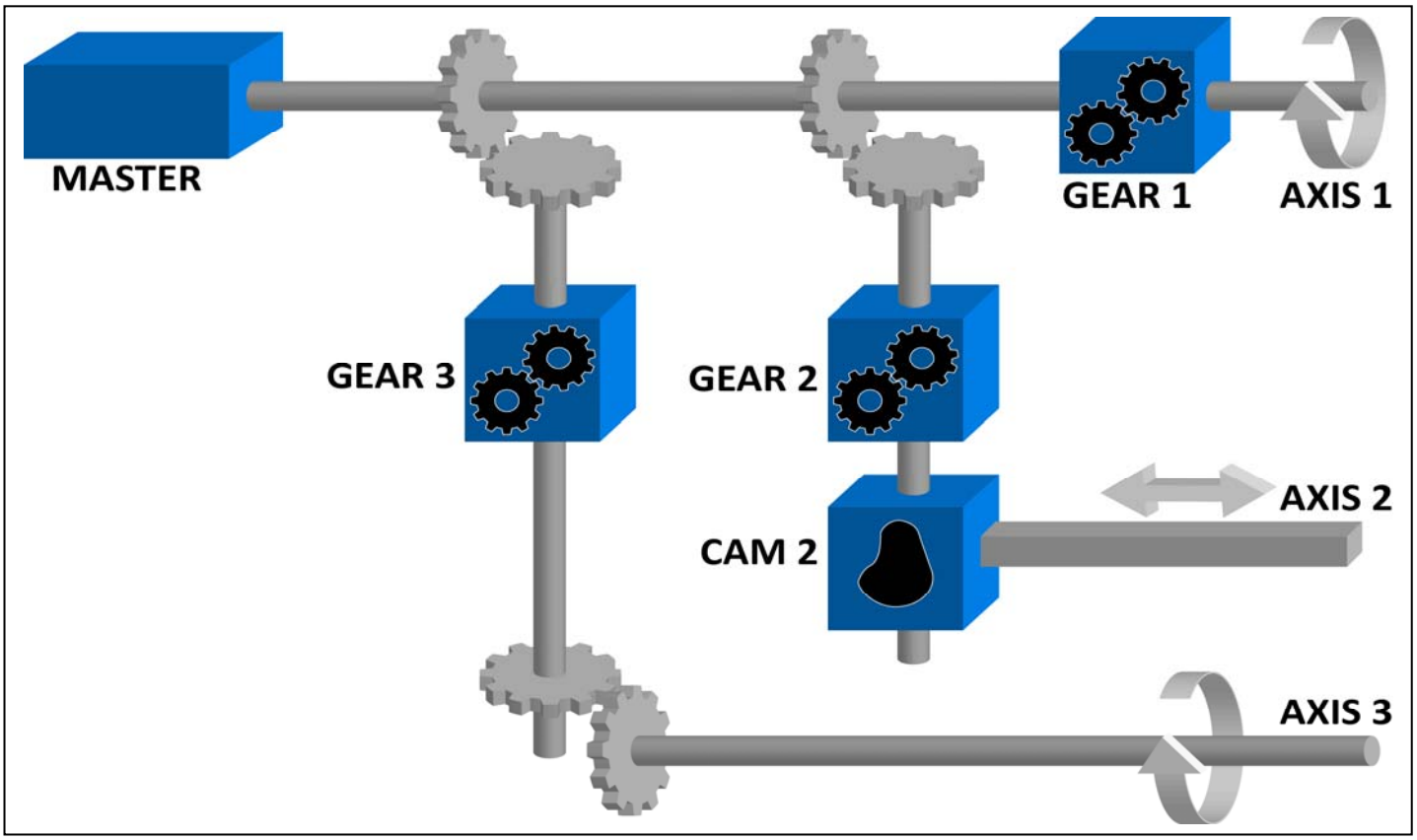


Pipe Network™ concept through mechanical analogy

Build Better Machines Faster by Simplifying the Creation of High-Performance Motion

Explanation on how to create a high-performance motion algorithm that is tightly integrated to the PLC program with a library of graphical function blocks.

Alex Hack, Product Manager and **Carroll Wontrop, Senior System Engineer**

One of the most challenging aspects of developing a motion control system is developing the motion profiles and associated code for each axis in the system, as well as the code that synchronizes these axes. While not easy even with simpler machines, this task can become challenging when dealing with complex high-performance systems with intricate synchronization requirements.

An alternative approach to motion algorithm development replaces the motion control code with blocks that can be dragged and dropped into place and connected together to develop the motion control solution. Programming and testing time are substantially reduced because of the intuitive nature of creating the topology and setting the attributes of the system.

The building blocks themselves have been extensively pre-tested so the developer can focus on optimizing the functionality of the system. For instance, Pipe Network™ motion control blocks from Kollmorgen are standard motion control terms and the flow of the information through the system is graphically represented so the system is self-explanatory and can easily be worked on by a team or transferred to a new programmer.

