High Speed Labeling using the PTi-210

Objective

Develop an application solution to accurately and repeatably place labels on moving products at high cycle-rates.

Overview

The PTi-210 module can be easily configured to perform high-speed labeling applications. Essential features used in the PTi-210 are the Capture Object, the Queue Object, and Program Multitasking. Each of these features will be discussed individually, and then used to create a fully functional labeling application. After a description of each of the core components, a section is provided to show the application in a graphical or flow chart form. From this flow chart we will develop the actual program code. A detailed explanation of the code will follow each program to facilitate understanding of the instructions. See the figure below for an example of an “in- line” labeling machine.

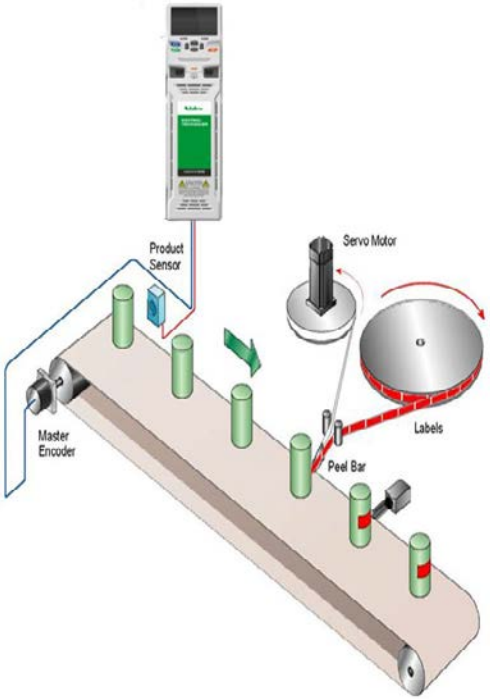


Figure 1

Capture Object

The Capture Object allows the PTi-210 to accurately capture the position of the master encoder at the exact time a product passes the incoming product sensor. To do this, the product sensor must be assigned to any one of the digital inputs located on the PTi-210 module itself (NOT to the M700 inputs or SI-I/O inputs).

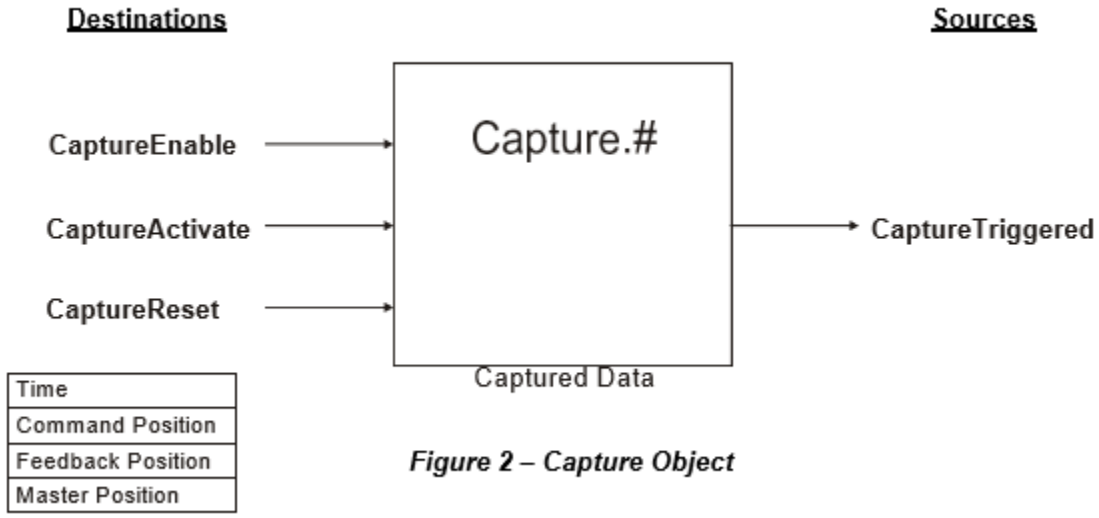


Figure 2 – Capture Object

The sources and destinations associated with the capture object can either be accessed through the Assignments screen or through a user program. This application will access the capture sources and destinations using both of these methods to operate the capture object.

In order to capture the position of the master encoder when a product passes the sensor, the PTi-210 module digital input that the product sensor is wired to must be assigned to the CaptureActivate destination found on the PowerTools Studio Assignments screen. Figure 8 shows all of the assignments used in the application. A User Program will deal with enabling and resetting the capture object.

