Automated Vision-Based Inspection System for Stents

A Major Qualifying Project
Submitted to the faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

Jason Farmer

Benjamin Mar

Erik Schmidtberg

Submitted to:
Instructors and Advisors (WPI)

Prof. John Orr (Project Advisor & Center Co-Director)

Prof. David Finkel (Center Co-Director)

Date: April 26, 2005
mqpstent@wpi.edu

Executive Summary

The purpose of this project was to address the challenge of designing and implementing a vision system to find defects in coated stents. A stent is a small medical device that is used to prevent blood vessels and other hollow passageways in the body from closing. These drug-eluting stents are coated with special drugs to prevent excess blood clotting and restenosis or re-narrowing due to the presence of the stent within the vessel. Defects in the stent geometry or stent coating can occur during manufacturing and defective stents must be repaired or removed before a product batch can be shipped. This project used a Guidant designed machine vision console to streamline the stent inspection process by automatically inspecting the stents for defects.

The main requirement for the automated inspection system was that it needed to identify all defects on a given stent and reject the stent if the defect size was above a known threshold. It was also preferred that the inspection examine the stent at a rate of 18 mm per 1 minute and identify less than 20% falsely identified defects. Priority was given to catching all of the true defects, as a missed defect could result in a defective stent entering the market.

The first step in creating this inspection system was to identify all of the defect types. Identifying the potential defects that the system would need to search for was the simplest of the tasks faced while completing the project. Guidant Corporation maintains a list of all defects that occur during the production of stents. Strict definitions for each defect type and whether it is “Acceptable” or “Unacceptable” have been developed previously by Guidant. After the list had been obtained, it was then condensed to include only the surface and coating defects that the inspection would be searching for.

After the potential defects had been identified, it was possible to develop the inspection system. The development of the inspection system consisted of three major challenges. These challenges consisted of maneuvering the stent using the CNC fixture on the machine vision console, inspecting the stent using the camera and software on the console, and designing a user interface to interact with the inspection operator.

To begin the inspection, the stent first needed to be loaded into the machine vision console and maneuvered into the inspection area by a CNC fixture. The fixture then rotated and horizontally moved the stent so that all geometrical features on the stent could be inspected. This required the use of G-code and the program NViewMMI to control the CNC fixture. While the fixture was manipulating the stent, the vision camera was used to image the stent and search for a predefined starting point. To do this, the DVT program FrameWork was used to define an inspection sensor package and inspection script. The goal was to automate most of the load/unload process. The operator would only need to place the stent on a mandrel and then load the mandrel into the chuck on the fixture. A mandrel is used to suspend the stent between the two chuck grips of the mechanical fixture. It runs through the hollow center of the stent and allows the fixture to manipulate the stent without damaging it.

After the algorithm to manipulate and properly align the stent for inspection had been created, the next step was to design sensors to scan the stent for defects. To do this, two sensor packages were created. The first sensor package was used to scan the strut and edges of the stent. It was called the Strut Sensor Package. The other sensor package, called the Black Space Sensor Package, was used to scan the area between the stent struts. By combining these two sensor packages, the inspection was guaranteed full coverage of the stent's geometry.

To manage the inspection system, a user interface was created using G-Code. This user interface allows the operator to start/abort the inspection, view an image of any stent features that failed inspection, accept/reject the feature, and classify the defect by type. It also generated a summary report that contained the number and types of the defects found.

After the automated inspection system had been created, it needed to be tested. The tests generated the experimental results that were used to analyze the accuracy, effectiveness, and efficiency of the system. The purpose of automating the inspection of stents was to create a system that could replace the manual inspection process that was being used. To do this, the test results had to demonstrate that the automated inspection process could provide an inspection that was as accurate or more accurate than the manual inspection. It also needed to prove that the inspection could run in a timely and efficient manner. After the tests were completed, it was found that the inspection identified 8 % false positives/acceptable defects, missed 0.08% true defects, and ran at a rate of 18 mm per 1 minute 6 seconds.

The results, while meeting the main objective of the project, show that it was very difficult to produce an inspection system that does not pass any defective stents, fail any acceptable stents without operator intervention, and inspects at a rate of 18 mm per 1 minute. Creating a sensor package that was sensitive enough to catch 100% of the defects resulted in the inspection time and the number of false positives increasing. A balance of these three objectives was achieved and the result of this project was an efficient yet accurate inspection system created for Guidant Corporation.

Further work and testing is required before this inspection system can be fully implemented. During the testing of the inspection, it was found that the system responds negatively to significant changes in the stent geometry or contamination caused by flakes from the Teflon mandrel coating. It also does not classify the defects that it identifies. Instead, it requires an operator to make the distinction at this phase. These issues, as well as the flexibility and robustness of the system will need to be addressed to further improve upon the inspection.

.

.

Table of Contents

Table of Figures

1. Introduction

A current business trend that exists in the market today is the shift from manual labor to automated production. Over the years, technology has evolved so quickly that there is now a surplus of technology awaiting proper utilization. Companies are now working to integrate technology into their design and manufacturing processes. Integrating technology into the manufacturing process reduces the cost of production and maximizes the company's efficiency. Companies that don't take advantage of the technology available to them will begin to lose their market share as other companies find newer, more efficient ways to manufacture their products.

The market for cardiovascular medical products has quickly become one of the most technological industries. Many of the products are very small in size and large in importance. These products are often used during surgery or other medical procedures and can be used to save people's lives. An example of these products is the stent.

Stents are inserted into arteries and veins to help prevent clogging and provide added support. Since they are used within the body, defective stents are unacceptable. The procedure for producing and inspecting stents must be as thorough as possible. This is an expensive process, so integrating technology becomes even more important to companies that produce stents, like Guidant Corporation.

Currently, Guidant inspects their stents by hand. This process is slow and expensive. Their goal over the next few years is to develop a process that will use the machine vision technology that exists in the technology market to create an automated inspection system. This project was conducted to develop this system for Guidant and was completed by working with an existing machine vision tools. The system checks the stents for potential defects, classifies any defects that are found, prompts an operator when a defect is found, and saves an image of any defective stent features. This inspection process and the user interface that was designed with it will help Guidant more efficiently produce and inspect their drug eluting stents.

2. Background Research

Guidant Corporation, which has a production facility in Silicon Valley, California, specializes in the production of cardiovascular medical devices. One of their main products is the drug-eluting stent. A stent is used to prevent blood vessels and other hollow passageways in the body from closing. It is inserted in an artery and then ballooned so that it supports the outer walls of the artery. Drug eluting stents are special because they are coated with a chemical that discourages clogging or clotting around the stent. Eluting means to purify or wash out, so drug-eluting stents use the chemicals that they coated with to elute or purify the area around the stent to prevent re-closure of the vessel. Due to the importance of the stent, each product needs to be thoroughly inspected before it can enter the market. To design a machine vision system to inspect these stents, background research was conducted on machine vision technology and stent production, along with sponsor-company and Silicon Valley history (found in Appendix A). This research has provided a solid understanding of the problem at hand and allowed the project to be most useful to Guidant.

