Session 1559

Factory Automation Emulator Design Using Object-Oriented Programming

Bruce Segee, Vincent Allen
University of Maine

Abstract

Development of a factory automation emulator for a complex material handling system having multiple interconnected conveyor belts can be a difficult task. Our goal was to develop a factory wide emulator that would mimic the actions of all conveyor belts, photosensors, barcode readers, diverter arms, motors, other actuators, and even the operators. The emulator would move virtual totes of partially finished product through all phases of assembly causing the programmable logic controllers (PLC's) and PC's to believe they are controlling the real factory and routing real totes. Our solution was an object-oriented software emulator written in C++. Development of the software involved breaking the automation system into multiple zones. Multiple zones allow the software programmer to focus on only a small number of inputs and outputs for a given zone, thus reducing the overwhelming task to one that is more simple. Each zone is represented by a software module, or C++ object, that has an input function, an output function, and an update function. These functions allow objects to enter a zone, leave a zone, as well as update position within a zone, respectively. Combining the zone modules with a graphical representation of the system resulted in an effective, real-time emulation tool. With a small group of programmers at a variety of different skill levels, the development of the software took only a few weeks.

1. Introduction

An emulator was needed to test code developed for controlling an automated goods delivery system before installing it in a factory. A small amount of time spent on an emulator would greatly reduce the time spent debugging the software on site. The purpose of the emulator was to emulate all aspects of the factory right down to an operator working on a tote. Most importantly it had to emulate totes moving throughout the system as well as emulate all hardware, i.e., the photosensors, barcode scanners, actuators, diverter arms, and motors.

Before discussing the framework of the emulator, some background of the conveyor system and hardware must be covered. The factory has multiple lines of many operator stations. Totes of partially assembled product are stored in storage areas located overhead. Each of the storage areas consists of many, gravity fed, columns that can hold up to four totes each. A circular conveyor belt, also located overhead, is used to move totes from a storage area on one line to a storage area on another line. A series of belts are responsible for moving totes from the operator stations back to storage. Other miscellaneous conveyor belts are used for entering totes into the system, routing them to areas where defective product can be dealt with, and exiting totes from the system.

Each of these different belts and gravity fed rollers can be implemented in an emulator as a separate zone. Each operator station is a zone. Each storage column makes up a zone and an

array of storage column zones is considered a storage area. The circular conveyor belt as well as each belt in the series of belts needed to get from storage and to storage also makes up a zone.

Combining all of the zones in one application that provides a graphical display to show totes within the emulated system produced a software emulator to fit our needs. The following section discusses the software framework for the emulator as well as the design of one of the zones.

2. Generalized Software

Each of the zones previously mentioned was implemented as a C++ object. The C++ programming language was chosen because the language's object-oriented aspect lends itself very easily to solving real world problems. Once all the C++ objects were written they could be fitted together like building blocks into a software emulator.

Each zone object had to have certain characteristics. Each object had to have an input function, an output function, and an update function. The input and output functions were responsible for entering totes into and exiting totes out of a zone, respectively. The update function was responsible for updating the location of a tote within the zone. The update function would be passed a certain amount of time, Δt. Given the amount of time elapsed and the characteristics of the zone, the tote would move a certain distance within the zone or it may exit the zone to enter another zone. The connecting of zones is accomplished by using pointers. Each zone contains a pointer to the next zone object or a number of possible next zone objects. The following section discusses the implementation of the launch zone.

2.1 Development of the Launch Zone

The following is an example of how a tote travels through the launch zone. When a tote exits a storage column it moves onto the launch belt (see Figure 1). After traveling past the first storage column, it passes a photosensor and a barcode scanner. When the sensor trips, the barcode scanner reads the barcode located on the side of the tote. As the tote travels further down the belt