When the capture object is enabled, the first rising-edge of the CaptureActivate signal will engage the capture, and the Time, Motor Command Position, Motor Feedback Position, and the Master Position will be captured in less than 2 μsec. Once the capture has taken place, the CaptureTriggered function will turn on indicating that data has been captured and it is available for use in the application. CaptureReset is then activated to prepare the system for the next capture, at which time the CaptureTriggered signal turns off automatically.

Below is a timing diagram that details how the functions associated with the capture object operate.

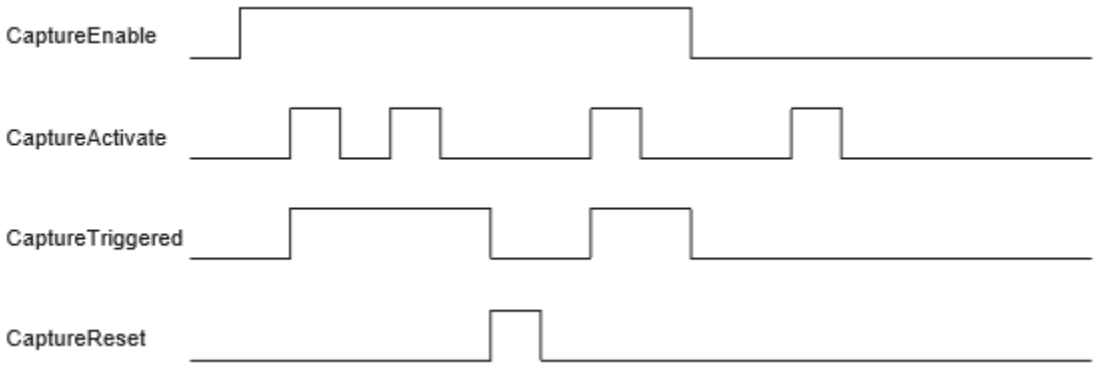


Figure 3 – High-Speed Capture Timing Diagram

Queue Object

The Queue is used in applications where multiple products exist between the incoming product sensor and the location where the process takes place (i.e. applying labels, bar code printing, vision inspection, part rejection, etc.). Up to eight Queue objects can be used simultaneously to control all of the processes in your application. Below is a diagram of the Queue Object.

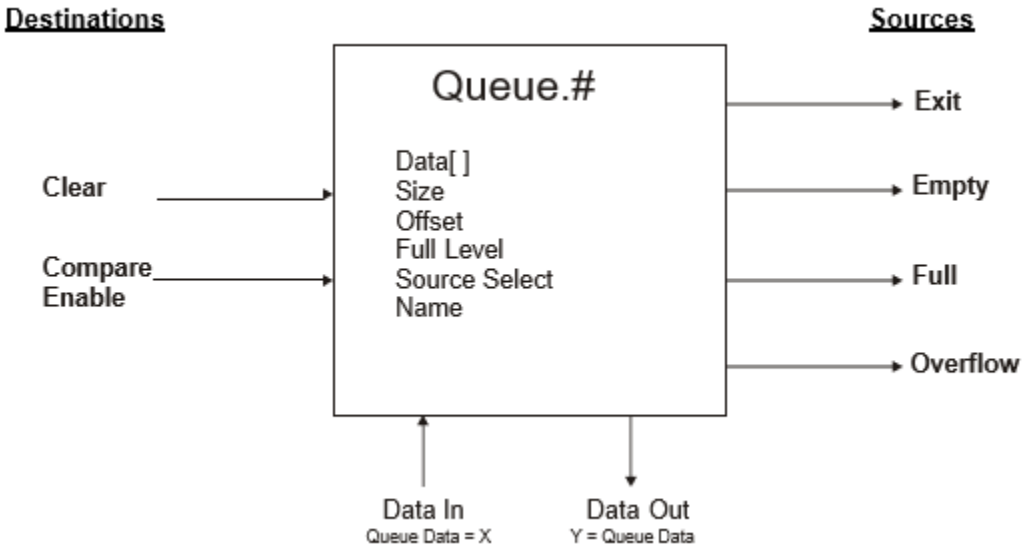


Figure 4 – Queue Object

The Queue object is used to store the captured master positions for each product that comes by the product sensor. Similar to the Capture Object, the Queue sources and destinations can be controlled through the assignments screen or in the User Program.

Note: Because the Queue uses captured data to initiate the index to apply the label, the product must not move relative to the master encoder once it has passed the incoming product sensor. If the product does move with respect to the master encoder, the label will be applied at the wrong time (or wrong position).