2.1. Stent Technology

Stents are named after Charles Stent, an English dentist, who invented a support structure to align teeth. The stent was later adapted for use in a blood vessels and arteries and, in 1986, Jacques Puel and Ulrich Sigwart inserted the first stent into a human coronary artery. It wasn't until 1994 that the first Palmaz-Schatz stent was approved for use in the United States.

The stent can either be plain mesh or it can be covered with a synthetic tissue lining or drugs. These tubes are inserted into the esophagus, trachea, or blood vessels and their purpose is to keep these parts of the body open. Before the stent is inserted into the body, it is collapsed to a much smaller diameter. Once inside the desired vessel, it is expanded using a small inflatable balloon. The stress between the surface of the stent and the vessel is utilized to hold it in place. Figure 1 shows a stent and the procedure for inserting it into a blood vessel.

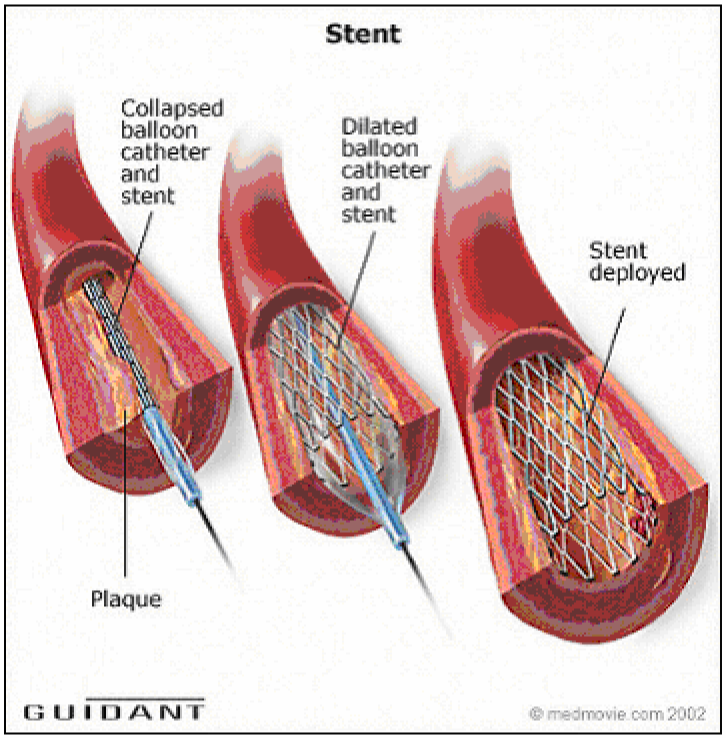

Stent
Collapsed balloon catheter and stent
Dilated balloon catheter and stent
Stent deployed
Plaque
GUIDANT
© medmovie.com 2002

Figure 1: Insertion of a Stent

2.1.1. Stent Usage

The main advantage of using stents is that the procedure to insert them is much less invasive than the equivalent surgical operation. When performing Coronary Angioplasty, which is a process to unblock a coronary artery, a wire is passed through the diseased artery, past the blockage. Then a balloon catheter is passed along the wire to the blockage and inflated to open up and stretch the walls of the artery. A stent can be connected to the balloon that will expand in the newly opened artery and help hold it open once the balloon is removed. Although this procedure has been proven to reduce symptoms due to coronary artery disease, and to reduce ischemia, it has not been proven to reduce mortality due to coronary artery disease.

Often times, the stent will cause small injuries in the blood vessel when it is being ballooned to its full size. These injuries, small scratches on the inside of the vessel are created purposely. They are created to encourage the body to heal the area around and under the stent. To heal this area, the body will cover the damaged surfaces within the artery with new, uninjured cells. This will create a fine layer between the stent and the inside of the vessel. With the stent thoroughly integrated into the vessel, it can then act as an internal support structure.

There are two major designs available for stents. The first of these is bare metal coronary stents. As the name would suggest, they are not coated with any chemicals. The major drawback with bare metal coronary stents is that they tend to trigger too severe a reaction within the body where too many cells are sent to heal the area around the stent. This severely increases the thickness of the vessel walls and reduces the diameter of the blood vessel. This process is called restenosis or re-narrowing of the blood vessel. To solve this problem, drug-eluting stents were introduced. The stents are coated with a fine layer of chemicals that will slow the healing process within the vessel. This reduces the thickness of the layer built by the healing cells and helps to prevent restenosis.

Bare metal coronary stents are simply a mechanical framework that holds the walls of the artery open, preventing the narrowing of such arteries. This technique is more effective than plain angioplasty. The two most effective and safe drugs that being used to coat newer stents are sirolimus and paclitaxel. In addition to these drugs, there are other drugs used to coat the stents that inhibit the growth of scar tissue after they are implanted into the arteries. A drug called Rapamycin is used to coat the stents that has a near-zero restenosis rate meaning there is no recurrent narrowing of the arteries. (Kulick, 6)

2.1.2. Stent Production

Due to their importance as cardiovascular medical tools, stents need to be thoroughly inspected before they can be shipped from the production facility. During the production of stents, there are many opportunities for defects to occur. Any of these defects could prove disastrous if they are not caught and fixed before the stent is used.

The defects that can occur during productions can range greatly in type and rate of occurrence. Many of these defects occur when the stent is coated with a solution that consists of a solvent, a polymer, and a small amount of a drug. Often times, the coating will form bridges or webs between supports on the stent, clumps on the surface of the stent, or thin cobwebs that extend over the surface of the stent. Also, the coating can be layered unevenly, causing bare spots to appear on the surface of the stent. These defects along with any chips or other structural defects must be caught during inspection. Examples of these defects are shown in Figures 2-3.

Please note these pictures and all others of defects that were used for this project were created specifically for this project and intended to produce defects for vision algorithm development. The stents did not contain drugs and are not representative of the quality of coated stents produced on the manufacturing lines.