Conventional Motion Control Programming Methods

There are a number of different control programming methods defined by IEC61131 such as ladder logic and structured text. The conventional method of defining the motion algorithm is to use IEC61131 standard function blocks provided by the motion controller vendor. These function blocks are called by the programming language. When using standard function blocks, team members must visualize the topology of the system and the interactions between the different axes. Then they need to write a considerable amount of code to link together the axes and move them as a unit to meet the requirements of the application.

Like any hand programming effort, motion control code can become complicated and it can even become difficult for even its creator to understand. When problems arise, they are often very difficult to diagnose and correct. The code becomes even more difficult to manage or correct in the situation where the original creator moves on to another assignment or leaves the company.

Emergence of Pipe Network™

Kollmorgen supports the IEC61131 PLCopen motion standard function blocks but also offers as an alternative the Pipe Network through its integrated Kollmorgen Automation Suite™ system. Pipe Network is a new, innovative approach in which the overall machine control program is still produced in a standard IEC61131-3 language standard but the motion algorithm is created in a graphical environment by connecting building blocks together. The blocks actually contain the code required to define the motion control algorithm but they are transparent to the user who sees instead the Pipe Network motion blocks defined in standard motion control terms.

The use of pre-tested graphical building blocks means that there is no longer a need to remember all the minute details of the system so the programmer can focus on fine tuning the motion profiles and synchronization. Training a software engineer to create, maintain or understand a system by its graphical description is much simpler and takes much less time than working with the hand-written code.

Another advantage of the Pipe Network is that developers can take advantage of a simulator to develop and test their application without any hardware available. In addition, motion variables anywhere in the Pipe Network can be traced on a software oscilloscope. This approach helps to verify the created set of Pipe Network blocks and that performance will be optimized.

Kollmorgen Automation Suite™ is a complete, integrated set of software tools and hardware components designed to accelerate the development of high-performance machines. Its pre-integrated components and easy-to-use rapid software development tools help bring better machines to market faster and more affordably. Using KAS, OEMs can create machines that produce the highest-quality products with optimum throughput, minimal waste and outstanding OEE.

It can be used to record when an input is sensed in a cycle, how much correction is made in each cycle, etc.

A key advantage of the Pipe Network is that the graphical function blocks can be strung together in unique combinations to create functions beyond what can often be accomplished with conventional function blocks. For example, you can place multiple CAMS together to achieve a more precise motion that would allow you to increase the throughput of the machine.

Another advantage is that the granularity of the Pipe Network reduces delays in executing the motion algorithm. Servo feed forward can be incorporated into the Pipe Network to make motion adjustment faster and more precise.

Visualizing the Motion Algorithm

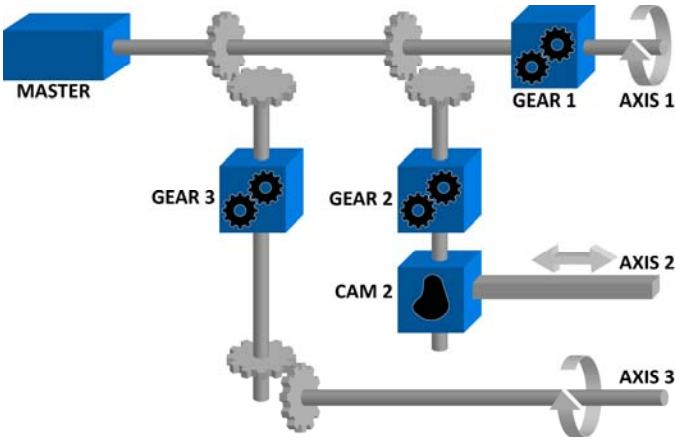


Figure 1: Visualizing the motion algorithm through a mechanical analogy

The Pipe Network concept can be defined through a mechanical analogy. In Figure 1, the mechanical system is composed of three axes and driven by one motor. All axes are connected to the motor through shafts, gears and CAMS. When the motor is in motion, all axes are driven synchronously. The speed relation between the master and the axis is achieved by using a mechanical gear. A mechanical CAM is used to get linear motion from a rotating wheel.