The following parameters must be entered into the Queue Setup Screen:

Name – Assign a name to each specific queue

Queue Size - This is the maximum number of products that can be stored in the Queue at a time. If the number of products entered into the queue reaches this value, the QueueOvervflow flag will activate. This value should be greater than the maximum number of products that can possibly be between the product sensor and the label head.

Queue Offset - The Queue Offset is a value that is added to the data put into the queue (captured position of master encoder in this example). The sum of the two values is called the QueueExitPosn. When the Source parameter is equal to the QueueExitPosn, the QueueExit function will activate. The QueueExit is what is used to initiate the index to apply the label. For this application, the Queue Offset is the measured distance between the product sensor and the label application point.

Full Level - This is a flag that notifies the user that a certain number of products are in the queue. Use this flag to notify the user that the queue is running at a certain “capacity”.

Source - The Source determines which parameter to compare to the QueueExitPosn to activate the QueueExit function. If set to FeedbackPosn, the QueueExitPosn is compared to the Motor Position Feedback parameter. If set to MasterPosn , then QueueExitPosn is compared to the Master Feedback Position parameter, and if set to CommandPosn, then QueueExitPosn is compared to the Motor Commanded Position. When the Source and the QueueExitPosn are equal, the QueueExit function activates.

Below is an example of the Queue Setup screen. The example uses two queues. The first queue controls the label application, and the second controls a deadband on the product sensor.

The label queue has a QueueSize of 12, indicating that there should never be more than 12 products between the product sensor and the label head. The QueueOffset is set to 10.0000 telling us that the distance between the sensor and the label head is 10 master units.

The deadband queue shows that there should never be more than 2 products in the queue, and that the QueueExit function will turn on 0.25 master units after the product passes the sensor. The deadband queue will be discussed further later in this application tool.

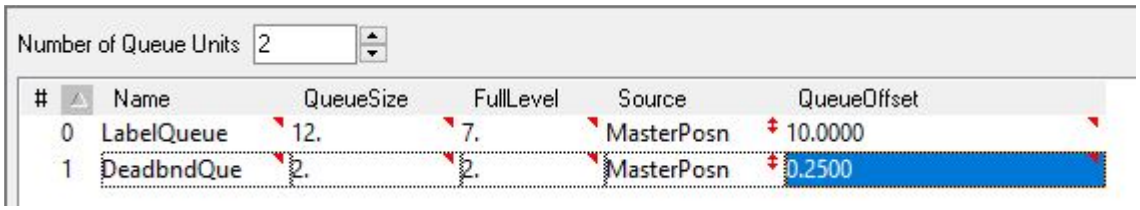


Figure 5 – Example Queuing Setup Screen

Following is a detailed view of the internal workings of the queue object. It shows how each of the queue parameters is used and how each source and destination functions.