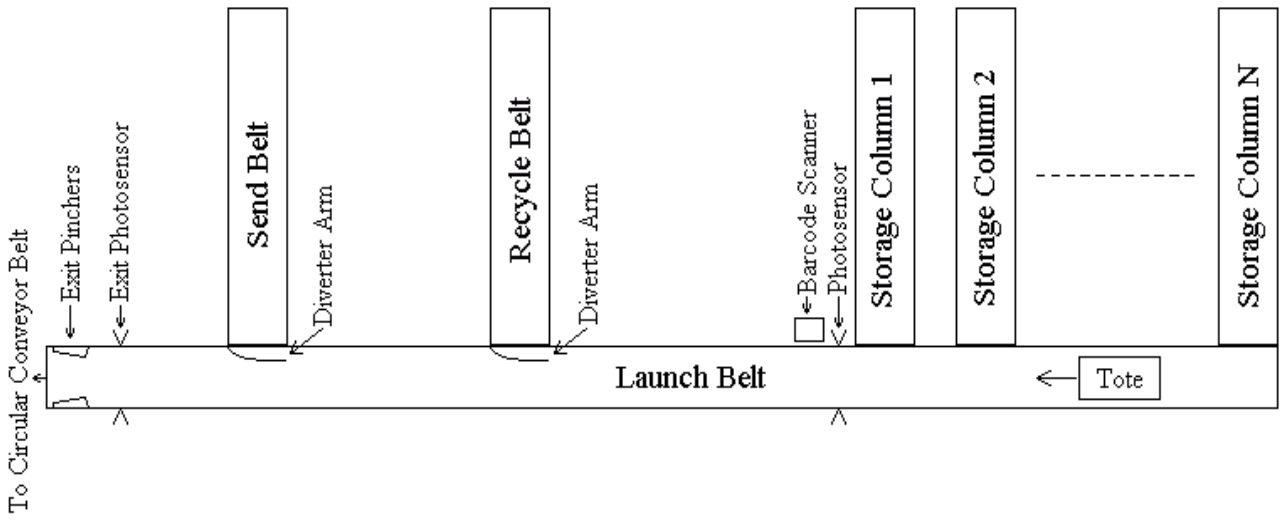

Figure 1: Schematic of Launch Zone

it approaches the recycle belt. If the tote is to be recycled, the PLC will lower a diverter arm to move the tote onto the recycle belt. If not, the tote will travel further down the belt towards the send belt. If the tote is being sent down to an operator station, it needs to be directed onto the send belt. Based on a longer amount of time elapsed since passing the barcode scanner, the diverter arm will lower and direct the tote onto the send belt. If the tote does not need to go to an operator station, the diverter arm will not lower and the tote will continue to travel down the launch belt towards the circular conveyor belt marking the end of the launch belt. A set of pinchers and a photosensor are located at the end of the launch belt. The photosensor is responsible for telling the PLC, which in turn tells the PC, when the tote has reached the end of the belt. The pinchers are responsible for preventing a tote from entering the circular conveyor belt if another tote is passing by the launch belt exit. This is to avoid any collision between totes.

In the emulator the launch zone is implemented as a C++ object. The object has an input function that is called by one of the many storage column zone objects each containing a pointer to the launch zone object. The parameters to the launch zone's enter function are a tote barcode and an entrance time. The entrance time is the time it will take the tote to get to the end of the launch belt from the location that it entered the launch zone. From the above description of the launch zone it is clear that a tote can exit the zone into one of three different zones. For this reason the launch zone has three different exit functions each containing a pointer to a different zone. The launch zone object has an update function that is passed a time, Δt. The function is responsible for subtracting the time, Δt, from the tote object and checking whether or not the tote has reached a critical time. A critical time is one that equals the time from the barcode scanner and photosensor to the end of the launch belt, the time from the recycle belt diverter arm to the end of the launch belt, the time from the send belt diverter arm to the end of the launch belt, or the time to the pinchers and photosensor located at the end of the launch belt. Depending on the amount of time elapsed in the update function the object could call one of three different exit functions. Each of these exit functions is responsible for exiting the tote out of the launch zone and entering it into the next zone. The function does this by calling the next zone's enter function with a tote barcode and an entrance time.

As mentioned above, the tote can be at four critical times within the launch zone. At each of these critical times, the software must interact with the PLC inputs and outputs. The I/O needed by the launch zone object is the following: mimicking the barcode scanner by placing the tote barcode into the PLC's memory, checking to see if a diverter arm is down at the recycle belt or the send belt, determining if there is a tote at the end of the launch belt, or to see if the pinchers are activated at the end of the launch belt.

Instead of making many reads and writes to and from the PLC's memory whenever necessary, an I/O object was developed to handle the data transfer to and from the PLC memory. This approach greatly reduces the data highway traffic necessary to handle the large amount of I/O. Before any zone's update function is called, the I/O object reads all bits from the PLC's memory. After all zone updates, the I/O object writes the modified bits to PLC memory.

2.2 Graphical Display

A graphical display was also developed to help show the location of totes in the emulated system. A small version of the factory including the storage area, operator stations, and the launch and store belts were displayed for three separate lines. The display was also aided by a text box that

displays strings telling what zone the tote was in. This allowed the user to know a tote's location when it was in a zone that the emulator did not display.

Implementing the visual part of the emulator was straight forward once the objects were written. A draw member function was implemented for each zone object that was to be displayed on the screen. With each update of a zone, the zone's draw member function would be called to redraw the updated zone.

3. Conclusion

The end result of the project was a software application that emulated all aspects of the hardware of the control system, the movement of totes, and the behavior of operators. Totes of unfinished product could be entered into the system using a hand held barcode scanner. Once in the emulator, the totes traveled between operators and lines until completed. Upon completion the totes exited the emulator. The emulator proved to be extremely beneficial in the development and debugging of the control software for the system. The emulator also took a relatively short amount of time to complete utilizing a considerable number of student programmers of vastly different skill levels.

4. References

[1] Perry, Greg, "Moving From C to C++", Sams Publishing, 1992

[2] "Turbo C++ User's Guide", Borland International Inc., 1992

Biography

Dr. BRUCE SEGEE received a PhD in Engineering from the University of New Hampshire in 1992. He has been an assistant professor of electrical and computer engineering at the University of Maine since that time. At the University of Maine he heads the Instrumentation Research Laboratory, an organization dedicated to research and teaching involving instrumentation and automation. Work in the lab includes the use of PC's, PLC's, and embedded controllers for instrumentation, automation, and networking. Work also includes the use of fuzzy logic and artificial neural networks.

VINCENT M. ALLEN has recently received a Bachelor of Science degree in Electrical Engineering at the University of Maine. Currently he is working in the Instrumentation Lab at the University of Maine toward a Master of Science degree in Computer Engineering.