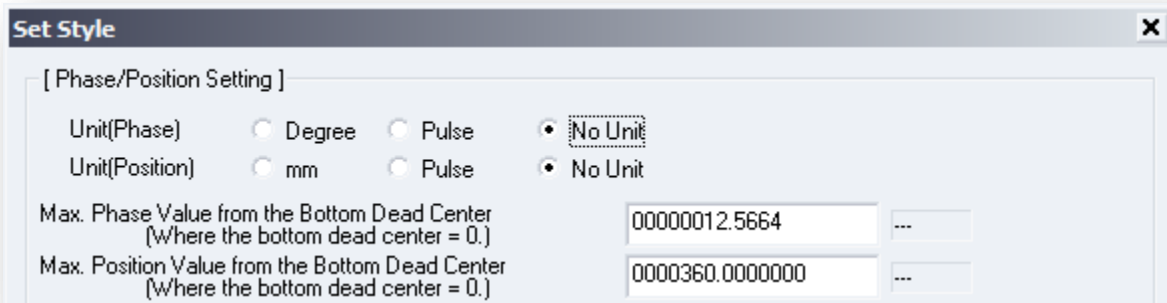
As illustrated below, it makes sense as part of a logical sequence to use the Done output from CamGenerator to execute Y_CamStructSelect. CamTableID is the ultimate output, used by Y_CamIn in the CamInOut POU.



CAM TOOL (SOFTWARE) AND CAMGENERATOR (FUNCTION BLOCK)

Cam generator functions virtually identically to Yaskawa’s Cam Tool software. So if you’ve used CamTool before, it is to your advantage. In this model, the cam profile is broken into sections and intermediate points are calculated based on a mathematical curve shape.

Cam Tool was used to create the csv file for the starter program. It was set for the master to go up to 12.5664 (one master cycle in inches) and the slave would go up to 360 degrees. This is what the software screen, “Set Style” looks like.

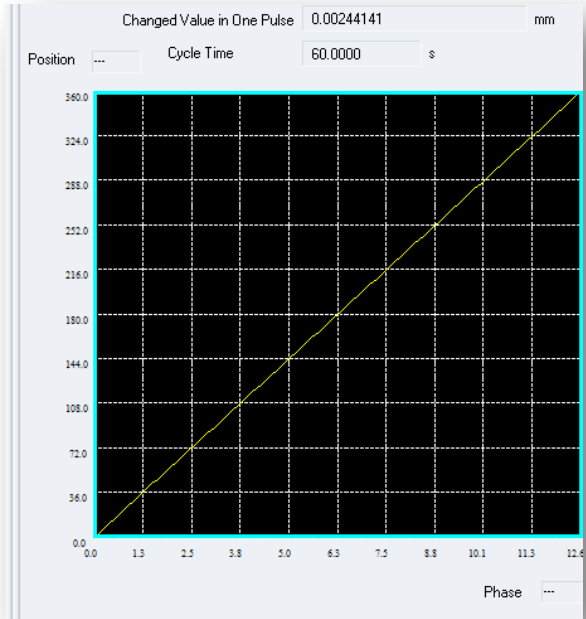


In Cam tool we defined just one linear section to the cam profile. This is the linear “1:1” profile that we have been shifting and engaging. Cam Tool uses a table format. The user enters values for Master End, Follower End, Curve Shape, and Master Plotting in this “Set Parameter” screen.

Num	Master Start	Master End	Follower Start Point	Follower End Point	Curve Shape	Master Plotting	Data Edition
1	00000000.0000	00000012.5664	0000000.0000000	0000360.0000000	Straight line	0000000.1000000	Not Provic
2							
3							

From these parameters, Cam Tool generates this linear cam profile, again with just 1 section.

Here are some of the intermediate data points in the cam table that cam tool generated. These two columns of master position and slave position were sent to the controller as a csv file.



Num	Phase	Position
107	00000010.6000	0000302.8571429
108	00000010.7000	0000305.7142857
109	00000010.8000	0000308.5714286
110	00000010.9000	0000311.4285714
111	00000011.0000	0000314.2857143
112	00000011.1000	0000317.1428571
113	00000011.2000	0000320.0000000
114	00000011.3000	0000322.8571429
115	00000011.4000	0000325.7142857
116	00000011.5000	0000328.5714286
117	00000011.6000	0000331.4285714
118	00000011.7000	0000334.2857143
119	00000011.8000	0000337.1428571
120	00000011.9000	0000340.0000000
121	00000012.0000	0000342.8571429
122	00000012.1000	0000345.7142857
123	00000012.2000	0000348.5714286
124	00000012.3000	0000351.4285714
125	00000012.4000	0000354.2857143
126	00000012.5000	0000357.1428571
127	00000012.5664	0000360.0000000

The CamData input of CamGenerator contains the same input parameters from the “set style” window and “set parameter” window of Cam Tool, and will generate the same data as Cam Tool. But CamGenerator saves the data to the CamData output structure instead of a csv. And of course, it does it all internally to the controller.