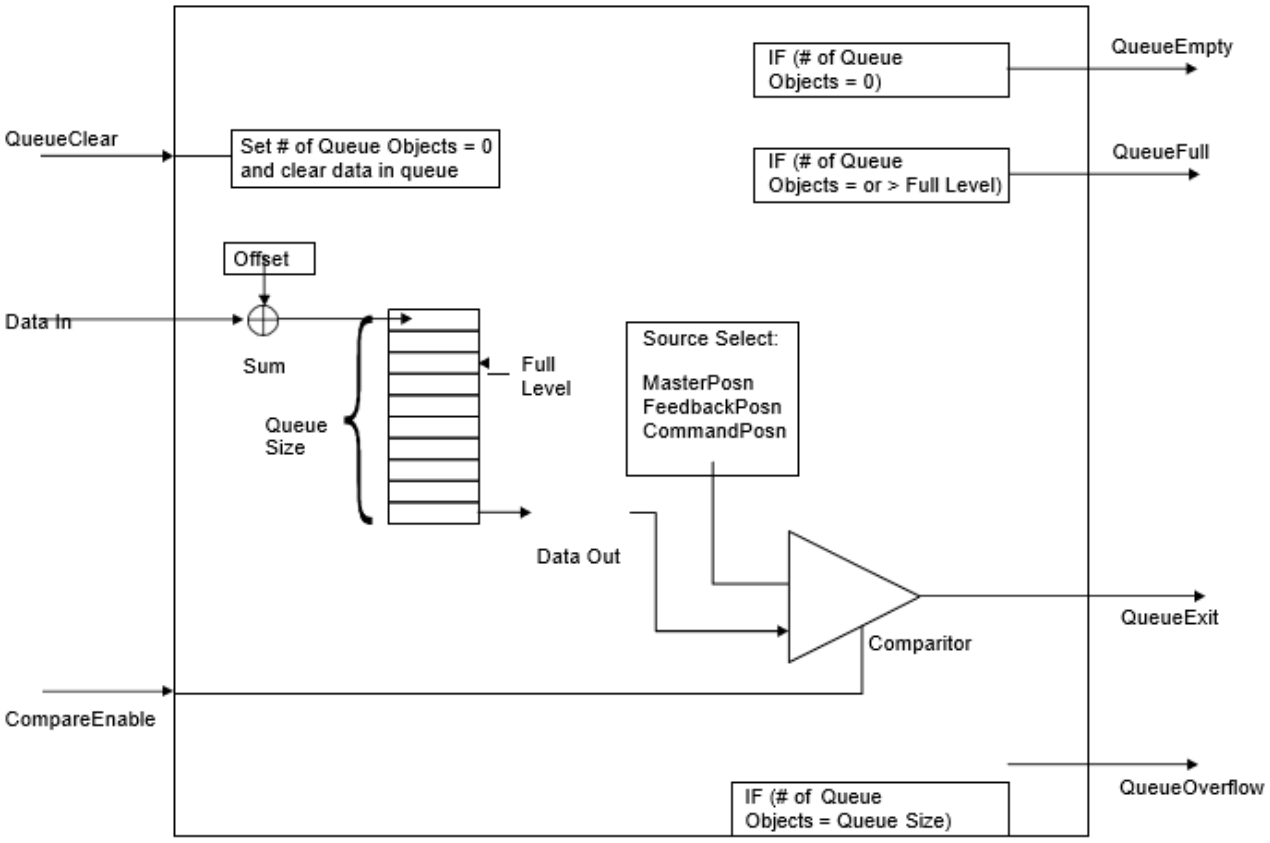
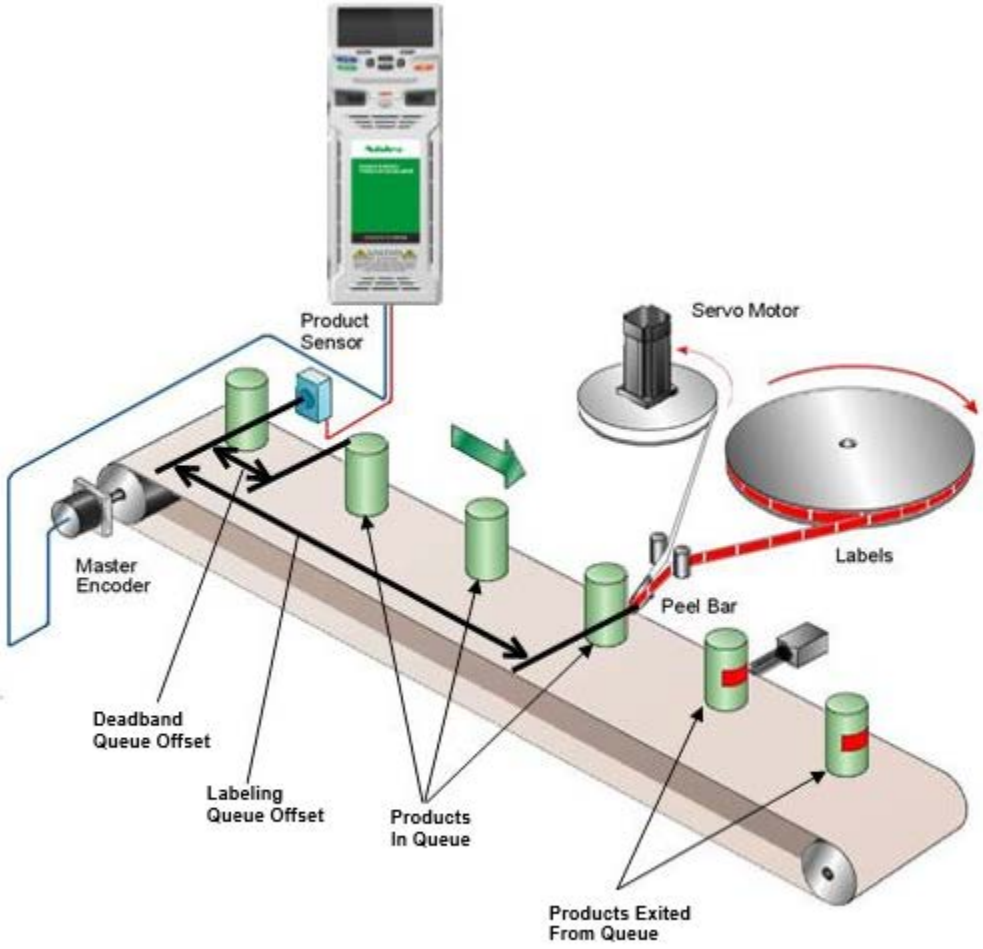