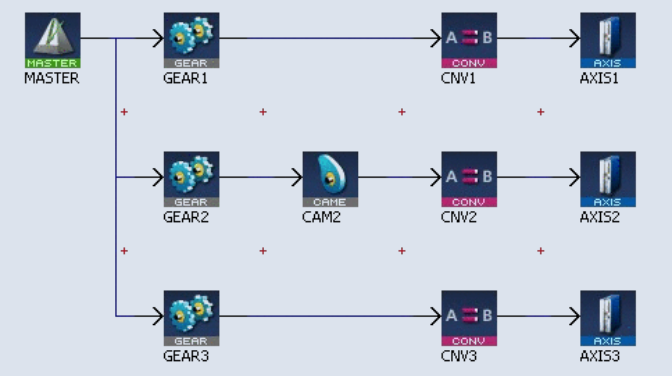


Figure 2: Pipe Network structure

The Pipe Network in Figure 2 corresponds to the mechanical system in Figure 1. Several pipes have been designed to control three dynamically interconnected axes. Each horizontal flow is considered a separate pipe. Relationships between the axes are developed and connected graphically, allowing you to visualize how the machine functions. Program code does not have to be written when setting up the foundation of a Pipe Network because required parameters can be entered into setup screens. The Pipe Network concept is a one-to-one translation of the mechanical system into the logical world:

- The main motor of the mechanical machine becomes a virtual master pipe block
- The gear boxes become gear blocks
- The mechanical CAM becomes a CAM block
- The axes become axis blocks

The Pipe Network concept enables machine designers to define in a very natural way the physical relationships between the different axes of the machine. The powerful modular approach provides a solution for almost any multi-axis machine. It also remains open for new functions that may be required in the future.

In programmable logic controller (PLC) programs, you can define activation or deactivation statements to install or remove pipes and pipe blocks. This allows the dynamic behavior of the machine to be adjusted based on conditions. The Pipe Network also contains a full library of single-axis motion commands for sections of an application where an axis operates independently.

Types of Pipe Network Blocks

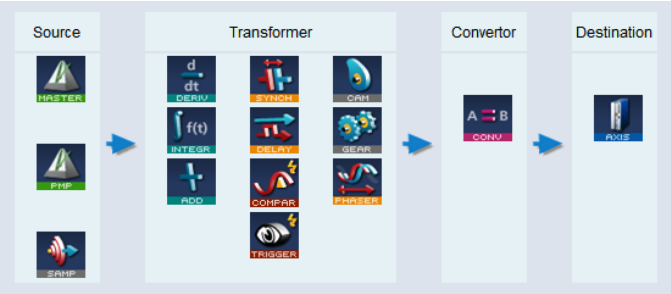


Figure 3: Overview of Pipe Network blocks

The general structure of a pipe block starts with an input pipe block as the source. Input pipe blocks work as a generator of values by sampling external source objects or creating a discrete flow of values from a virtual master as an input to the pipe. Transformer pipe blocks apply a specific algorithm to the input value to produce their output. Transformations can be either linear or complex and the type of transformation can be dependent upon the incoming values. An output pipe block is then used as a converter. Output pipe blocks can end a pipe by converting the incoming values from user units to correct internal units for the destination objects. The Pipe Network finishes with a destination pipe block. Destination blocks simply model a physical axis of the machine.

Master pipe blocks are used to create a virtual master to link two or more axes. The profile generator in a master block is trapezoidal. If a parabolic type profile is required, use the PMP pipe block instead. If the master is an external encoder or another axis, use the sampler pipe block.

The sampler pipe block is used to read an external encoder as an input signal into the Pipe Network or to directly read the actual position of another axis.

The gearing pipe block is used to perform electronic gearing. Initial gear ratios and the slope of gear changes are initially set with the pipe block parameters but they can be changed from the application program.

The CAM pipe block is used to optimize a motion profile. An adder block can be used with a CAM block to dynamically change the distance moved during each period or modulo of the machine load.

By tracking the position of one point of the Pipe Network, a comparator pipe block can be used to synchronize when code is executed in a PLC application program.

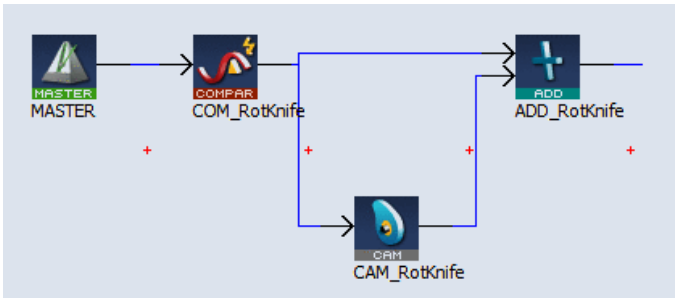


Figure 3: Modifying an offset move

The example in Figure 3 shows how an offset move can be changed by altering the amplitude or offset of the CAM pipe block. In a PLC application program, the MLCompWriteRef function block is used to arm the comparator block and the MLCompCheck function block is used to check the position. By using condition statements in a user program, specific actions, such as changing the move distance of the offset, can then be taken.

A trigger block can be used to read the position when a high speed input is triggered on the machine. The trigger block allows you to catch the position at a particular location in the Pipe Network, as required by the application.

Delay pipe blocks are used to delay the flow of position through a Pipe Network. It takes a couple of cycles to get information out of the drive and into the controller so by the time it is received by the Pipe Network is may be a few cycles old. Delay pipe blocks are used to delay the Pipe Network to synchronize it with real world motion.

