YASKAWA

PLCs and Robots

The Quest for a Single Controller

Edward Nicolson, Senior Director, Motion Product Development
Yaskawa America, Inc.

1 Abstract:

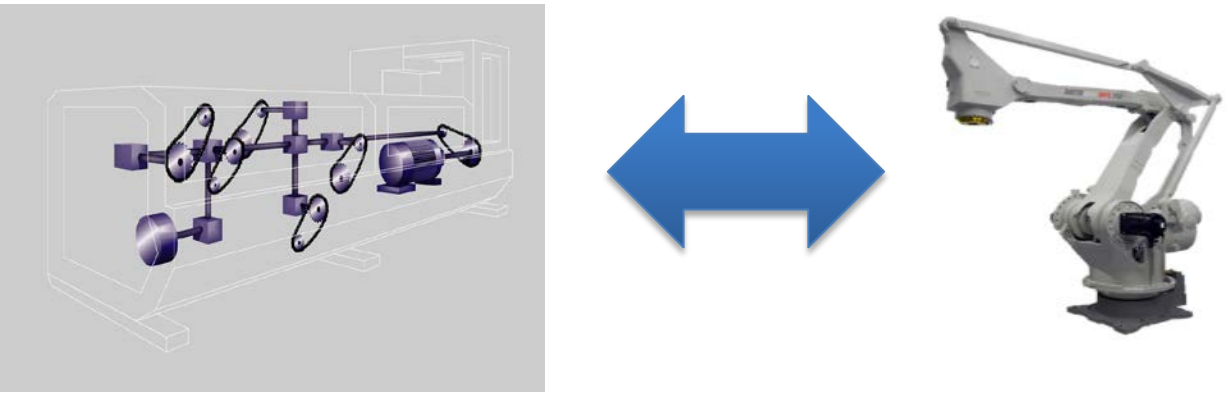
As automation applications continue to grow, two distinct areas are starting to merge into common deployments: robotics and motion. Robotics has traditionally been a path-dependent event-based application. In contrast, motion control has more typically been a scan-based velocity or positioning application, with the exception of machine tools.

The contrast between the two is not as great as in the user interface where robotics tends to be very teach pendant centric and motion control uses programming methods such as structured text, ladder and others. As these technologies merge together, they present many technical challenges. This paper analyzes the two architectures and presents some possibilities for combined solutions.

2 Motivation:

This article is whimsically sub-titled “The Quest for a Single Controller”, the implication being that a single controller could be as achievable as finding the Holy Grail. As we will review, there are many compelling reasons why a single controller should be used. However, there are opposing factors that can make this goal difficult to achieve as a successful product.

Consider the problem faced by an end user in an automated food packaging plant where a packaging machine and a robot are to be installed to complete a packaging and palletizing application. The plant manager desires to get the machines integrated as quickly and efficiently as possible to start production.



The packaging machine contains motors, actuators and IO and is run by a PLC controller with integrated motion capabilities. The robot arrives with a handheld teach pendant for programming and operation. In order for these two pieces of automation equipment to be placed into production, the plant employs application engineers to complete the PLC programming. A separate applications engineer completes the robot programming. As the task of integrating the equipment proceeds, it may get blocked when one or more of the resources are unavailable to continue work on one system. The plant is also faced with the additional cost and support for two different control platforms. As production starts, the plant may also find that the system fails to meet rate due to the communication delays between the two controller systems.

Given issues of multiple applications groups, increased cost, and sub-par performance, the plant manager asks the question, “With all the high speed microprocessors available on the market today, why can’t this all just be integrated in one controller with a single applications environment?”

This question can be analyzed by contrasting the internals of motion and robot controllers from the viewpoint of their hardware architecture, software architecture, safety systems, and application environment.

3 Hardware Architecture

Table 1 compares the hardware architecture of a motion controller and a robot controller. As the table shows, motion controllers support a rich set of options for motors, I/O and drives. A deterministic network is used to integrate these components into a stand-alone controller box. In contrast, a robot controller typically uses a single type of actuation, has a limited number of sensors, and includes a cost-optimized multi-axis drive package that contains power conversion capacity tailored to the needs of the robot. The robot controller typically will use cards, instead of stand-alone boxes, in the cabinet to reduce cost.