ELEMENTS OF THE CAMDATA STRUCTURE

Now let's compare the data inside the CamData structure to the data in Cam Tool software.

Variable	Value	Type	Instance
RunningCamData		CamSegment...	Configurati
+ CamParameters		CamSegment...	Configurati
SlaveStart	0.0000000	LREAL	Configurati
LastSegment	1	INT	Configurati

SlaveStart –The first entry of Follower Start Point, which in CamTool automatically defaults to 0.00 and cannot be changed.

LastSegment – In Cam Tool the “Num” column numbers each segment in the cam profile. CamGenerator requires that you specify the last segment manually. CamTool recognizes the last segment automatically.

Num	Master Start	Master End	Follower Start Point	Follower End Point	Curve Shape	Master Plotting	Data Edition
1	00000000.0000	00000012.5664	00000000.0000000	0000360.0000000	Straight line	0000000.1000000	Not Provic
2							
3							

Expand the CamParameters and you see it contains an array of structures.

Each **array element** corresponds to a row in Cam Tool – the “Num” column.

Each **element of the structure** corresponds to a column in the Cam Tool “Set Parameter” screen.

MasterEnd – “Master End” column in Cam Tool

SlaveEnd – “Follower End Point” column in Cam Tool

CurveType – “Curve Shape” column in Cam Tool. The number corresponding to each curve type is defined as an enumerated type in the CamToolbox “CamTypes” datatypes definition. (Blue “Libraries” tab). Good graphical explanations are listed in the Toolbox Help file under “Cam Toolbox - Creating Cam Tables”

Variable	Value	Type
CamParameters		Ca
[0]		Ca
MasterEnd	0.0000000	LF
SlaveEnd	0.0000000	LF
CurveType	0	IN
Resolution	0.0000000	RE
[1]		Ca
MasterEnd	12.5663706	LF
SlaveEnd	360.0000000	LF
CurveType	1	IN
Resolution	0.1000000	RE
+ [2]		Ca

Resolution – “Master Plotting” in Cam Tool

You’ll notice that there is an array element [0] but Cam Tool starts with Num =1. Array element [0] is reserved for use by CamGenerator and the data in it must be left at 0.00.

NOTE: Different versions of Cam Tool have different text for these columns. Cam Tool version 4.61 was used in this example.

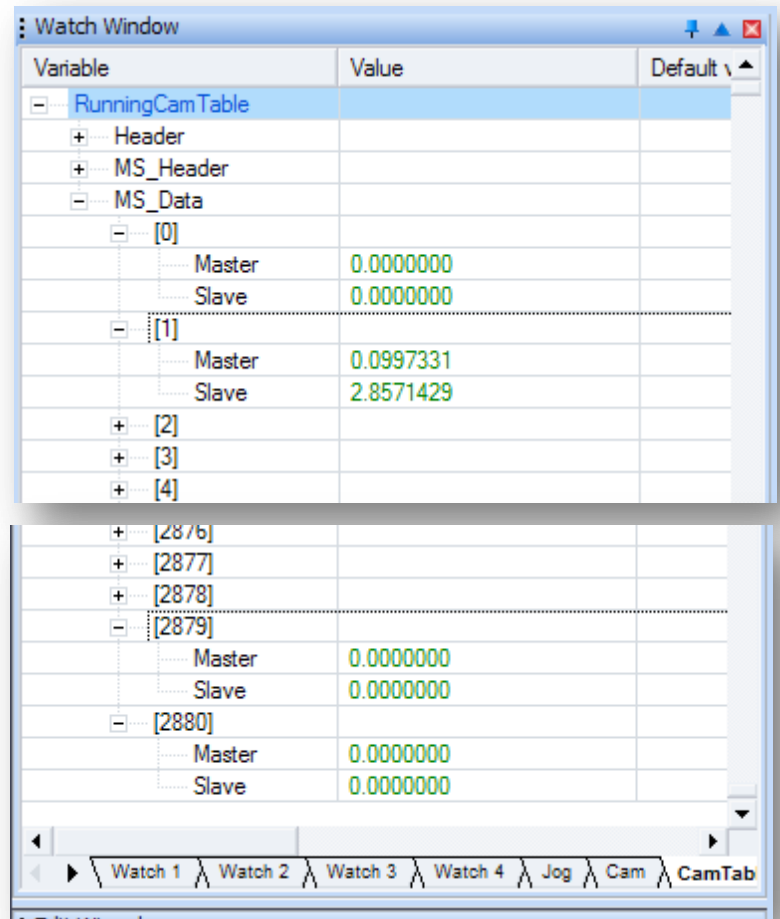
TableSize

The final input of note is TableSize. This input can usually be left to the default of 2880.

TableSize is the maximum number of master-slave pairs that could possibly be generated by CamGenerator.

If you expand the RunningCamTable in the watch page and you’ll see that it has memory for up to 2880 master-slave positions.