The phaser pipe block is used to perform a dynamic phase adjustment inside the Pipe Network. The block can be used to phase-advance or phase-retard a position as required to synchronize different motion elements on a machine.

A synchronizer pipe block is used to synchronize two axes. This pipe block is useful in applications where it is necessary to start the motion of a second axis in sync with the first.

The converter block ensures that the pipe block data is formatted correctly for the drive.

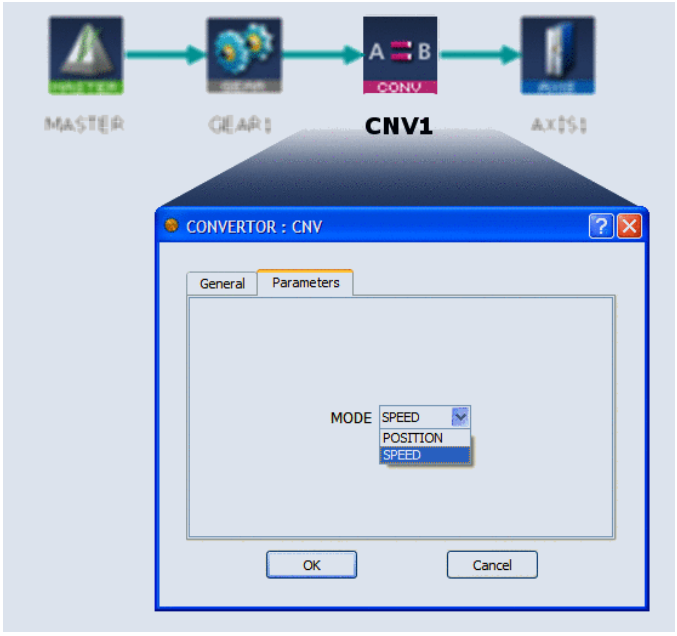


Figure 4: Converter pipe block

Understanding Position Data

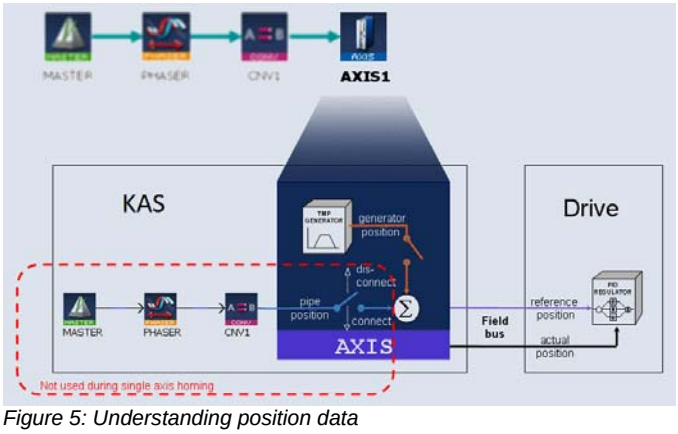


Figure 5: Understanding position data

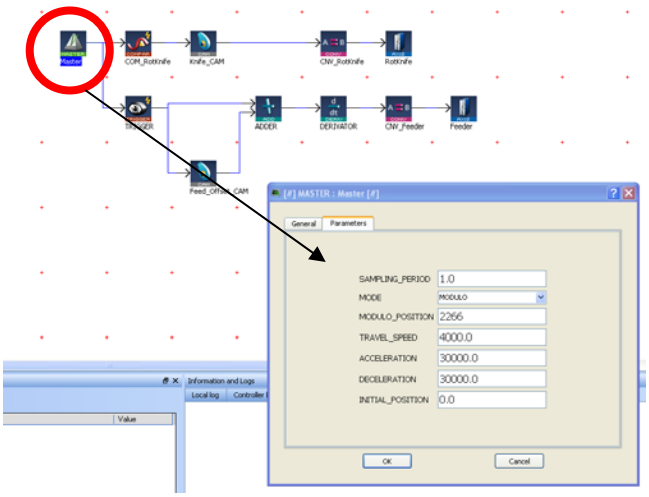
The axis pipe block makes the link between the physical and logical worlds by managing position data.

The motion command to a servo drive is called the reference position. The reference position is the sum of an axis generated position command and a Pipe Network position command.

The pipe position command is the input position of the axis issued by the upstream pipe through the convertor block and sent to the motion bus. The generator position command is the position profile generated by the axis block. The programmer has the option to generate motion to the axis from either or both the pipe or generator position commands.