	PLC + Motion	Robot
Motor	Servo motor, stepper motor, pneumatic actuator, VFD, multi vendor motor and drive.	Servo motor
I/O	Many I/O points (100 or more)	Camera and a few IO points.
Drive	Individual drive units with internal power conversion	Integrated drive units with power conversion optimized for the robot payload and motion.
Servo Network	Open network for combining multiple vendor solutions.	Usually an internal network or backplane
Motion Controller	Stand-alone PLC box	Controller card in a backplane
Safety	Safety PLC box	Safety module card in backplane
User Interface	HMI panel Fixed location	Teach pendant Handheld

Table 1: Comparison of hardware architectures

Interestingly, the cost optimization attempted by the use of customized drive packages compared to individual drive units of motion controllers can be thwarted by the large difference in production volume of the two units. Robot controllers are typically manufactured in a smaller volume than individual drives, allowing the individual drive solution to be cost competitive, or less expensive, than the customized multi-axis drive solution.

One other point of difference is in the HMI. Motion controllers typically have a machine mounted touch panel, whereas robots have a hand-held teach pendant.

Based on this comparison, outside of the safety hardware, it is quite feasible to create a robot controller from the components of a PLC+Motion controller, especially if a vision system is added and a portable HMI touch panel is used.

4 Software Architecture

A comparison of the software architecture of a motion and robot controller is shown in Table 2.

	PLC + Motion	Robot
Drive and Servo Network	Support multiple control modes, torque, velocity, position, interpolation	Mainly interpolation with feed forward
I/O	Field bus drivers for remote I/O, local I/O drivers synchronized to motion scan.	Built-in I/O drivers, vision may be integrated into motion software.
Motion Controller	Single axis point to point, cam/gear master/slave relationships, virtual axes, torque, velocity, position, actual path can be critical.	Interpolated position with path blending, kinematics, handling of singularities, focus on end point in position.
Logic	Integrate wide array of I/O and logic calculations into a continuous scan	Trigger motion off of I/O events or I/O off of motion.
Communications	Support MES communications	Support communications to cell controller. May be remotely controlled by PLC.
Safety	Safety PLC is programmed separately.	Safety Module is included in the controller.
User Interface	HMI is programmed separately as part of the application	Teach pendant is used as the programming tool

Table 2: Comparison of software architectures

A key difference is that a motion controller must support a rich set of control options, from control modes to master and slave axis relationships. In contrast, a robot controller typically uses a single control mode (interpolation) to the drive. The complexity in a robot controller comes from the multi-axis kinematics calculations and the trajectory-blending calculations. Robot controllers often focus more on end point positioning and less on the accuracy of the path followed, whereas the actual path is often critical for motion controllers.

With the addition of kinematics, path blending and singularity detection to a motion controller, it is conceivable that from a software architecture standpoint it is also possible to include the required components of a robot controller inside a motion controller.

5 Safety Architecture

A comparison of the safety architecture of a motion and robot controller is shown in Table 3.

	PLC + Motion	Robot
Motor / Encoder	May need safety encoder	May need safety encoder
Drive	A safety option is added on each axis. The option monitors the drive and shuts it down if safety parameters are exceeded.	A safety module monitors all axes together.
Servo Network	Black channel safe communications	Encoder lines may feed directly to the safety module.
Motion Controller	Responsible for controlling the machine using inputs from safety system. Safety PLC monitors the safety inputs.	Responsible for keeping the robot inside a working envelope.
Safety monitoring	Each drive monitors the safety conditions on a per axis basis.	The safety module tracks the worldspace position of the robot, including all its links. Must implement kinematics.
Certification	Machine safety Risk assessment : ISO 13849 Electrical components: EN 61508	Robot safety ISO 10218-1 Application specific standard Implementations are clearly defined.

Table 3: Safety architecture comparison

Safety certification is one area that provides a clear challenge for the integration of a motion and robot controller. The robot safety standard, ISO 10218-1, provides clear instructions for the implementation requirements of a robot controller in order to meet certification standards based on risk assessment done on robots. However, the motion controller application must first go through a risk assessment governed by the machine safety standards, such as ISO 13849. After the risk assessment is done, the required safety components and logic can then be determined.

Another key difference is that the safety module for a robot controller must include a kinematic model for the robot and must receive the encoder positions for all the axes. With this information, the robot safety module can determine when the robot is within its safe area. For a motion controller, safety is implemented on a per-axis basis with optional add-on advanced safety modules to each drive.

These differences present two challenges to implementing a robot safety system based on a motion controller. First, the Safety PLC must be capable of receiving all the axis positions and must have a complete kinematic model that is guaranteed to match the physical system. Second, for the manufacturer of the controller, a method for creating a safety sandbox for the robot control must be found so that the non-robot part of the motion application does not cause a breakdown in the safety system.

6 Application Environment

A comparison of the application environment for motion and robot controllers is shown in Table 4.