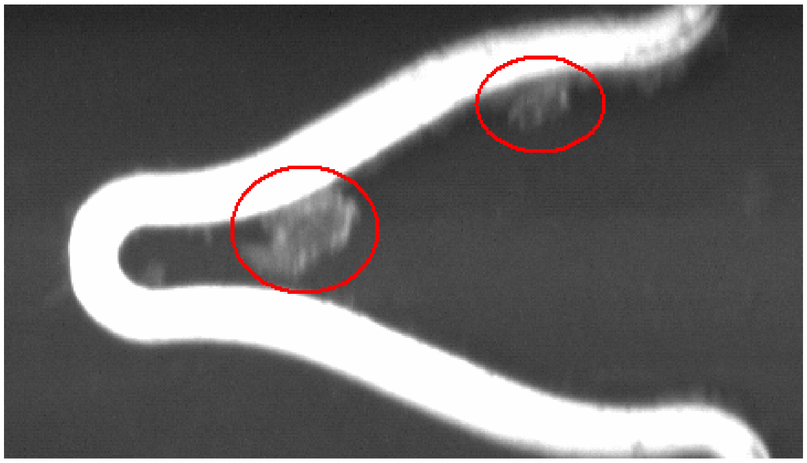


Figure 2: Example of Clumping

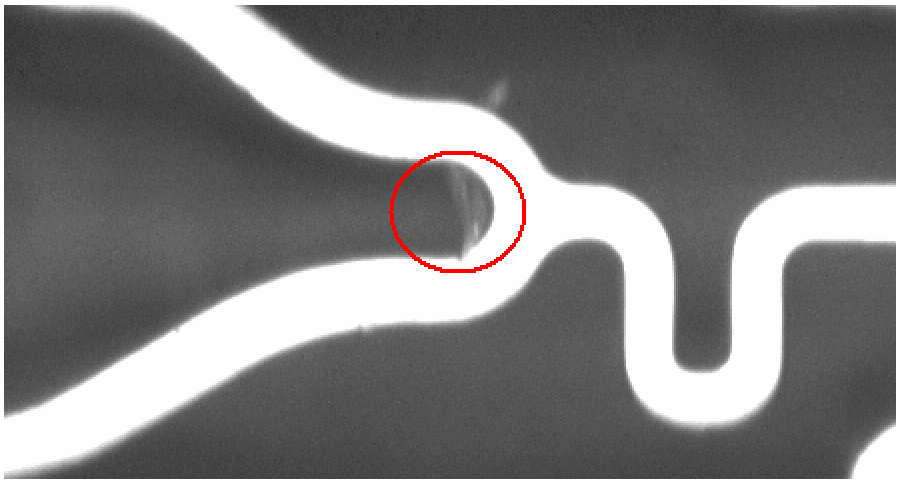


Figure 3: Example of Pool Webbing

Should any of the defects make it through inspection, they could prove costly to Guidant Corporation, the reputation of the doctor and hospital using them, and the patient who is receiving them. The danger lies in the fragile nature of the coating defects. Should any of the webs, clumps, or bridges break off of the stent after it is inserted in the body, they could flow through the vessel to other parts of the body. These pieces, though small, could get caught in the vessel and cause the vessel to clog. This danger makes the inspection of stents a priority during production.

2.2. Machine Vision

Machine vision tools are used to by companies to automate the inspection of their products. In the case of Guidant Corporation, they wish to use machine vision to automatically inspect their drug-eluting stents for coating and structural defects.

2.2.1. Machine Vision Basics

Machine vision is a term used for numerous vision tools working together. Often times, the different tools are arranged together on a console that maximizes the effectiveness of each part. Machine vision consoles consist of a number of cameras and sensors, various lighting fixtures, a mechanical fixture to manipulate the part being observed, and a computer to control and integrate all of the tools. The console is often designed with a central viewing area in mind. The mechanical fixture is programmed to place the product being observed within this specified area. The cameras, sensors, and lighting fixtures are then arranged above and to the sides of this area to best attain the desired exposure and lighting effect. An example of this setup is shown in Figure 4.

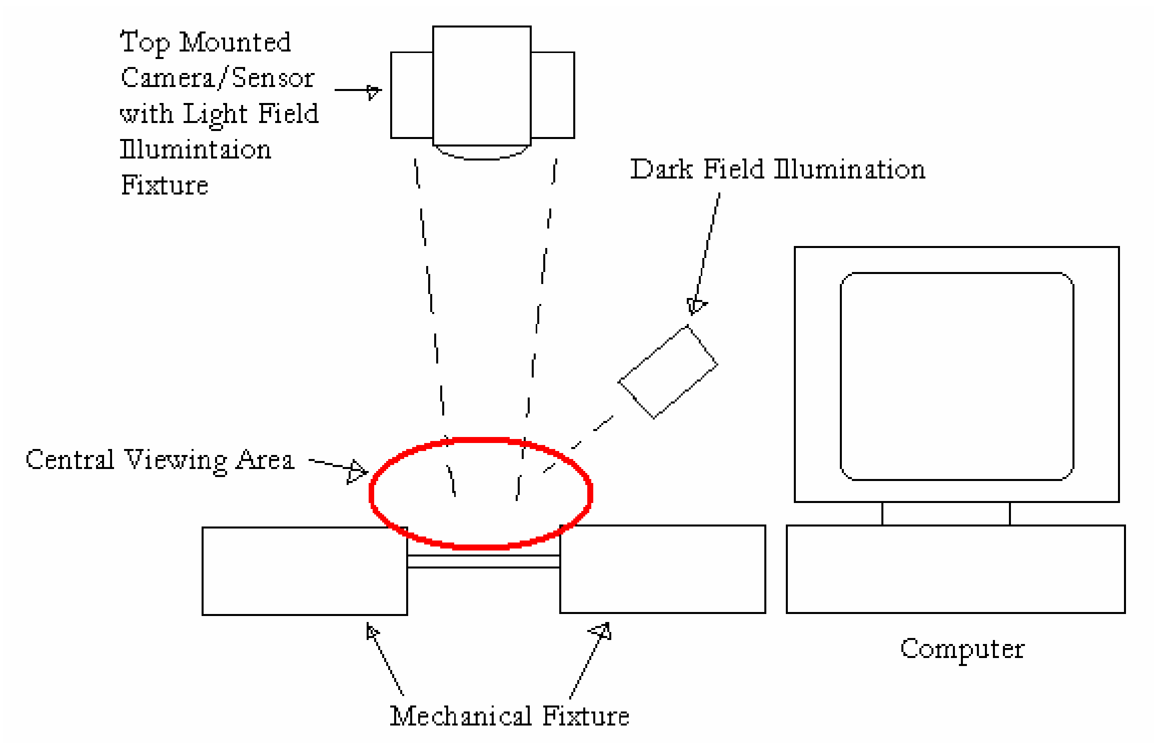


Figure 4: Sample Machine Vision Console

2.2.2. Machine Vision Options

To achieve a complete machine vision analysis inspection system there must be a harmony of software, sensor, lighting, and mounting of the product. Each part of the machine vision console needs to be arranged for maximum result and work in conjunction with the control software on the computer.

2.2.2.1. Lighting Options

The lighting options are the most complex and important in the machine vision inspection system. Many different types of lighting can be used to illuminate the product. These lighting options are chosen based on the surface of the product and desired resolution of the system. The machine vision console that Guidant Corporation had designed was flexible enough to allow for numerous lighting options. Based on this flexibility, we explored the lighting options to determine which would best fit our requirements.

The two most popular lighting options are light-field and dark-field illumination. Light field illumination is used best with non-reflective surfaces to provide high contrast pictures for inspection. Dark field illumination works well with reflective surfaces to easily detect scratches, packaging tears, and other surface flaws. Both lighting applications use a series of lights such as LED, laser, or halogen, and project it on the product being inspected. With light field illumination the light is directly pointed down, usually from a ring around the camera itself so that most of the light is reflected off the surface in question and back into the camera. Dark field uses the same lights, but at an angle such as 45 degrees inward towards the camera. This means that the light will reflect off of the surface under inspection and away from the camera unless a surface flaw causes the light to change direction and reflect into the camera. The lighting positions and reflections for dark and light field illumination are shown in Figure 5. It is also possible to use a combination of these two lighting methods to look for different features. Light field illumination could be used to easily verify that there is not Webbing in the coating, where dark field illumination could check the consistency and complete nature of the coating.