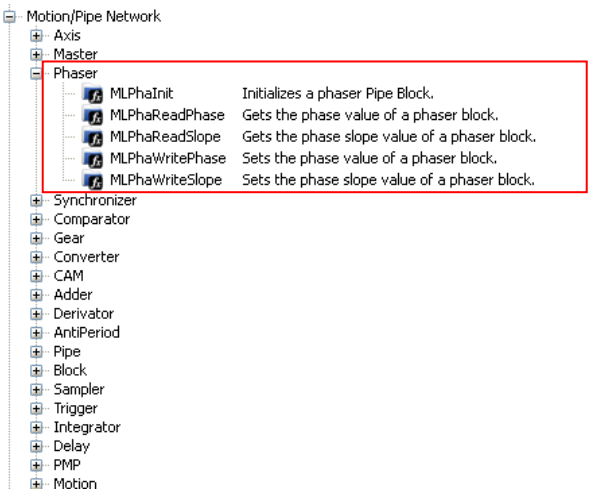
The feedback position is the absolute position provided by the drive through the motion bus. The motion command to a servo drive is called the reference position. The reference position is the sum of a position command from the axis generator and the Pipe Network.

To design motion with the Pipe Network, the user opens the Pipe Network editor, adds pipe blocks to the pallet, and connects the pipe blocks to define the motion algorithms. The user then edits the properties of the pipe blocks by right clicking on a pipe block and selecting the properties command. The parameters tab gives access to the properties that apply to each type of pipe block.



The user can also change pipe block properties dynamically in the control system code using function blocks (FB) predefined for each pipe network block type.

For example for the phasor block the following FB are available:



After creating the Pipe Network, the complete project is compiled which creates a list of functions (Figure 6) that can be used in the PLC program. These functions simplify programming by combining the same function block for all axes in the Pipe Network.

Pipe Network Function	Function Blocks included (for 2 axis system)
MLPN_ACTIVATE :	MLPipeAct(PipeAXIS1); MLPipeAct(PipeAXIS2);
MLPN_CONNECT :	MLCNVConnect(CNV1, AXIS1); MLCNVConnect(CNV2, AXIS2);
MLPN_POWER_ON :	MLAxisPower(AXIS1); MLAxisPower(AXIS2);
MLPN_POWER_OFF :	MLAxisPowerOFF(AXIS1); MLAxisPowerOFF(AXIS2);
MLPN_DEACTIVATE :	MLPipeDeact(PipeAXIS1); MLPipeDeact(PipeAXIS2);

Figure6: Pipe Network functions

Step	ML function blocks	Description
Motion Init	MLMotionInit	Initialization of the Motion is done with this dedicated function Set the Motion engine update rate. Wait for acknowledgement: MLMotionStatus() = MLSTATUS_INITIALISED to continue program operation
Create Cam Profiles	Profiles (MLPR_CREATE_PROFILES);	Create Cam Profiles from cam files
Create Pipe Network	PipeNetwork (MLPN_CREATE_OBJECTS);	
Motion Start	MLMotionStart	The Start function initializes the motion engine and prepares it for execution, then waits for acknowledgement: MLMotionStatus() = MLSTATUS_RUNNING to continue program operation
Power on all axes	PipeNetwork (MLPN_POWER_ON);	
Activate the pipes	PipeNetwork (MLPN_ACTIVATE);	
Connect the axes to the pipes	PipeNetwork (MLPN_CONNECT);	For example: in the following Pipe Network this function connects the Converter blocks (CNV1, CNV2 and CNV3) to the Axis blocks

Figure 7: Starting up the Pipe Network in the application program

Example Application

This example details a 5 axis form, fill and seal machine (FFS) whose motion algorithm was defined using the Pipe Network motion engine. The unwind axis maintains the desired tension on the film as the film transport uses registration marks on the film to position the film. When the film is positioned, the filling screw fills the bag, the sealing jaw seals the bag and the flying shear cuts on the seal to provide an individually formed, filled and sealed bag.