In the case that your table requires more than 2880 positions, you would increase the number on this TableSize input, and also make a change to the PLCopen Toolbox user library where this MD_Data array is defined. See the help for more information.



PROGRAMMING: INTERNAL CAM TABLE GENERATION

TRAINING ACTION:

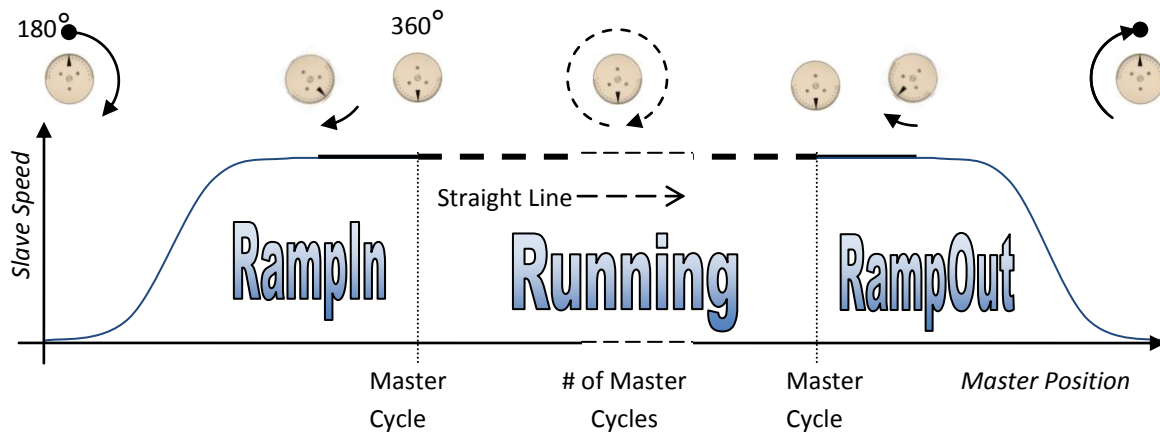
- Create a new worksheet named “CamGen” in the Initialize POU to load values into the G_RunningCamData variable
 - Avoid hard-coding any values to the initialization. Build your cam table from existing variables. For example, instead of 12.5664, use the G_MasterCycle variable.
- Integrate CamGenerator and Y_CamStructSelect
- Run the code. Operation will be identical, but now you have the possibility of creating a slightly different cam table without loading a separate CSV
 - Be sure that CamGenerator and CamStructSelect run without errors
 - Look at the data in the watch window to be sure it was loaded

PHASE 3 – CAM BLEND

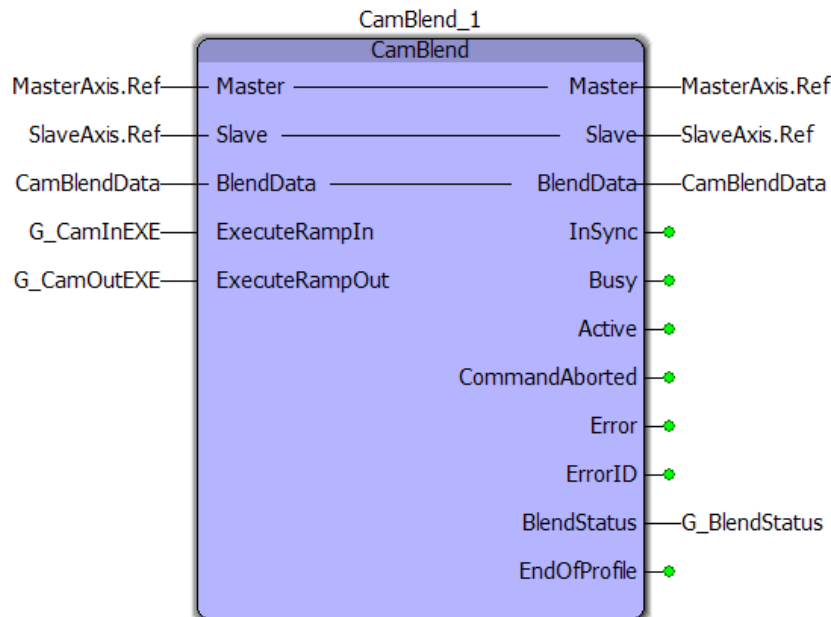
CamBlend solves the following problem: the slave instantly “jerks” into motion when the Y_CamIn block engages and disengages the cam. This is because the master is already moving. The 1:1 linear cam profile has no position at which the slave is not moving relative to the master. What if we could use a different cam profile to engage and disengage the cam smoothly? This is the function of CamBlend.