Figure 6 – Internal Functionality of Queue Object

Graphic with key dimensions and details



Assignments Used

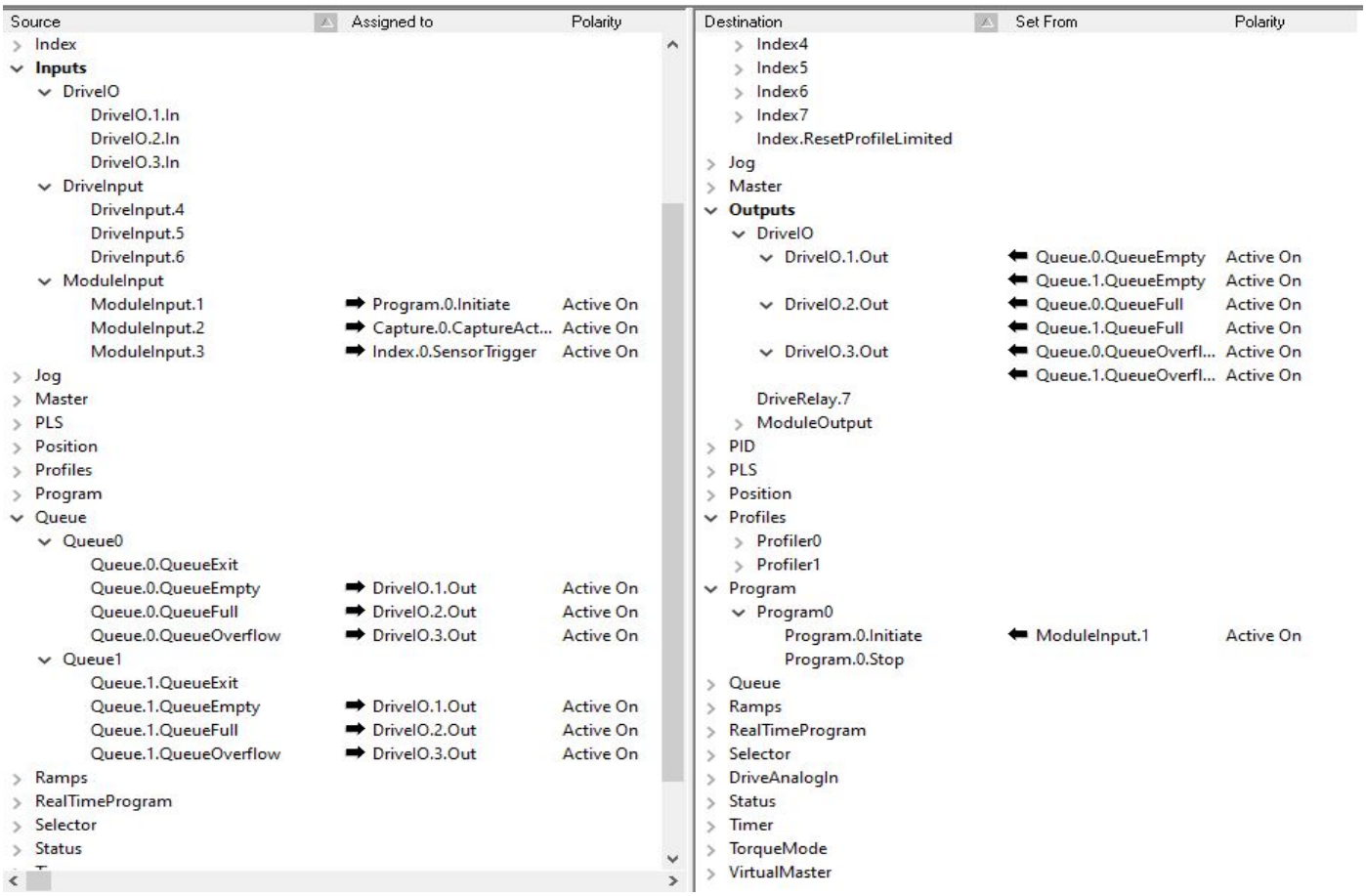


Figure 8 – Assignments Screen

The most important assignment used in this application is the ModuleInput.2 to Capture.0.CaptureActivate assignment. In this example, the product sensor is wired to ModuleInput.2. The sensor could be wired to any Module input, but NOT to a drive input. The drive digital inputs cannot perform the high-speed position capture the same way that the module inputs can.