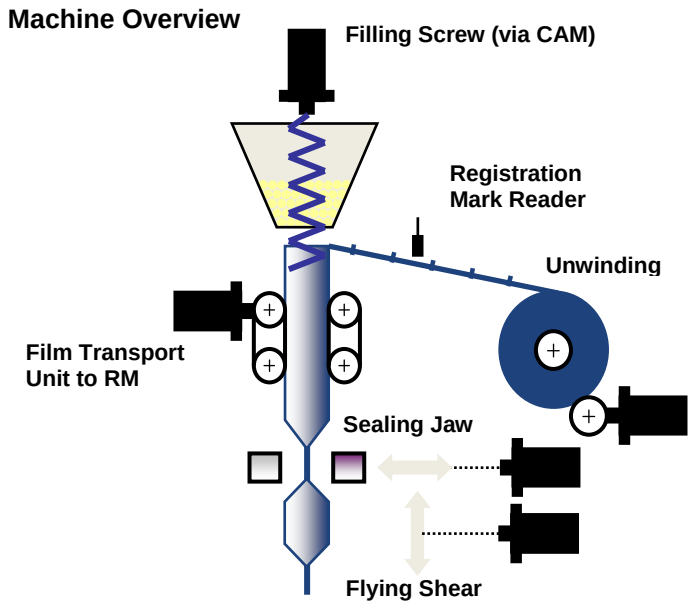


Figure 8: Form, fill and seal machine overview

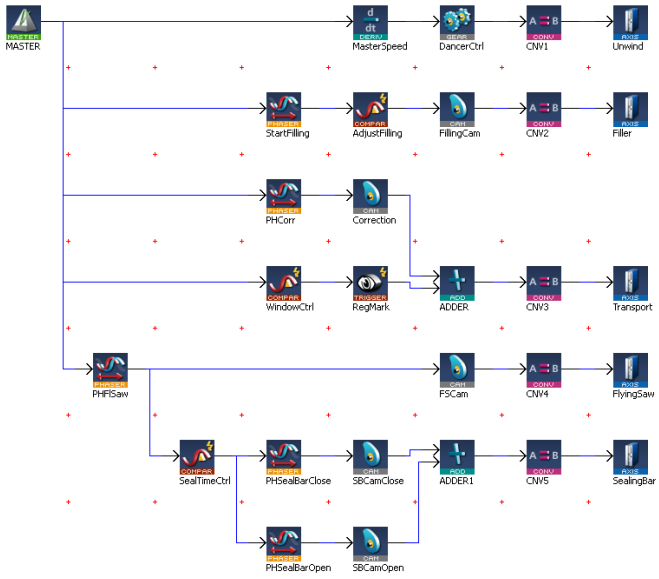


Figure 9: Pipe Network for form, fill and seal machine

The combination of pipe blocks and PLC code used in the form, fill and seal machine and shown in Figure 9 can easily be modified for use in a variety of form fill and seal applications, by modifying the CAM profiles and machine cycle times to fit machines with varying mechanics and desired material cut lengths.

Unwind Axis

The unwind axis is geared to the virtual master axis. As the film is being unwound from the parent roll a dancer roll is monitoring the desired tension. If the tension is outside the desired adjustable dead band the gear ratio between the virtual master axis and unwind axis is adjusted to maintain the desired tension. The tension (Regulation Tn), gain to control the tension (Regulation Kp), minimum and maximum dead band (Min Deadband & max Deadband) is entered on the operator panel.

Filling Axis

The filling axis is geared to the master virtual axis based on a profile in the code called FillingPro. The phase shift, speed, and quantity variables for the filling screw are entered on the operator panel. The phase shift is the position on the master axis that the filling axis profile (will start. The speed is the time from the start position to a point in the master position that the filling screw profile follows the master. The quantity is the amount of the FillingPro profile to be executed.

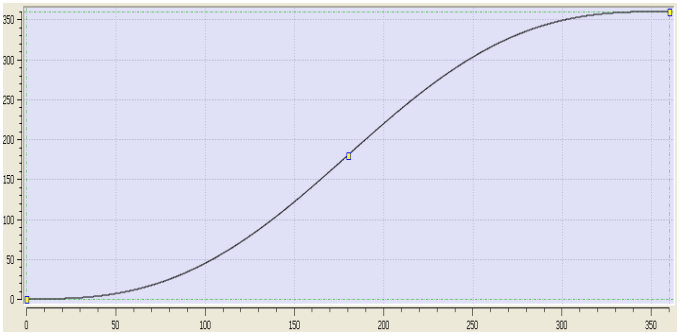


Figure 10: Filling screw base profile

Film Transport Axis

The film transport axis monitors the registration marks on the film and adjusts the film based on the registration mark parameters, Open Window, Close Window, and Reference Pos entered on the operator panel. If the registration mark is detected between the Open Window and Close Window the registration mark is considered a good mark and the Reference Pos value will be used to calculate the necessary adjustment. If the registration mark is detected outside the Open Window and Close Window values then the registration mark is ignored. Adjustments are not made during the seal process to avoid destroying product.

Sealing Jaw

The sealing jaw axis is geared to the master virtual axis based on two profiles (SealBarClose and SealBaropen). The profile starts at the same starting position as the flying shear. The sealing time is entered at the operator interface.

Figure 11: Sealing jaw base profile

Flying Shear

The flying shear axis is geared to the master virtual axis based on a profile (FlySaw). The profile starts at the phase shift entered at the operator interface.