CamBlend is designed to use 3 different cam profiles

- RampIn
- Running
- RampOut



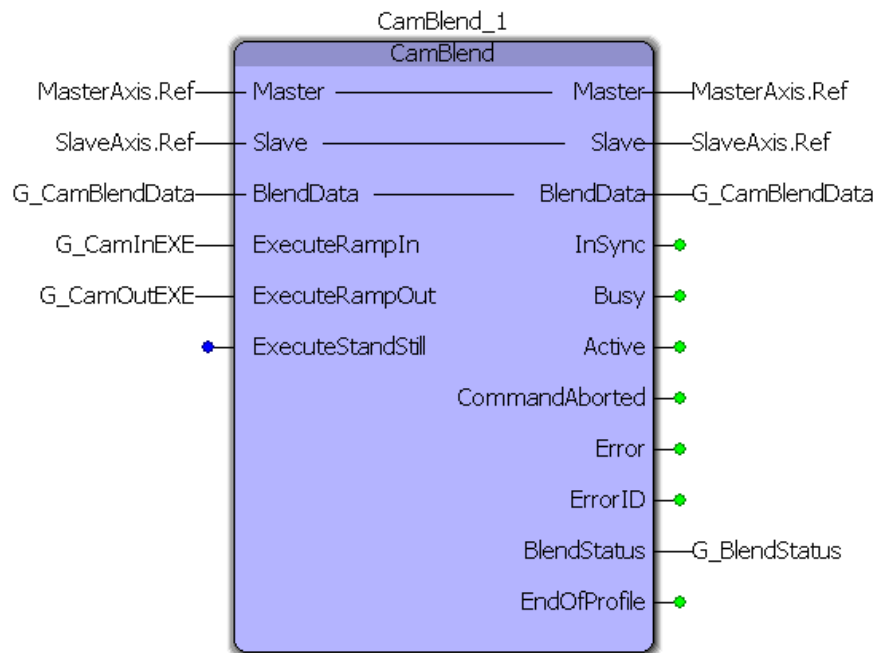
CamBlend REPLACES the functionality of Y_CamIn and Y_CamOut. Instead of executing Y_CamIn and Y_CamOut, execute the RampIn and RampOut inputs of CamBlend.



Let's first set up the blocks to calculate and load these profiles. Once that is in place, we can focus on the cam table calculations for RampIn and RampOut.

TRAINING ACTION

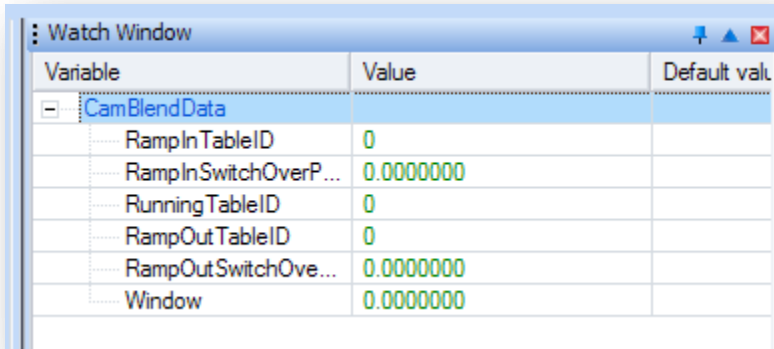
- Create a new LD program POU named "BlendInOut" and run it in the fast task
- Remove CamInOut from the fast task
- Add the CamBlend block to the POU, and attach the master and slave axes
- What variables were originally used to execute CamIn and CamOut? Attach these to the appropriate inputs of CamBlend
- Attach a global variable named G_CamBlendData to the BlendData input. Be sure the data type is BlendStruct
- Download changes, then add G_CamBlendData to a new watch tab.



Notice from the help that InSync output of CamBlend turns on only when the running cam profile is active. Therefore it should not be used in the SFC transition for CamIN. More details are explained later.

BLENDDATA STRUCTURE

BlendData is another example of a variable used as VAR_IN_OUT simply for efficient memory use. In practice it is just a grouping of inputs to the block. You can see the elements of BlendData in the watch window.



The screenshot shows a 'Watch Window' with a table of variables and their values. The table has three columns: 'Variable', 'Value', and 'Default val'. The variable 'CamBlendData' is expanded to show its sub-variables: 'RampInTableID' (0), 'RampInSwitchOverP...' (0.0000000), 'RunningTableID' (0), 'RampOutTableID' (0), 'RampOutSwitchOve...' (0.0000000), and 'Window' (0.0000000).

Variable	Value	Default val
CamBlendData		
RampInTableID	0	
RampInSwitchOverP...	0.0000000	
RunningTableID	0	
RampOutTableID	0	
RampOutSwitchOve...	0.0000000	
Window	0.0000000	

RampInTableID, RunningTableID, and RampOutTableID – will be loaded with the TableID numbers from either Y_CamFileSelect or Y_CamStructSelect function blocks. This is just like the CamTableID input from Y_CamIn, only there are three tables involved.

RampInSwitchOverPos – The CamMasterShiftedCyclic(1502) position at which the block will switch from camming to the RampIn table, and start camming to the Running table. These two tables must be created to have an “overlap” portion, such as a straight-line portion. The master and slave positions near the switch over position should be identical in all 3 cam tables. This is normally at CamMasterShiftedCyclic(1502) position = zero.