	PLC + Motion	Robot
Drive	Drive tuning software	Hidden from applications engineer
Servo Network	Applications engineer must wire and configure the network.	Hidden from applications engineer. May allow an external axis.
I/O	Map remote and local I/O by means of variables or tags.	Vision system programming. I/O typically in a fixed configuration.
Motion Controller	Combine function blocks to initiate, modify and terminate motion.	Call routines to start and wait for motion to complete.
Languages	Ladder, Structured Text, Function block diagram, Sequential function chart	Procedural textual language, point and path teaching, I/O events
Debugging	Live display of program variables, download of updates to live production machine.	Step through the robot program.
Error Handling	Set error code, stop machine, manual recovery	Catch error conditions and perform automatic handling. Have to recover from the appropriate point.
User Interface	HMI programming software	(N/A)

Table 4: Application environment comparison

What is immediately apparent is that the skill set required for motion controller applications engineers is significantly more comprehensive than that for a robot applications engineer. Motion application engineers must be able to tune servo drives, configure and connect servo networks, create an application involving multiple languages in the PLC environment through the combination of function blocks, and must be able to create an HMI application. In contrast, a robot applications engineer will typically teach locations for the robot to pass through and use those locations in a script language to

cause the robot to sequence through the taught positions. Complexity in the robot application comes through the addition of a vision system that requires dynamic adjustment of the end point targets.

The most important difference between the two environments is in the scan-based nature of a PLC and motion controller compared to the event-based nature of robot programming. This will be examined in more detail in the following sections.

6.1 Application Languages

In , a typical state machine implemented in the IEC 61131-3 Sequential Function Chart (SFC) language is presented. In this language, individual states, transitions between states, and actions taken in each state are graphically represented. When debugging in this environment, it is possible to see the active states and follow transitions between states. It is conceivable to implement a robot pick and place sequence in such a state diagram by representing each stage in the sequence as a step and the transitions between the states as the IO events and other triggers.

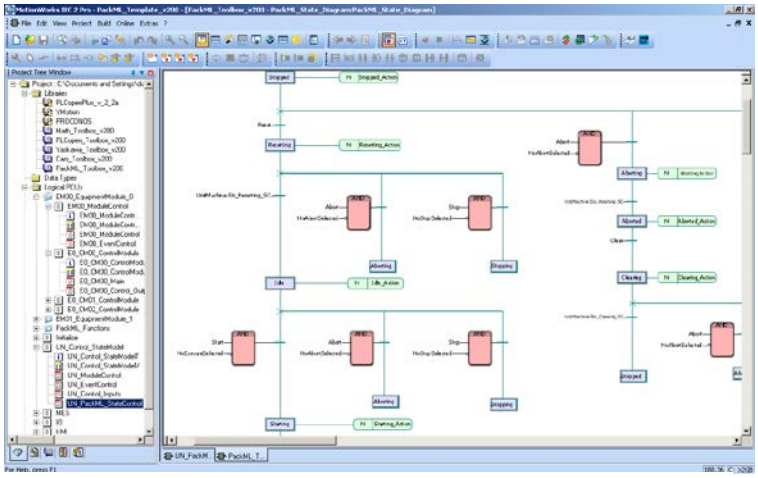


Figure 1: SFC example code

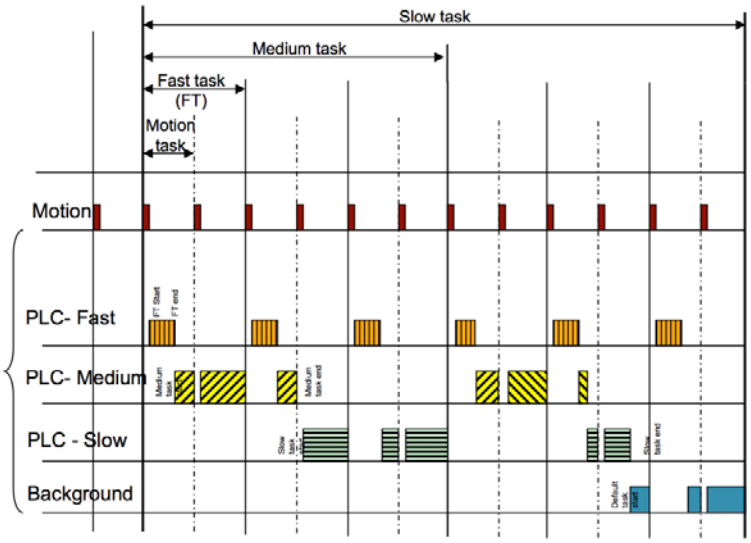


Figure 2: Example robot code

Comparatively, the pick and place sequence in a typical robot text language is shown in Figure 2. It is immediately apparent that the robot text language is a much simpler way of expressing the intended sequence of operations. It is quicker for an application engineer to put together this text sequence than it is to implement the robot sequence in the SFC, or any other IEC 61131-3 language. The key reason is the difference between event and scan-based architectures, which will be analyzed in the next section.