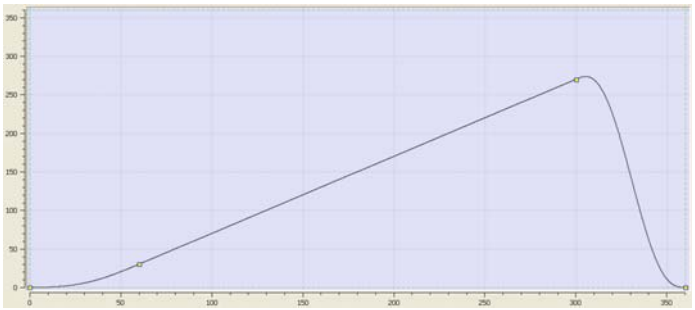


Figure 12: Flying shear base profile

PLC Program Code

Key Function Blocks and ASFBs used in the application

Name	How Used
MC_GearIn	Gears one axis to another
MC_CamIn	Gears one axis to another using a CAM profile
MC_TouchProbe	Arms the fast input and returns the latched position when the fast input occurs

Operation of the Pipe Network is integrated into the PLC program using the ST program language. The main section of the PLC code contains several If/Then statements. The program sets up gearing and CAM profiles for the slave axis to a virtual master. The virtual master controls the machine speed. The slave axes are manipulated by either changing the operator panel inputs or profiles. The program monitors registration marks on the film and makes correction to the film position based on the entries at the operator panel.

```
//*****************
// Machine Control
//*****************
IF bStart THEN
fStart(bStart);
  IF OldMachineSpeed<>MachineSpeed THEN
    MLMstRun(PipeNetwork.MASTER, MachineSpeed);
    OldMachineSpeed:=MachineSpeed;
  END_IF;
ELSE
  IF fStart.Q THEN
    MLMstWriteSpeed(PipeNetwork.MASTER,
MachineSpeed);
    MLMstAbs(PipeNetwork.MASTER, 720.0);
    OldMachineSpeed:=0.0;
  END_IF;
END_IF;
```

```
//*****************
// Dancer Control
//*****************
// Range 0-10V, 5V mid, 4-6V Deadband, <4V increase speed, >6V decrease speed
IF bStart THEN
  IF (AnaInDancer<MinDeadBand) THEN
    DancerGear:=(AnaInDancer-DancerRef)*Kp;
    IF Tn>0.0 THEN
      DancerGearI:=OldDancerGearI+(AnaInDancer-DancerRef)*Kp/Tn*1000;
      OldDancerGearI := DancerGearI;
    ELSE
      DancerGearI := 0.0;
      OldDancerGearI := 0.0;
    END_IF;
    DancerGear := DancerGear+DancerGearI;
    IF DancerGear > LREAL#1.0 THEN
      DancerGear:=LREAL#1.0;
      OldDancerGearI := LREAL#1.0;
    ELSIF DancerGear < LREAL#-1.0 THEN
      DancerGear:=LREAL#-1.0;
      OldDancerGearI := LREAL#-1.0;
    END_IF;
    OldDancerGear:=DancerGear;
    MLGearWriteRatio(PipeNetwork.DancerCtrl, (LREAL#1.0+DancerGear));
  ELSIF (AnaInDancer>MaxDeadBand) THEN
    DancerGear:=(AnaInDancer-DancerRef)*Kp;
    IF Tn>0.0 THEN
      DancerGearI:=OldDancerGearI+(AnaInDancer-DancerRef)*Kp/Tn*1000;
      OldDancerGearI := DancerGearI;
    ELSE
      DancerGearI := 0.0;
      OldDancerGearI := 0.0;
    END_IF;
    DancerGear := DancerGear+DancerGearI;
    IF DancerGear > LREAL#1.0 THEN
      DancerGear:=LREAL#1.0;
      OldDancerGearI := 1.0;
    ELSIF DancerGear < LREAL#-1.0 THEN
      DancerGear:=LREAL#-1.0;
      OldDancerGearI := LREAL#-1.0;
    END_IF;
    OldDancerGear:=DancerGear;
    MLGearWriteRatio(PipeNetwork.DancerCtrl, (LREAL#1.0+DancerGear));
  ELSE
    // Deadband
  END_IF;

END_IF;
```

```
//*****************
// Filling axis
//*****************
IF MLCompCheck(PipeNetwork.AdjustFilling) THEN
  MLCompReset(PipeNetwork.AdjustFilling);

MLPhaWritePhase(PipeNetwork.StartFilling,PhaseFilling);
  MLPrfWriteIScale(Profiles.FillingPro,IRatioFilling);
  MLPrfWriteOScale(Profiles.FillingPro,ORatioFilling);
END_IF;

//*****************************************
// Transport axis with registration control
//*****************************************
IF MLCompCheck(PipeNetwork.WindowCtrl) THEN
  IF bWindowOpen THEN
    MLCompWriteRef(PipeNetwork.WindowCtrl, (OpenWindow<MLBlkReadOutVal(PipeNetwork.WindowCtrl)), OpenWindow);
    MLCompReset(PipeNetwork.WindowCtrl);
    bWindowOpen:=FALSE;
    IF MLTrigIsTrigged(PipeNetwork.RegMark) THEN

CorrLength:=MLTrigReadPos(PipeNetwork.RegMark)-ReferencePos;
      IF CorrLength>MaxCorrLength THEN
        CorrLength:=MaxCorrLength;
      ELSIF CorrLength< -MaxCorrLength THEN
        CorrLength:=-MaxCorrLength;
      END_IF;
    ELSE
      CorrLength:=MaxCorrLength;
    END_IF;
    MLPhaWritePhase(PipeNetwork.PHCorr,LREAL#-1.0*CloseWindow);
    MLPrfWriteOScale(Profiles.Corr, CorrLength);
  ELSE
    MLCompWriteRef(PipeNetwork.WindowCtrl, (CloseWindow<MLBlkReadOutVal(PipeNetwork.WindowCtrl)), CloseWindow);
    MLCompReset(PipeNetwork.WindowCtrl);
    bWindowOpen:=TRUE;
    MLTrigClearFlag(PipeNetwork.RegMark);
    MLAxisRstFastIn(PipeNetwork.Transport, 0);
  END_IF;
END_IF;
```

```
//*************
// Flying saw
//*************
IF PhaseFlySaw <> OldPhaseFlySaw THEN

MLPhaWritePhase(PipeNetwork.PHFlSaw,PhaseFlySaw);
  OldPhaseFlySaw := PhaseFlySaw;
END_IF;
```

```
//*************
// Sealing bar
//*************
IF MLCompCheck(PipeNetwork.SealTimeCtrl) THEN
  MLCompReset(PipeNetwork.SealTimeCtrl);

MLPhaWritePhase(PipeNetwork.PHSealBarOpen,LREA
L#-270.0);
  IF bStart THEN
    bStartSealTimer := TRUE;
  END_IF;
END_IF;

TimerSealing(bStartSealTimer,SealTime);
IF TimerSealing.Q THEN
  bStartSealTimer := FALSE;

MLPhaWritePhase(PipeNetwork.PHSealBarOpen,LREA
L#-1.0*MLBlkReadOutVal(PipeNetwork.SealTimeCtrl));
  MLPrfWriteOScale(Profiles.SealBarOpen,360.0);
  MLPrfWriteOOffset(Profiles.SealBarOpen,LREAL#-
90.0);
END_IF;
```