RampOutSwitchOverPos - The CamMasterShiftedCyclic(1502) position at which the block will switch from camming to the Running table, and start camming to the RampIn table. These two tables must be created to have an “overlap” portion, such as a straight-line portion. The master and slave positions near the switch over position should be identical in all 3 cam tables. This is normally at CamMasterShiftedCyclic(1502) position = zero.

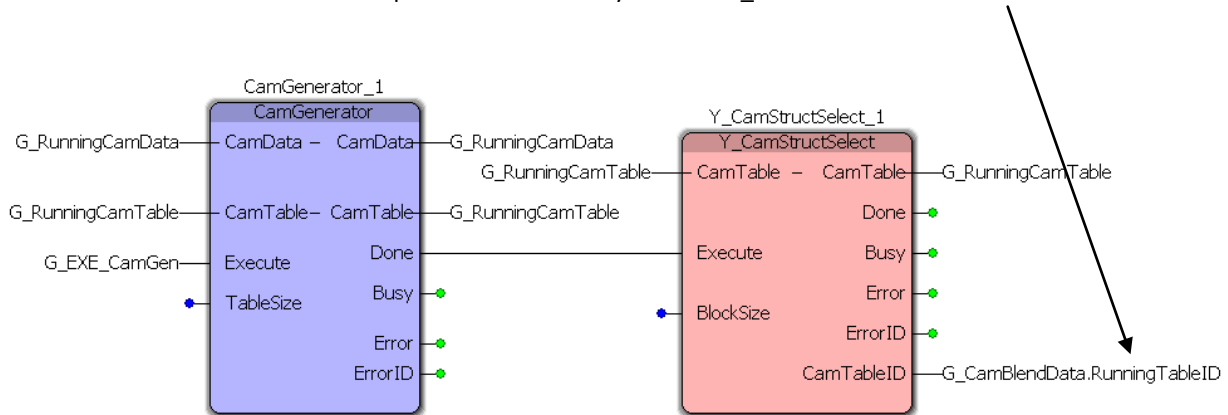
Window - Window sets a zone around the switch over position that is valid. Same as the “EngageWindow” input for Y_CamIn and Y_CamOut. It is set as a percentage of the master cycle. If the task interval is low and the master speed is fast, it is possible that the program POU does not reliably find a CamMasterShiftedCyclic(1502) position within this window. Calculation is possible based on maximum master speed and task interval to find the master distance moved in one scan. If that distance is greater than 1% of the master cycle, then increase the window to an appropriate value. The default of 1% is usually sufficient.

PROGRAMMING: CAMBLEND

Set up the CamCalc POU to load CamTableID numbers into the G_CamBlendData elements.

TRAINING ACTION: GENERATE CAM TABLE ID FOR CAMBLEND

- Add CamGenerator and Y_CamStructSelect blocks to CamCalc POU to calculate RampIn, Running, and RampOut table id numbers.
- The illustration below shows how to accomplish this for the existing RunningTableID. Repeat for RampIn, and RampOut.
 - Use variables named
 - G_RampInCamData
 - G_RampInCamTable
 - G_RampOutCamData
 - G_RampOutCamTable
 - The CamTableID outputs can feed directly into the G_CamBlendData structure.

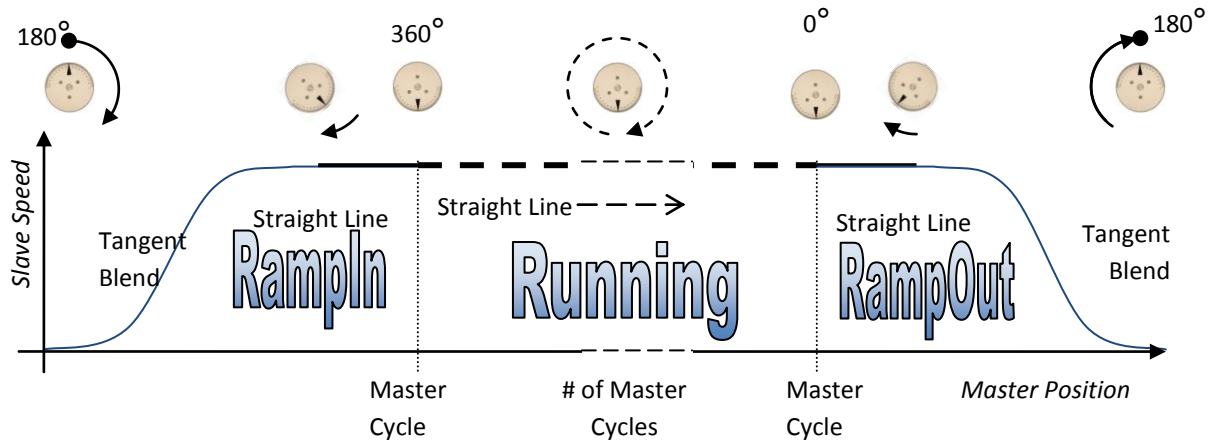


At this point we just have empty CamGenerators feeding into the RampInTableID and RampOutTableID of CamBlend. These cam tables must be generated. A few illustrations on the next page will better explain what's going on with CamBlend, the 3 cam tables, and the switch over positions.