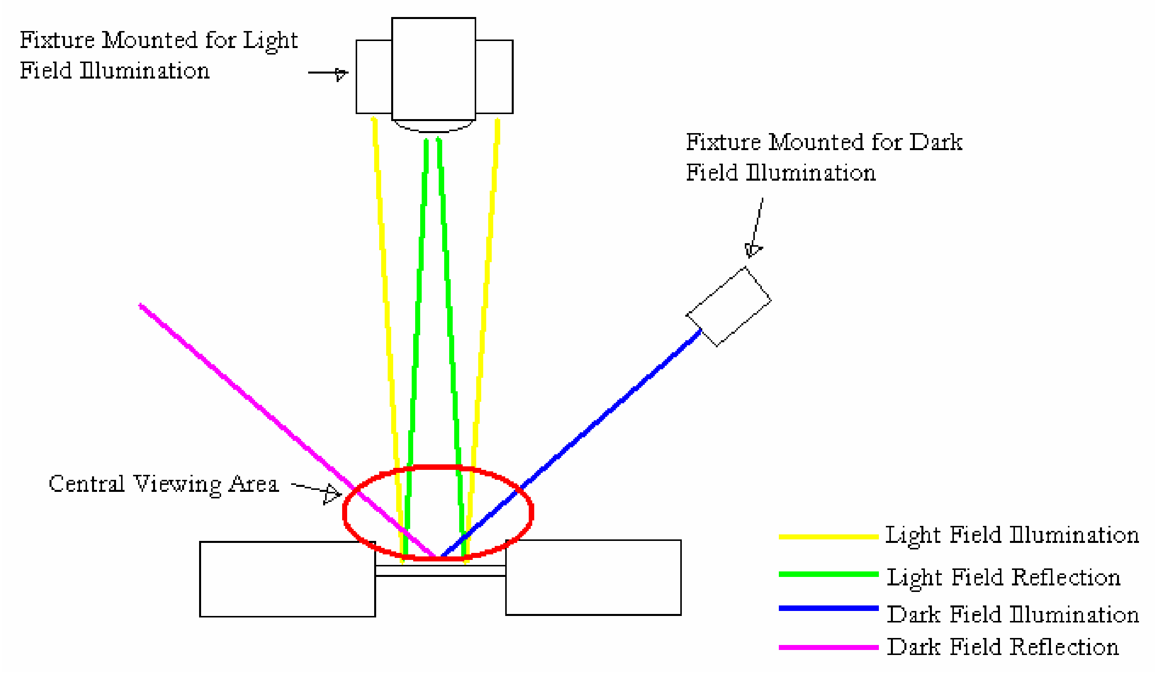


Figure 5: Light and Dark Field Illumination with Ideal Reflections

2.2.2.2. Camera and Sensor Options

Selecting a camera and sensor package is another important decision that the designer needs to make when creating an inspection system. There are currently numerous camera and sensor packages available on the market. Each package has its own strengths and weaknesses. For our project, the camera and sensor package was chosen for us. Guidant had already designed and built a machine vision console that used the DVT Legend 540 Series camera/sensor package.

The Legend 540 Series camera is one of DVT's most advanced models. It specializes in high-speed inspection and can inspect 60,000 parts per minute. Its electronic shutter allows it to image products at speeds ranging from 1 every 10 microseconds to 1 every second. The camera's Ethernet capability allows the camera to communicate with the control software and transfer the images onto a computer hard drive. This allows it to save the images of the stent that it inspects.

2.2.2.3. Software Options

Selecting the appropriate software package and designing the code for the system is another important step in designing a machine vision inspection system. Part of this problem has already been settled. Guidant named DVT Corporation, who is a leader in the machine vision world, their primary software provider.

This is another design decision that Guidant Corporation had already made. The console they had designed used DVT's FrameWork software to control the camera and sensor packages, NViewMMI to control the mechanical fixture, and Visual Basic to implement the user interface to the completed system. FrameWork was used to set up digital sensors through the computer and look for certain attributes on the stent. NViewMMI used G-code (G Code is the language of CNC controllers) to program the mechanical fixture.

2.2.2.4. Mechanical Fixture Options

The final component in this system is the mounting of the stent for inspection. Since a stent is a 3D object and must be inspected on all sides there will be requirements for CNC motion and automated insertion and removal from the inspection area. In most cases, this area of the project would be another important step in the design process. There are numerous choices that could be made regarding how to manipulate the stent. Fortunately, this is another design option that Guidant Corporation had made for us. They used a custom built mechanical fixture to move and rotate the stent in the central viewing area.

3. Inspection Procedure

The purpose of this project was to address the challenge of designing and implementing a vision system to find defects in coated stents. These drug-eluting stents are coated with special drugs to prevent excess blood clotting and restenosis due to the presence of the stent within the artery. As discussed above, defects occur during the manufacturing of stents and defective stents must be repaired or removed before a product batch can be shipped. This project used a Guidant designed machine vision console to streamline the stent inspection process by automatically inspecting the stents for defects

3.1. Product Specifications

By working with the project liaison, Mr. Jason Van Sciver, a set of product specifications for the design was developed. These specifications are listed below.

1. Identify potential coating or structural defects on the stent geometry and surface.
2. Classify any found defects using a list of known defects.
3. Prompt an operator when a defect is found and pause the inspection.
4. Allow an operator to classify any found defects as "Acceptable" or "Unacceptable."
5. Allow an operator to change the classification of any defects that were misclassified.
6. Display and save an image of any defects that are found.
7. Maintain a record of all defects found and their classification, preferably in the form of an MS Access Database.
8. Complete inspection of an 18 mm stent in less than one minute.
9. Allow 0% true negatives (actual defects) to pass through the inspection and identify less than 20% false positives (non-defects). Priority is given to 0% true negatives, as these cannot be caught by the system operator.

3.2. Design Flow Chart

Based on the product specifications defined above, a flow chart that displayed the basic ideas for how the process should work was created. This outline showed some of the key steps that were necessary to complete an automated inspection and was used to focus design ideas. It also assisted with defining the project objectives. The flow chart for the design outline is shown in Figure 6.

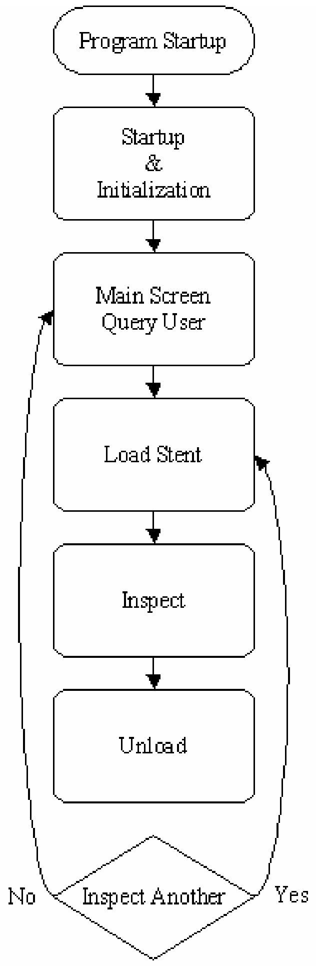


Figure 6: Design Flow Chart

3.3. Project Objectives

Using the above criteria, the project was divided into four major objectives. The project objectives are as follows:

1. Create a list of potential defects. This list included information regarding the severity and occurrence rate of each defect type.
2. Use G-code to program the motion control system that holds and rotates the stent mounted on a mandrel during inspection. This allows users to maneuver the stent into proper position for inspection.
3. Create an algorithm to identify the defects. This algorithm used a minimum number of strategically placed sensors to test for the wide range of defects and provide the best classification options while maintaining the stringent time requirements.
4. Use Visual Basic to create a manageable user interface for the system. This UI integrates the CNC and machine vision pieces, as well as prompting an operator to accept or reject any defective product it found.

