Automated Sorting Conveyor

Joshua Ashley | July 2025

Software: DirectSoft5

PLC/IO Modules

- DirectLOGIC CPU D2-250-1
- D2-16ND3-2 DC Input
- D2-12TR Relay Output

Inputs

- Photoeyes-LTR-M18PA-PMS-603
- Tri-Tronics UD-A Proximity Switch
- NO-PB/NC-PB Control Station

Outputs

- 24VDC Motor
- SMC VZ5000 Series Solenoid Valve

Project Overview/Design Constraints & Decisions

This isn't a perfect replication of an industrial system, but that was not the goal. The focus was on reliable signal interpretation, accurate and dynamic queueing, and consistent logic execution based on that queue.

This system was built with limited hardware and I/O, and a few concessions were necessary to keep it functional. The control station was compact, and due to space constraints, some components had to be mounted in ways that weren't quite ideal. For example, the proximity sensor couldn't be aligned directly with the solenoid valve, so the valve activates on a short delay relative to the sensor signal. Additionally, while rejected parts don't always land perfectly in the rejection bin, they do consistently leave the conveyor, which was sufficient for the intended logic.

The full program will be provided in the files for examination of how the system operates "under the hood". It may seem unnecessary to have the system operable in multiple sorting modes, but it was requested by a peer and was viewed as another design challenge to meet. Additionally, the I/O is not masked with internal bits. While this wouldn't be ideal in a more complex system, it's acceptable here as this is a relatively small system.

System Behavior

This system uses a pair of photoeyes to detect and ID parts based on height, stores part IDs in a FIFO queue, and uses those values to control rejection at a sorting zone.

The system runs in three different modes. In all 3 modes the conveyor will run continuously, however sorting logic varies between modes. Modes are as follows:

- **Bypass mode** - All sorting logic disabled.
- **Auto Sort Tall** - Parts categorized as “tall” will be rejected at the sorting zone.
- **Auto Sort Short** - Parts categorized as “short” will be rejected at the sorting zone.

In this system, “**Short” part ID = 1 | “Tall” part ID = 2**

In either sorting mode, parts are categorized and queued at the **loading zone** using two photoeyes. As parts travel down the conveyor, a proximity switch detects their arrival at the **sorting zone**. If the part meets rejection criteria, the solenoid valve actuates and diverts it with a blast of compressed air. If the part does not meet rejection criteria, it will continue down the conveyor.

Whenever a part is rejected or a part is allowed to bypass rejection, it will be deemed sorted.

Logic Design

Signal Confirmation & Part ID Logic

To prevent parts from being miscategorized due to electrical noise or physical movement, such as rocking or bouncing on the conveyor, a few filtering and validation techniques were implemented.

All sensor signals are debounced with a short timer of approximately 100 ms. Additionally, to avoid misclassifying a tall part as short, the system checks that the tall sensor was never triggered during the entire detection window of the short sensor before confirming the part as short.

Due to the physical build of the system, it is extremely unlikely for a short part to be mislabeled as tall, so this is not addressed in the code.

Queueing Logic
Because DirectSOFT5 provides no built-in FIFO instructions, the queue had to be built manually using indirect addressing. There are a few parts to this queue that make it work properly.

Read Address: This is a static address in memory that holds the oldest data of the queue. It's used in the sorting logic as the first/oldest part loaded into the queue will be the first that needs sorting(FIFO).

Up/Down Counter: This counter keeps track of the number of items currently in the queue. Its accumulated value is used as an offset, it is added to the Read Address to determine where new data needs to be written.

Write Address: This address is calculated based on the Read Address and counter value. This is where new data is stored in the queue. As objects enter or leave the queue, this address will change.

For proper part tracking, it's critical that new data is placed at the correct address in the queue. New data must never override existing data, and there must be no space between valid entries in the queue. Either of these occurrences will cause issues with missorting.

As parts enter the queue, the counter increments, and as parts are sorted and leave the queue, the counter decrements. Since the Read Address is fixed, using the counter value as an offset always gives the proper Write Address.

Lastly, as parts are sorted, the current value at the Read Address is discarded and every entry after that is shifted one address closer towards the Read Address.

Example Queue Operation

Here is a breakdown of how the queue would work in action. In this system, the Read Address is v5000.

A part is loaded onto the conveyor.
The photoeyes ID it as tall. The Write Address will be (v5000 + 0 = v5000). A value of 2 is entered at v5000 and the counter increments to 1.

Next, a second part is confirmed as short. The Write Address will be (v5000 + 1 = v5001). A value of 1 is entered at v5001 and the counter increments to 2.

Next, a third part is confirmed as short. The Write Address will be (v5000 + 2 = v5002). A value of 1 is entered at v5002 and the counter increments to 3.

This is what the queue looks like:
v5000 : 2
v5001 : 1
v5002 : 1
Counter accumulated value : 3.

Now, the very first part reaches the sorting zone. The sorting logic will examine the value in v5000. It will determine if this part needs to be rejected or not. Regardless of the decision, this part is now sorted.

Even though a part has been sorted, a few things need to happen to keep the queue in proper order. This is all handled in the logic to happen immediately after sorting. The counter will decrement, the oldest value in the queue will be discarded, and all entries in the queue will shift towards v5000 by one address.

This is what the new queue will look like:
v5000 : 1
v5001 : 1
v5002 : 0
Counter accumulated value : 2.

The queue has now been maintained and appropriately reflects what is on the conveyor.

Conclusions

Many conclusions can be made both about building a system and building the program for a system. It’s a good idea to have a well thought plan, but staying flexible in one’s ideas helps make different problems have less of an impact. Things that work on paper sometimes are not able to be implemented just due to real world constraints.

Testing components individually before integrating them into the system is critical, as it allows faults to be identified while still isolated. This makes troubleshooting much more effective. It also helps develop a better understanding of the signals produced by each sensor.

For instance, the proximity sensor was configured to send a LOW signal when detecting an object, while the photoeyes sent a HIGH signal. Knowing this upfront helped streamline the program development.

Additionally, a leaking valve manifold could have caused issues with part rejection, but this was caught and fixed prior to running the system. To continue, testing the detection ranges of the photoeyes and proximity sensor before mounting helped determine placement to ensure reliable operation.

Working in an unfamiliar programming environment has its advantages. It allows you to see how familiar concepts, like instruction sets or memory, are implemented differently. This can broaden one’s understanding and build flexibility.

For example, some environments include a built-in FIFO instruction. In those that don’t, alternative solutions like shift registers or manually built stacks can be used. In this project, a manual stack was used instead of a shift register because shift registers are generally used for discrete values, while a stack can store full words. This was a more appropriate choice because while the system only processes two part types, if it needed to process more than two, it’d be an adjustment that would take minimal effort to make.

Lastly, when dealing with I/O modules, sensors, instructions or anything that isn’t readily understood, it's crucial to lean on available resources. Operator manuals, datasheets and technical support documentation are often overlooked, but they provide reliable answers. Issues can often be due to user error, not system error, and reading first can save lots of confusion.