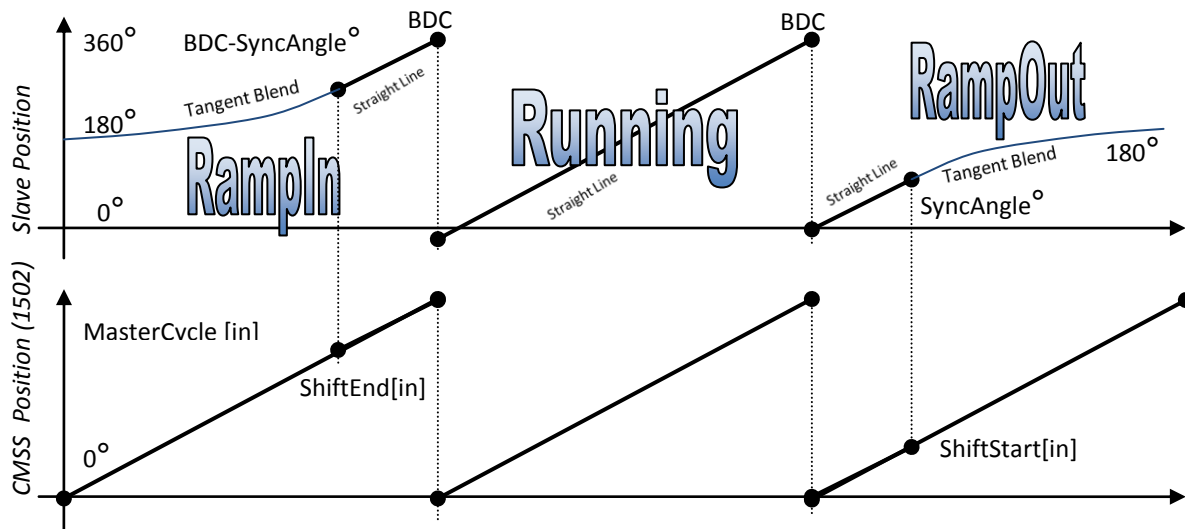
CAMBLEND ILLUSTRATION

The following illustrations will be used to explain CamBlend by means of a few questions.

Slave Speed (top) shows the general speed profile of the knife. This is basically the same as what has already been implemented, but with the blend in and out.



These tables show the cam profiles – slave position and master position. These diagrams will be used as reference to generate the cam tables.



TRAINING ACTION: BLEND QUESTIONS

- Based on the diagram, at what CamMasterShiftedCyclic(1502) position does it “switch over” from RampIn to Running?
- Based on the diagram, at what CamMasterShiftedCyclic(1502) position does it “switch over” from Running to RampOut?

Refer to the previous cam profile illustrations to answer the following questions. The answers contain the information required to initialize G_RampInCamData and G_RampOutCamData. Give the variable or mathematical expression when possible.

QUESTIONS: RAMPIN PROFILE

- What is the slave speed at the beginning of the profile?
- What is the slave position at the beginning of the profile?
- What is the slave position at the end of the profile?
- How many “sections” does the profile have?
- What curve type (curve shape) do the “sections” have?
- At what CamMasterShiftedCyclic(1502) position does one “section” stop and the other “section” start?
- What is the slave (knife) position at the end of the “section”?

QUESTIONS: RAMPOUT PROFILE

- What is the slave speed at the beginning of the profile?
- What is the slave position at the beginning of the profile?
- What is the slave position at the end of the profile?
- How many “sections” does the profile have?
- What curve type (curve shape) do the “sections” have?
- At what CamMasterShiftedCyclic(1502) position does one “section” stop and the other “section” start?
- What is the slave (knife) position at the end of the “section”?

CAMBLEND: GENERAL REQUIREMENTS FOR CAM TABLES

Programming with CamBlend requires you to generate the RampIn, Running, and RampOut profiles.

MASTER CYCLE: All cam tables must have the same master cycle. The master cycle is somewhat arbitrary. Set the cycle equal to a significant distance, such as the circumference of the rotary knife.

SLAVE POSITION: The cam tables will blend smoothly only if the last slave position on one table is the same as the first slave position on the next table in the blend sequence. Also, within the CamData for each table, slave end positions in one section of the cam data must be higher than the end position in the previous section. For example, in a rotary slave axis this affects whether you use 360 or 0. Remember that for a rotary knife, the position 0 degrees = 360 degrees.

SLAVESTART: CamGenerator allows the slave to start at an arbitrary position instead of forcing the cam to use slave position zero for the first position like Cam Tool does.

TANGENTBLENDING CURVE TYPE

A TangentBlending segment may be used in a cam table with ONLY one other StraightLine segment, either before or after, for a total of exactly 2 segments in the CamData.