3.4. *Identifying Potential Defects*

Identifying the potential defects that the system would need to search for was the simplest of the tasks faced while completing the project. Guidant Corporation maintains a list of all defects that occur during the production of stents. They already have strict definitions for each defect type and whether it is "Acceptable" or "Unacceptable." After the list had been obtained from the project liaison, Mr. Van Sciver, it was then condensed to include only surface and coating defects. The system did not need to search for structural defects because they are found and fixed before the stent is coated. The final potential defect list is shown in Figure 7. The "# of Occurrences" and "% of Occurrence" were generated using a sample batch from stent production.

This table was removed for confidentiality reasons. The table illustrated that the most common defects are: Pealing / Tearing Clumping Cob Webbing Contamination All others

Figure 7: Potential Defects List and Occurrence Rates

After the list of potential defects was obtained, several hours were spent in the laboratory using microscopes to view defective stents. The defect list was used during this time to classify the defects that were found and to assist with identifying the characteristics of each defect. Understanding how the manual inspection of stents is completed helped in designing a system that would replicate this process automatically.

3.5. *Manipulating the Stent*

The first step in during an automated stent inspection is to load the stent and maneuver it to the central viewing area. This required the use of G-code (explained in Appendix C) and the program NViewMMI to control the CNC fixture. While the fixture was manipulating the stent, the vision camera was used to image the stent and search for a predefined starting point. To do this, the DVT program FrameWork was used to define an inspection sensor package and inspection script. The goal was to automate most of the load/unload process. The operator would only need to place the stent on a mandrel and then load the mandrel into the chuck on the fixture. A mandrel is used to suspend the stent between the two chuck grips of the mechanical fixture. It runs through the hollow center of the stent as shown in Figure 8.

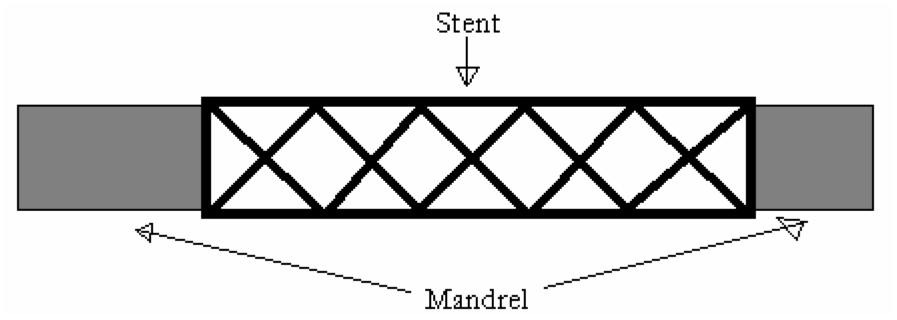


Figure 8: Stent and Mandrel Positioning

Once the stent and mandrel have been secured into the chuck of the mechanical fixture, the operator can press the “Start” button. The system then automatically closes the other side of the chuck, called the tailstock, and maneuvers the stent to the central viewing area.

Once in the central viewing area, the camera is focused on the beginning of the mandrel and the console shifts the mandrel slowly until a predefined starting point on the stent’s geometry is found. The stent and mandrel then begin to rotate. This is where the stent inspection software starts to run. The fixture control program continues to run during the inspection. It rotates the stent a full 360 degrees while the inspection program inspects and images the structure of the stent. The fixture is then shifted a predefined increment so that another ring of the stent can be inspected. This process continues until the stent has been fully inspected. The fixture then returns to the starting point and opens the tailstock. The flow chart for this process is shown in Figure 9.

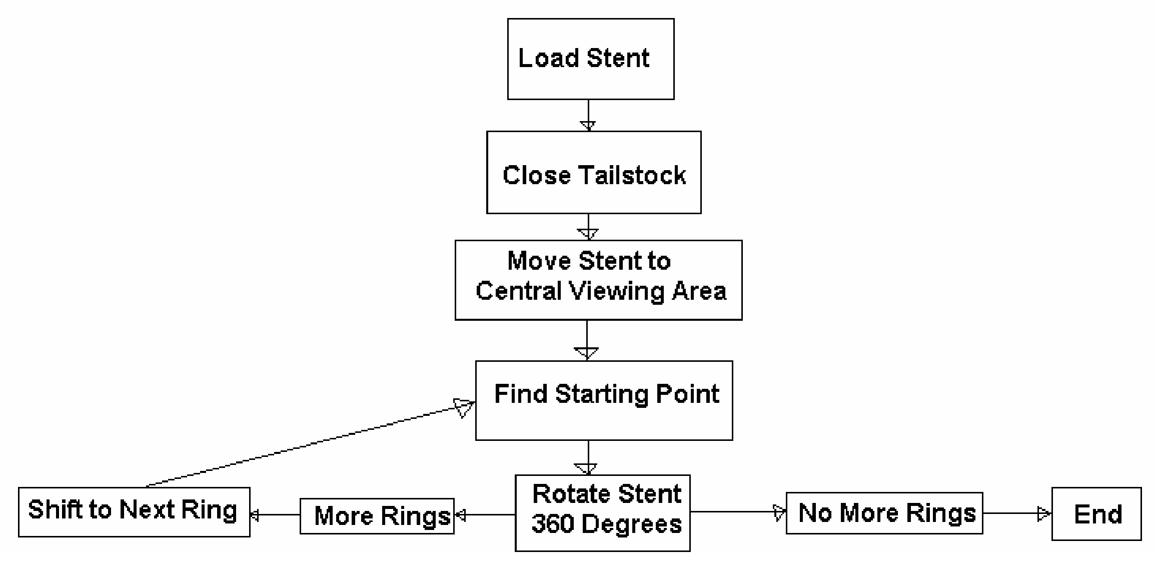


Figure 9: Manipulating the Stent Flow Chart

To complete this step of the inspection process, Guidant Corporation provided a CNC fixture with their custom vision console. The fixture consists of a custom chuck grip that rests on a CNC controlled track. The CNC controller in the machine vision console controls opening and closing of the tailstock. It is also responsible for maneuvering the chuck on its track. This controller communicates directly with the computer in the console. Using the computer, a program to control the motion of the mechanical fixture was developed. This code was copied to the CNC controller, where it ran with little supervision from the computer. The code for this program can be found in Appendix F.

3.5.1. Closing the Tailstock

To use the tailstock, simple G-code command was used. A binary output was associated with a variable to control. This variable controlled the tailstock. When it was equal to 1, the tailstock opened and when it was equal to 0, the tailstock closed.

3.5.2. Moving the Stent

To control the movement of the mechanical fixture, variables were defined for the target position in the X, Y, and Z directions. The X direction is east to west, the Z is north to south, and the Y is rotational. This specifies to the fixture the exact position that it should move to. Using G-codes other motion parameters such as the fixture's linear acceleration/deceleration and rate-based acceleration/deceleration were enabled. This allowed a specific type of acceleration to used for each movement. By using simple "Go" commands and specifying the coordinates to go to, it was possible to move the fixture into the central viewing area.