Task Scheduling in Motion and Robot Controllers

Figure 3 shows the typical task scheduling in a PLC motion controller. The motion application engineer will assign portions of the application code to each of the fast, medium, and slow tasks. Actual motion processing is handled by the motion task, running at the highest priority and frequency. The challenge for the application engineer is that at no time can the code assigned to each task exceed the time period for the task, otherwise a watchdog will occur and the PLC will stop execution. This can be challenging when the code execution for the task varies greatly depending on the amount of code to be executed during a particular scan. If a programmer has not been exposed to the constraints of hard, real-time application programming, the constraints that watchdogs impose on applications can be difficult to overcome.

In contrast, a robot script program can be considered to run like the background task in the diagram. A background task runs at lower priority, and is interrupted by, the motion task. The script engine in the robot controller will execute the script until it is blocked by an event, such as waiting for a motion to finish or an I/O to trigger.

As has been discussed, it is possible to implement the robot application in the scan-based environment using a PLC language such as SFC, but consider how that code will actually execute on the controller. In the scan-based implementation, the code enters the move state. During each scan, the move completion status is checked. When the move is detected to be complete, then execution continues to the next state. This is the essence of a polling architecture.

In contrast, the central processing unit load in the case of an event-based script language is much lower. In this case, the script task will block (i.e. wait) for a system event such as a semaphore until the time that the system determines it is time for execution to proceed. This relieves the central processing unit from the task switching and repeated code execution required in a scan-based environment.

Example

```
NOP
MOVJ VJ=50.00          • • • Step 1
MOVL V=220             • • • Step 2
MOVL V=200             • • • Step 3
WVON WEV#(2)           • • • Weaving starts
ARCON AC=220 AVP=100 T=0.50 • • • Welding starts
MOVL V=138             • • • Step 4
ARCOF AC=160 AVP=90 T=0.50 • • • Welding ends
WVOF                   • • • Weaving ends
MOVL V=200             • • • Step 5
MOVJ VJ=50.00          • • • Step 6
END
```

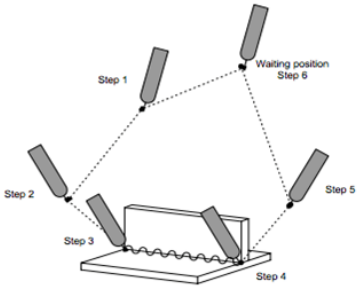


Figure 3: Task Scheduling

7 Analysis

It is time to review the question: “Could a Robot controller be implemented as a special single purpose use of a PLC+Motion controller?” The answer can be “Yes” if you conclude that the PLC and motion controller appears to have all the hardware, I/O and other required components, and it is technically feasible to put kinematics and other requirements in the controller. On the other hand, the answer is “No” if you conclude that a PLC and motion controller lacks the programming languages, vision integration, and trajectory generation capabilities required for a robot controller.

To some degree, the question is not what is technically feasible, but what is the best application user experience. Many software engineers have been steeped in procedural programming skills from the computer science education. Polling is often taught to be the wrong way to implement applications and is shunned. Programmers with this background find it awkward to then start programming a procedural robot sequence in an environment that requires polling. Although text languages, like structure text (ST), exist in the IEC 61131-3 environment, this text is scanned so it is confusing for procedural programmers to realize that they must keep track of the state for each scan and switch the code to be executed based on the current state.

In many ways, it would be much easier and more natural to add a robot scripting language to the PLC and motion controller that executes in an event based fashion. Ideally, this would be a standardized language, not one that is proprietary to a particular vendor. With this addition it would be possible to program the motion and robot code side by side. The motion applications engineer would then need to be taught one more language environment and be able to complete the entire combined controller application. Of course there are some technical issues to resolve such as:

- How can this be debugged in a step by step fashion?
- How to tie events into scanned code?
- How to train application engineers to think both in scan and event-based ways?
- Can error handling be coordinated between the two environments?

The other looming issue that has been discussed is safety. To resolve the issue of safety certification would require deeper analysis of the safety standards. In the worst case, a complete risk assessment of the combined motion and robot system would need to be conducted. Unfortunately, that might be cost prohibitive to do on a per-site basis.

This paper has reviewed the possibility of creating a combined motion and robot controller. Creating such a controller is entirely feasible, however the best application developer experience will be achieved if a standardized event based scripting language can be added to the PLC and motion controller.