ModuleInput.1 is used to start Program0 which in-turn starts the other two programs.

The Queue status bits are assigned to the Drive and Module Outputs to give the user some indication of how the queue is functioning. The example application has chosen to assign the QueueEmpty from both queues to DriveIO.1.Out (the first drive digital output), QueueFull from both queues to DriveIO.2.Out, and the QueueOverflow from both queues to DriveIO.3.Out. The user could choose to assign each queue status bit to a different output, or not to assign them at all. These assignments are not vital to the operation of the application.

NOTE: It is optional to the application, but most high-speed labeling applications use another sensor on the label material (not shown in the application graphic in Figure 1) to sense the leading edge of the next label (or the gap between labels). This sensor should be wired to ModuleInput.3 which is assigned to the Index.0.SensorTrigger destination. The user can then adjust the Registration Offset of Index 0 to control the amount of label "Hang Off" or "Hang Out" for the system. "Hang Off" is simply the distance that the label hangs out past the peel plate when the labeling index is complete.

Graphical Interpretation

The following page contains a graphical depiction of how the user programs will function, and how they work together. Each of the boxes depicts a user program. The diagrams inside the box show the process performed by the specific program. Several lines then flow from one "program" box to the next. This indicates that some data or signals are shared between those two programs.

The Program 0 box shows how the product sensor is used to activate Capture 0. When CaptureTriggered turns on, the Capture.0.CapturedMasterPosition is loaded into Queue0 and Queue1. When the Queue.1.QueueExit function activates, Capture 0 is reset and the process is repeated again.

The Program 1 box then shows how the Queue.0.ExitPosition is loaded into Capture 1. Index 0 is then initiated using the Capture 1 data. This process then happens all over again. It is important to recognize that Program 0 and Program 1 operate simultaneously allowing product to be sensed and captured in Program 0 while labels are being applied in Program 1 at the same time.

The Program 2 box does not use any data or signals from the other two programs and therefore runs completely independently. The program simply checks for either the increment or decrement inputs being on, and then checks if the Coarse Adjust input is on. Then the specified amount is added to or subtracted from the Queue.0.Offset parameter.

Following the flow chart, each user program is written and discussed in detail.