TangentBlending assumes a zero start speed if it is the first segment, and a zero end speed if it is the second (and last) segment in the CamData. This curve type is designed for use with the CamBlend function block.

PROGRAMMING: CAMBLEND

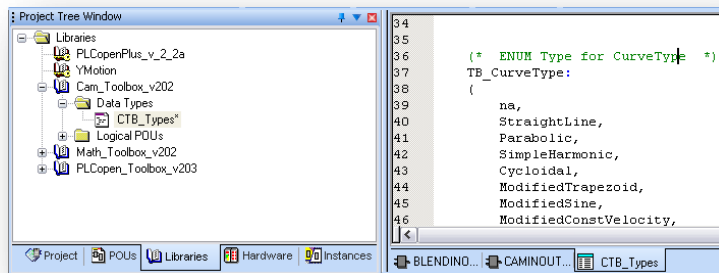
TRAINING ACTION: CALCULATE CAM TABLES FOR CAMBLEND

1. Open the Intialize.CamGen worksheet and initialize the remaining G_RampInCamData and G_RampOutCamData structures
 - Avoid the use of literals in the initialization. Most values you need for the cam data have already been calculated elsewhere in the Initialize POU. For example:
 - G_ShiftStart
 - G_ShiftEnd
 - G_BDC
 - G_MasterCYcle
 - G_SyncAngle
 - Curve Type can be assigned using the Enumerated Type format. For example
 - TB_CurveType#StraightLine
 - TB_CurveType#TangentBlending
 - A complete list can be found in the CamToolbox Data Types (Project Tree - Libraries tab) or in Toolbox Help

!!! ATTENTION !!!!

There is a KNOWN ISSUE with the Blend curve type in Cam_Toolbox_V202, fixed in later versions.

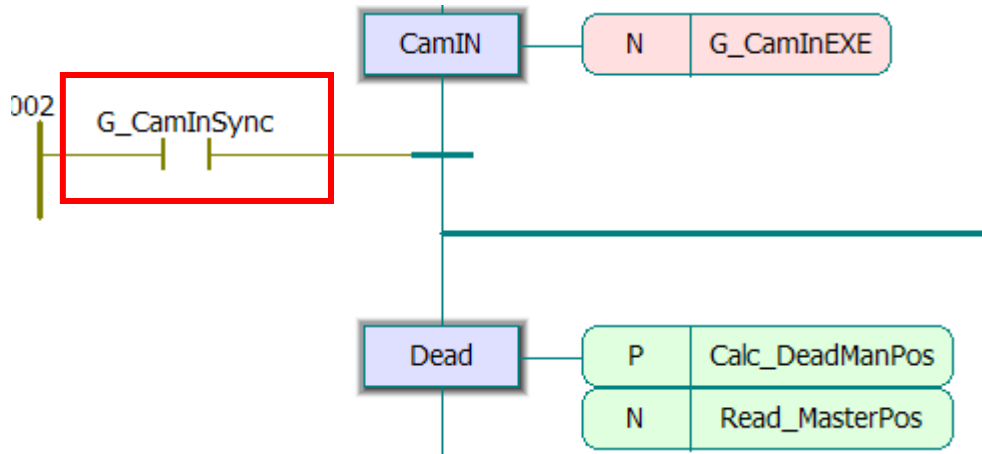
If you have Cam_Toolbox_V202, please remove it and use a later version, if available, or use version V201, available from the archives in Yaskawa.com/iectb.



2. Verify that all three tables calculate with no errors.
3. Force CamBlend to ramp in and ramp out
 - Set a parallel contact to “Toggle Boolean”
 - Forget about buffering, shifting, and SFC for a minute. Does cam blend work?
4. Integrate CamBlend in place of CamIn and Cam Out.
 - Details on the following pages

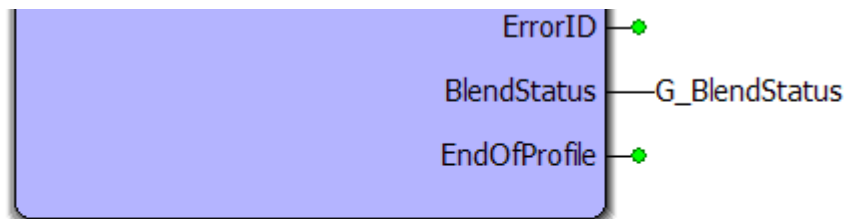
PROGRAMMING: CAM_CONTROL_SFC MODIFICATION

The Cam_Control_SFC program POU uses the variable G_CamInSync in several places. One place is at the transition between the CamIn step and Dead step. This variable comes from the Y_CamIn_1.InSync output in the CamInOut program POU, which will no longer be used.