3.5.3. Finding the Starting Point

To start the stent inspection, the stent needed to be maneuvered so that a predefined starting point was centered underneath the camera. This starting point is defined by a specific juncture on the stent geometry, thus making it easy to find. Since the stent geometry repeats itself, defining this point was a simple task. It was decided that the inspection should start at the stent junction shown in Figure 10, due to ease of detection and the repeatability of alignment.

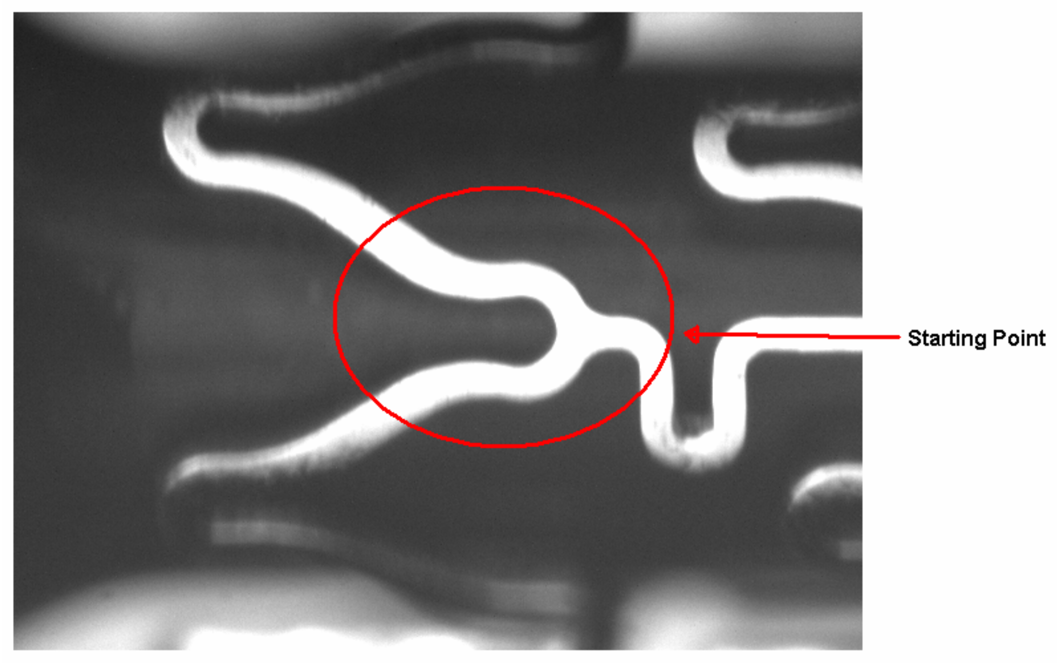


Figure 10: Inspection Starting Point

To locate this starting point, a FrameWork script was developed in conjunction with a stent manipulation program. The FrameWork script controls the camera and can be used to find certain features on the product in the central viewing area. The script is also used to communicate with SoftSensors that were defined during the development of the inspection. SoftSensors are tools used in FrameWork to manipulate or inspect an image. A description of these tools is available in Appendix D. By setting up SoftSensors within FrameWork, it was possible to scan the real time image of the central viewing area for distinctive stent features. The first step in the process was to set up an Intensity Soft Senor. The sensor scans the image for a predefined number of pixels with a bright intensity. Since the mandrel appears black under the camera and the stent appears white, this sensor can be used to find when the edge of the stent enters the central viewing area. Then, the stent will begin to slowly rotate, using a Template Match SoftSensor to try to locate the starting point geometry. The Template Match SoftSensor searches within the specified Scan Region for the predefined template that resembles the geometry in the above figure. When a match is made, the CNC fixture stops rotating and centers on the starting point. The SoftSensors used for this process are shown below.

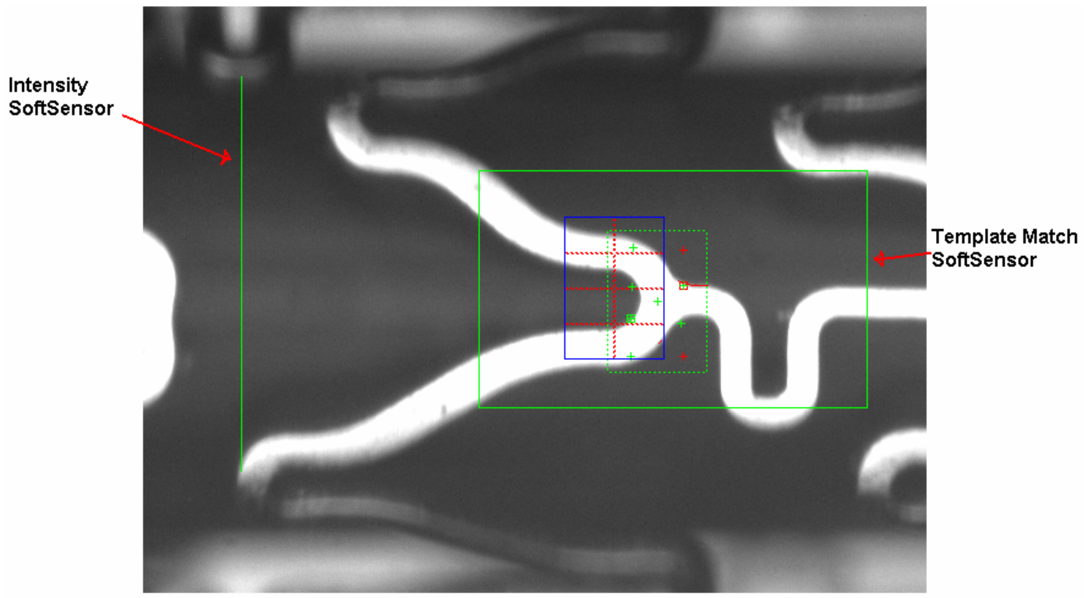


Figure 11: Starting Point SoftSensors

During testing, problems arose with the accuracy of the Template Match SoftSensor. Template Match SoftSensors are great for locating an object or shape but are only accurate to within a dozen pixels. This happens because the stent is at an angle when the Template Match SoftSensor identifies it, which skews the center of mass of the object and causes it to be misaligned. This misalignment caused some problems with realigning the stent after a ring had been inspected. To overcome this problem, multiple Blob SoftSensors were added to the design. Blob SoftSensors are accurate to within sub-pixels, so they served as perfect sensors for the subtle movements needed to realign the stent once the Template Match SoftSensor had identified it.