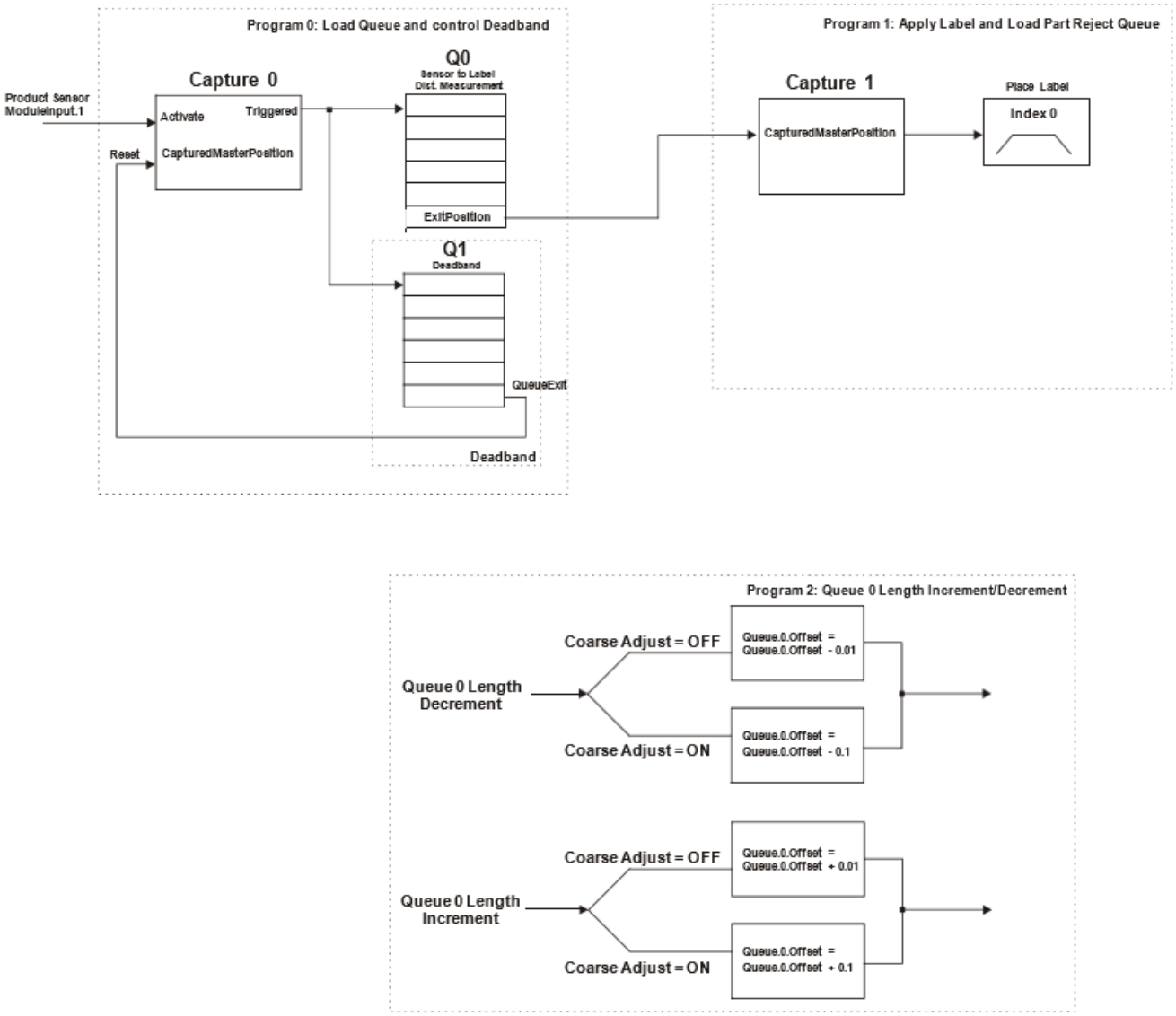


Figure 9 – Labeling Flow Chart

Program 0 – Actual Program Code

```
'------------------------------------------------------------------------

'Program 0 (running on Task 0)                                          -

'Description: High Speed Labeling Application                           -

'Filename(s): High Speed Labeler                                        - '---------------------
----------------------------------------------


Capture.0.CaptureEnable=
ON
Capture.0.CaptureClear
Queue.0.QueueClear=ON
Queue.1.QueueClear=ON

Wait For Queue.0.QueueEmpty
Queue.0.QueueCompareEnable=
ON
Queue.1.QueueCompareEnable=
ON

Home.0.Initiate

Wait For
Home.0.CommandComplete
Program.1.Initiate
Program.2.Initiate

Do While TRUE

    Wait For Capture.0.CaptureTriggered
    Queue.0.DataIn=Capture.0.CapturedMasterPositi
    on
    Queue.1.DataIn=Capture.0.CapturedMasterPositi
    on Wait For Queue.1.QueueExit

    Queue.1.QueueRemove
    Capture.0.CaptureCle
    ar

Loop
```

Description of Program Code

Program 0 loads captured data into the labeling queue and the deadband queue. The first grouping of instructions simply prepares the Capture and Queue objects to start the process. The Capture is enabled and cleared so that the next product to pass the sensor will generate a position capture. The Queue is then cleared to remove any residual data that may be in the queue, and the comparator internal to the queue is enabled so that QueueExit functions will be activated.

The Home function is then initiated, and the program will wait until the home is complete before starting Programs 1 and 2, and then entering the main loop. Note that this program does not “Call” Programs 1 and 2, but rather starts them. Since Programs 1 and 2 are assigned to different tasks, they will run simultaneously with Program 0. (For more information on multitasking, see Application Tool AT-5)

The main loop will run until the drive is disabled, or a Stop function is activated through an assignment. The first product to pass the sensor will generate a position capture, and the CaptureTriggered function will activate.

When the CaptureTriggered turns on, the CapturedMasterPosition will be loaded into Queues 0 and 1 using the “DataIn” command. The program then waits for the Queue.1.QueueExit function to activate providing the deadband. When the Queue.1.QueueOffset distance has passed on the master encoder, the Queue.1.QueueExit function will turn on. The program then removes the last piece of data from Queue 1 and resets the capture object so that the next product to pass by the sensor will again generate a position capture.

The purpose of the deadband queue is to prevent “false triggering” on the product sensor. If the sensor signal bounces, false data can be entered into the queue, and labels could be dispensed at the wrong time. Because the deadband queue exit signal resets the position capture object, adjusting the deadband queue offset will adjust the master distance that must pass before another product can be entered into the queue.

Note: The deadband queue offset should be longer than the product length, and shorter than the product length plus the minimum product spacing.