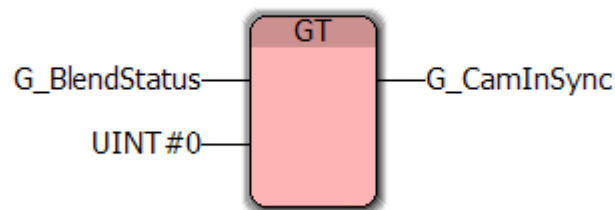


The “equivalent” InSync output of CamBlend is NOT equivalent for the sequential logic. As stated in the help, CamBlend.InSync=TRUE only when the RunningCam profile is in sync. The transition will therefore remain false during the entire RampIn profile when the original intention was to proceed to the Dead step as soon as the cam engages. At slow speeds there is enough time for the controller to scan each transition and handle the next part, but at high speeds it will not.

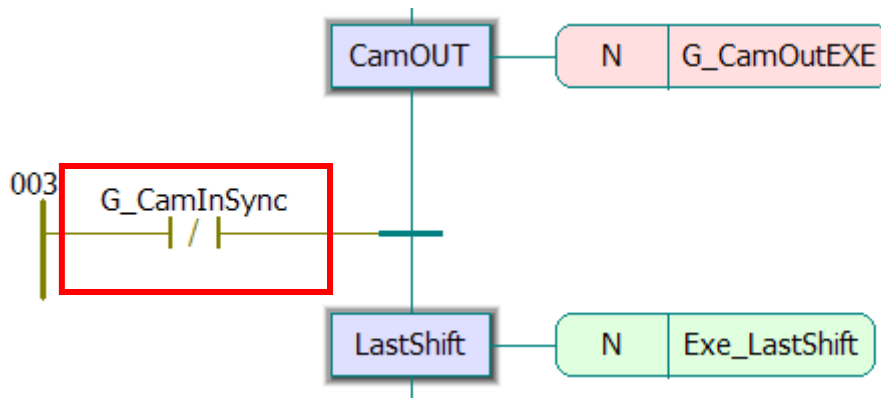
One solution is to instead read the CamBlend.BlendStatus output. As the help indicates, this output is 0 when cam disengaged but 1,2, or 3 to indicates the table in use. This can be used with CamBlend to check for “camming active” or “camming not active” and for ease of use, keep the existing variable name, G_CamInSync.



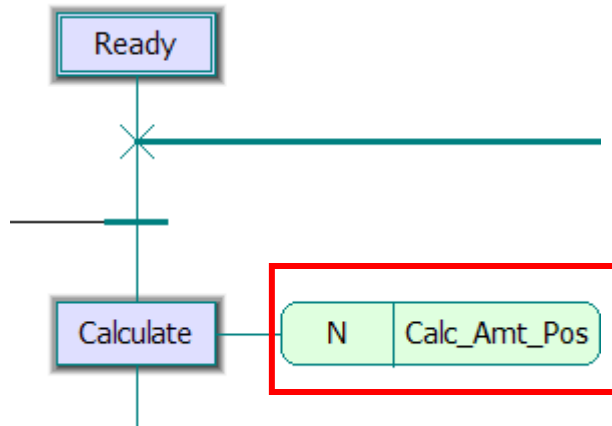
(*InSync" for this application means that camming is active*)



In the same way, the CamOUT transition must be modified to look for Cam not in sync



Another important point is the CamIn_Position must be calculated differently. Before CamBlend was implemented, we engage 1 master cycle ahead of time, since the cam engaged at $\frac{1}{2}$ master cycle. CamBlend works in full master cycles so now we need to increase this to 1.5 master cycles. See Y_CamIn Execution Zone on page 37.



PERFORMANCE ADJUSTMENTS

The following sections are open-ended and completely optional. When your program application has reached this point, please informally consult with the instructor and Yaskawa application engineers to explore any of the topics of interest to you.

MECHATROLINK II SUBINTERPOLATION

Review and implement these concepts. This material is presented in the basic class *MPiec and Sigma-5 Application Programming*. PDF of this document is available.

Pn812 = 2 ms (equal to Mlink scan time) – MP2600iec use Pn217

Param 1311 (sub-interpolation filter) = 2 (enable s-curve accel/decel)

TUNING FOR LOW POSITION ERROR

Review and implement these concepts. This material is presented in the basic class *MPiec and Sigma-5 Application Programming*. PDF of this document is available.

Use mode 1 (standard) Means Model Following Control is off

Run Advanced Auto-Tuning in SigmaWin+

Param 1310 (controller feed forward) = disabled

Pn109 = 90 (Servopack feed forward level)

SCAN COMPENSATION

The demo can be re-wired to use encoder input from the LIO-01 card (from servopack encoder output on CN-1). This simulates the situation where the controller does not have control over the master axis and must rely on an external encoder feedback data from the master instead of the commanded position to the master. I/O scan delays are incorporated, but scan compensation is effective in predicting the future delay and compensating for it.

E-STOP RECOVERY

CamBlend includes an input called ExecuteStandStill. According to the HELP, “Upon the rising edge, this function block will prepare to engage the slave to the Running cam profile at the StandStillEngage position (calculated after an E-Stop recovery routine) in the BlendData structure”.