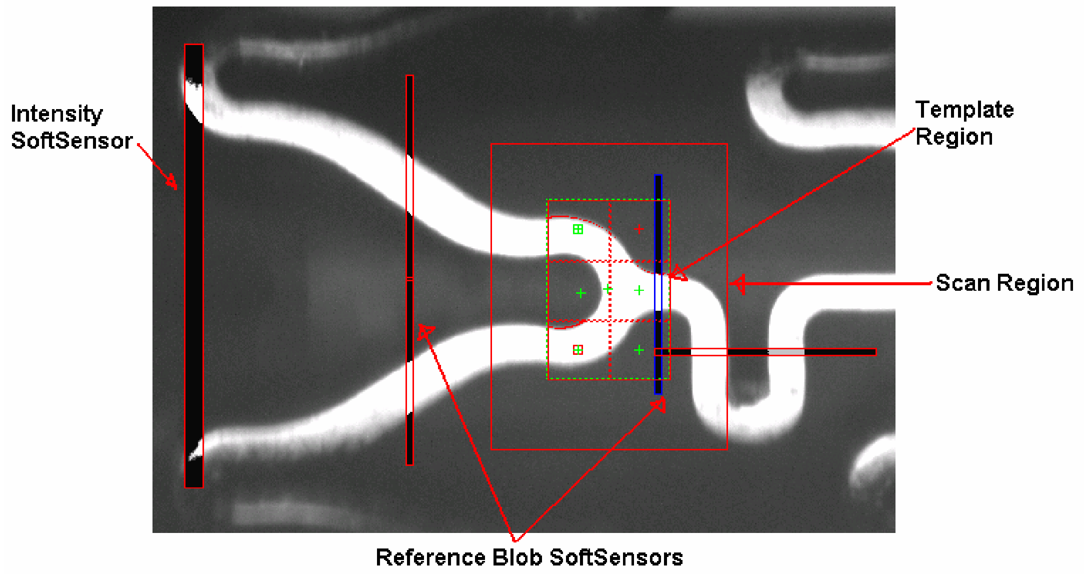


Figure 12: Starting Point SoftSensors with Blob SoftSensors

The final design used the strengths of the SoftSensors described above to accurately find the starting point on the stent. First, the Template Match SoftSensor locates the desired template and roughly centers the object under the camera. Then, a series of Blob SoftSensors are used to make the fine adjustments necessary to center it under the camera.

3.5.4. Rotating the Stent

Once the starting point of the stent had been found, the inspecting the stent is ready to begin. To do this with as much flexibility and efficiency as possible, the stent was divided into rings. A ring consists of a sliver of the stent with a repeating geometry that the camera can focus on. An example of a ring is shown in Figure 13.

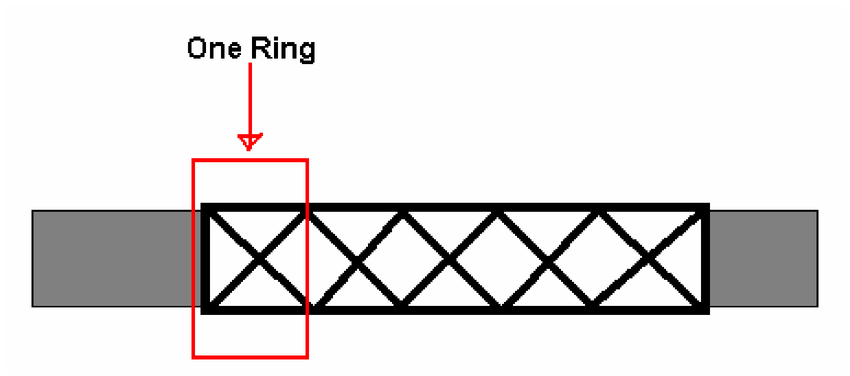


Figure 13: Example of a Ring

By dividing the stent into rings, it became easy to thoroughly inspect stents of variable length by simply increasing the number of rings to inspect. To rotate the stent, the same "Go" command from above was used. The only difference between the two is that the fixture moved in the Y (rotational) direction.

The last part of rotating the stent was to stop the rotation after 360 degrees. This also proved to be simple task. Based on background research, it was known that a stent ring included 12 geometrical features to examine. The fixture was programmed so that it stopped rotating when the camera had identified these 12 features. This completed the inspection of the ring and returned the stent to the starting point. The CNC controller then incremented the stent by the width of one ring and realigned it for the next inspection cycle. This moved the next ring of the stent into view and allowed us to continue with the inspection. By continuously finding the starting point in a ring, rotating the stent 360 degrees, and then incrementing to the next ring, the process was able to thoroughly inspect the entire outer surface of the stent.

3.6. Inspecting the Stent

After the algorithm to manipulate and properly align the stent for inspection had been created, the next step was to design sensors to scan the stent for defects. To do this, two sensor packages were created. The first sensor package was used to scan the strut and edges of the stent. It was called the Strut Sensor Package. The other sensor package was used to scan the area between the stent struts. It was called the Black Space Sensor Package.

Each sensor package was then divided up into sub-packages. The sub-packages were used to scan the 12 different geometric features of the stent. These features are shown in Figure 14.

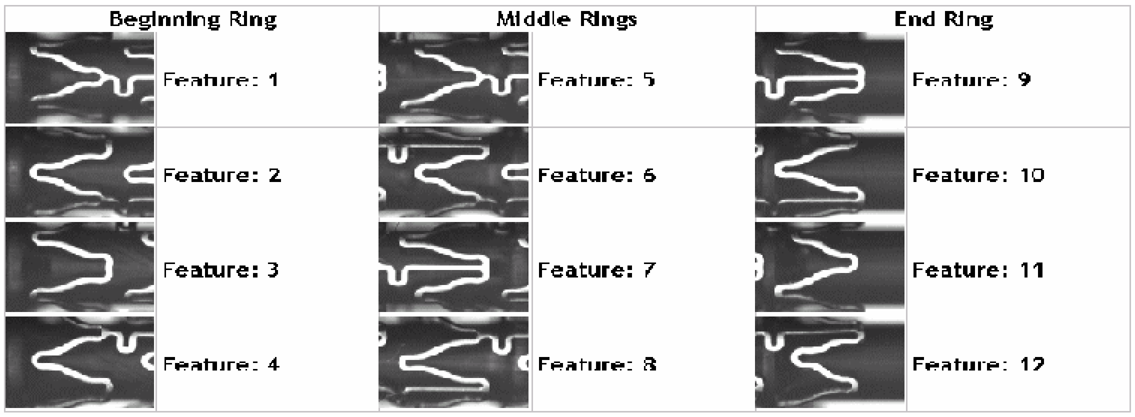
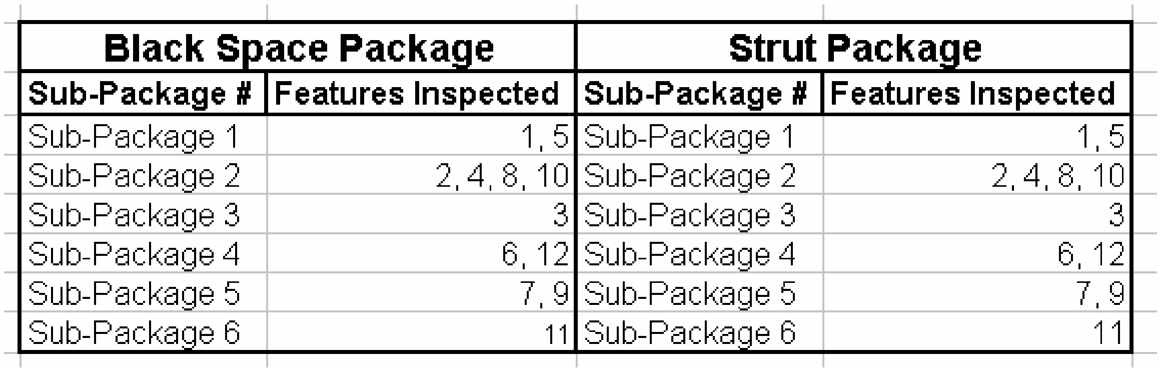


Figure 14: Stent Features

Since many of the stents' features are similar to each other, it was possible to condense the number of sub-packages that were used down to 6 sub-packages for the Strut sensor package and 6 sub-packages for the Black Space Sensor Package. These sub-packages and the features they inspect are shown in Figure 15. The DVT script that calls these sub-packages can be found in Appendix G and a complete description of the sensor packages and their sensor layouts can be found in Appendix E.