Program 1 – Actual Program Code

```
'--------------------------------------------------------------------

'Program 0 (running on Task 0)                                      -

'Description: High Speed Labeling Application                       -

'Filename(s): High Speed Labeler.                                   - '---------------------
-----------------------------------------------

Do While TRUE

    Wait For Not Queue.0.QueueEmpty
    Capture.1.CapturedMasterPosition=Queue.0.ExitPositi
    on Index.0.Initiate Using Capture.1

    Wait For
    Queue.0.QueueExit
    Queue.0.QueueRemove

Loop

End
```

Description of Program Code

Program 1 handles the process of applying the labels to the product. This program is initiated by Program 0 and runs in unison with Programs 0 and 2. The program will run until the drive is disabled or a stop function is activated.

The first instruction waits until some data is entered into the queue (until the first product passes the sensor). Once some data has been entered into the queue, the Queue.0.ExitPosition is loaded into Capture 1. The Queue.0.ExitPostion is the sum of the captured master position when the product passed the sensor plus the Queue.0.QueueOffset distance. The Queue.0.ExitPosition is loaded into the Capture 1 object so that the ExitPosition is used as the exact starting point for the index. In order to guarantee that the index starts at the right time (therefore ensuring accurate label placement), the "Using Capture" instruction is inserted after the Index Initiate. By using the "Using Capture.1" instruction, the index can automatically adjust its profile to compensate for any delay in getting started.

Index 0 is a synchronized index so that the index velocity is referenced to the master encoder velocity and position. This allows the label to be dispensed at the proper velocity regardless of how fast the product is moving.

Once the Index is initiated, the motor waits until the master axis position reaches the Queue.0.ExitPostion and then the index runs therefore applying the label. The program waits for the Queue.0.QueueExit function to activate meaning that the product has passed the label head, and then discards the data related to the passed bottle using the Queue.0.QueueRemove instruction.

Once the data for the labeled bottle is removed, program flow proceeds to the top of the loop and wait for the next product. In most situations, there will already be more products in the queue, so immediately the next ExitPosition is loaded into the Capture 1 object and the next index is initiated.

By using separate tasks to load the queue and apply the labels, products can be entered into the queue and apply labels at the same time. This is the advantage of the PTi-210 multitasking controller.

Program 2 – Actual Program Code

```
'--------------------------------------------------------------------
'Program 0 (running on Task 0)                                      -
'Description: High Speed Labeling Application                       -
'Filename(s): High Speed Labeler.                                   -
'--------------------------------------------------------------------
```

```
 Do While TRUE
      Wait For (DriveInput.4=ON OR DriveInput.5=ON)
      If (DriveInput.4=ON) Then
            If (DriveInput.6=ON) Then
                  Queue.0.QueueOffset=Queue.0.QueueOffset + 0.10
            Else
                  Queue.0.QueueOffset=Queue.0.QueueOffset + 0.01
            Endif
      Else
            If (DriveInput.6=ON) Then
                  Queue.0.QueueOffset=Queue.0.QueueOffset - 0.10
            Else
                  Queue.0.QueueOffset=Queue.0.QueueOffset - 0.01
            Endif
      Endif
      Wait For (DriveInput.4=OFF AND DriveInput.5=OFF)
Loop

End
```

Description of Program Code

If the machine does not include an HMI or other device to modify system parameters, a task can be created to Increment/Decrement the Queue Offset parameter using digital I/O.

Program 2 is used to adjust the Queue.0.QueueOffset parameter. The QueueOffset determines when the label will be applied to the product. If this value is not accurate, the label can be applied early or late which correlates to a misplaced label.

The QueueOffset for the labeling queue is simply the distance between the product sensor and the label head. On some machines, the product sensor may get bumped or moved. If this happens, the user typically does not want to use the PC to upload the PTi-210 program and adjust the parameter accordingly. Therefore, a program is created to use digital inputs to adjust the QueueOffset distance.

In this example, DriveInput.4 is used to increment the QueueOffset, DriveInput.5 to decrement the QueueOffset, and DriveInput.6 is a “Coarse Adjust” selector to magnify the amount of increment or decrement.

The program simply waits for either the increment or decrement inputs to be activated. It then checks to see if the increment input was used (DriveInput.4). If so, the program checks to see if the coarse adjust input is active (DriveInput.6). If coarse adjust is active, then 0.10 is added to the QueueOffset, otherwise 0.01 is added. If the increment input was not active, then the program checks again for the coarse adjust input and either subtracts

0.10 or 0.01 from the QueueOffset.

Notice how the program waits until the inputs 4 and 5 are both off, therefore guaranteeing that the QueueOffset is incremented or decremented only once each time the input is activated.