Operator Panel

Within the integrated Kollmorgen Automation Suite system, there is an internal control panel for use when creating and debugging the project. The control panel sets the desired values for each product and to start and stop the machine. The specific entries are described in the axis sections above. Below is a screen print of the control panel used this application.

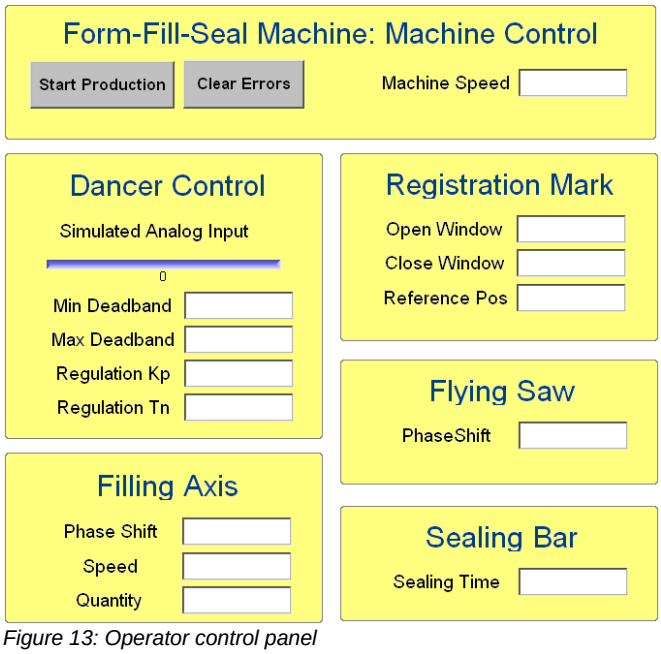


Figure 13: Operator control panel

Conclusion

Machine designers can build a better machine faster and simplify the creation of high-performance motion by using the Pipe Network over other conventional programming methods

Pipe Network delivers exceptional performance and effectively reduces development time by making it possible to create a high-performance motion algorithm that is tightly integrated to the PLC program with motion library function blocks. The Pipe Network concept provides a simple conversion of mechanical applications into a graphical representation of application elements and the process flow with improved quality and accuracy. This format makes it easy to understand, program and update the motion profiles and positional relationships.

ABOUT KOLLMORGEN

Kollmorgen is a leading provider of motion systems and components for machine builders around the globe, with over 70 years of motion control design and application expertise.

Through world-class knowledge in motion, industry-leading quality and deep expertise in linking and integrating standard and custom products, Kollmorgen delivers breakthrough solutions unmatched in performance, reliability and ease-of-use, giving machine builders an irrefutable marketplace advantage.

For more information visit www.kollmorgen.com, email support@kollmorgen.com or call 1-540-633-3545.

 KM_WP_000162_RevA_EN