Black Space Package		Strut Package	
Sub-Package #	**Features Inspected**	**Sub-Package #**	**Features Inspected**
Sub-Package 1	1, 5	Sub-Package 1	1, 5
Sub-Package 2	2, 4, 8, 10	Sub-Package 2	2, 4, 8, 10
Sub-Package 3	3	Sub-Package 3	3
Sub-Package 4	6, 12	Sub-Package 4	6, 12
Sub-Package 5	7, 9	Sub-Package 5	7, 9
Sub-Package 6	11	Sub-Package 6	11

Figure 15: Sensor Sub-Package List

3.6.1. Strut Sensor Package

The purpose of the Strut Sensor Package was to scan the surface and edges of the stent for defects. To do this, two sets of Blob SoftSensors were used. The first set of Blob SoftSensors properly aligned the features of the stent and the second set scanned the surface of the feature for defects.

The first set of Blob SoftSensor was used to locate predefined struts within the feature's geometry. Blob SoftSensors work by separating the viewing area into light and dark "blobs." By defining the Blob search area and identifying the type of blob (light or dark) to search for, the user can obtain blobs of a certain size and shape. Once a blob had been had obtained, the Blob Selector could calculate the center of mass, perimeter, and surface area of the blob. By placing long, thin Blob sensors at strategic points within the camera view the sensors could sample the image on the screen and determine where the stent struts were located. By using a series of three or four of these reference blobs, it was possible to locate all of the key struts within a stent image. This allowed a stable reference system to place inspection sensors around the stent struts. A sample of the reference blobs is shown in Figure 16.

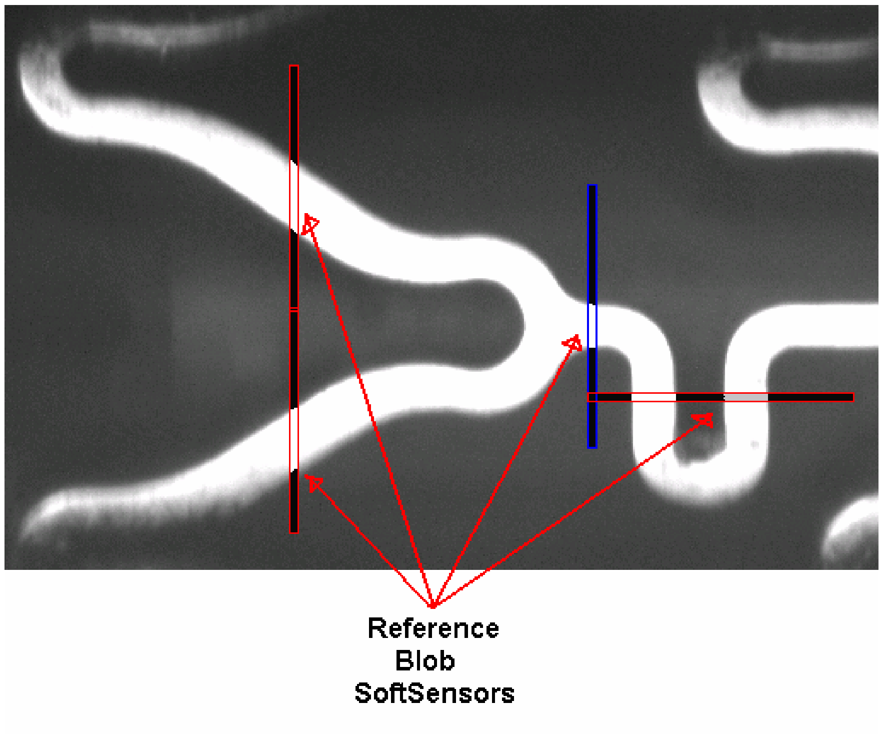


Figure 16: Reference Blob SoftSensor

By identifying the various struts of the stent feature and calculating the centers of mass for these struts, the system was able to account for any misalignment that occurred while the stent was being spun.

As discussed above, the Blob SoftSensors separate the white stent surface from the black mandrel background. This feature was used to identify defects that would appear on the edges of the stent. Using a series of Blob SoftSensors, it was possible to define search areas and specify which type of blob (light or dark) the sensor should identify. The sensor then identified all of the blobs or light areas within the search area. After that, Blob Selectors were created to scan these blobs. The selectors used the Blob SoftSensors to calculate the surface area and perimeter of a given blob. By searching for drastic changes in the blob's perimeter or surface area, the system is able to effectively identify defects on the stent's surface and edges. Figure 17 shows the Blob SoftSensors and Selectors for this package.

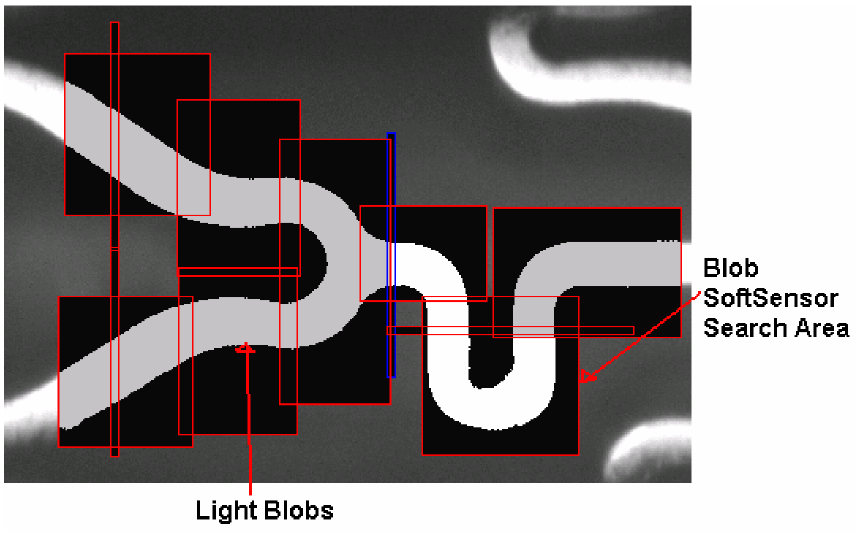


Figure 17: Strut Blob Sensor Layout

Using many small Blob SoftSensors instead of one large Blob SoftSensor increased the sensitivity of the inspection. Defects on a large blob appear as a small change to a large perimeter or surface area. Defects on a small blob appear as large changes to a small perimeter and surface area. Figure 18 shows an example of a defect being detected by the Strut Sensor Package.

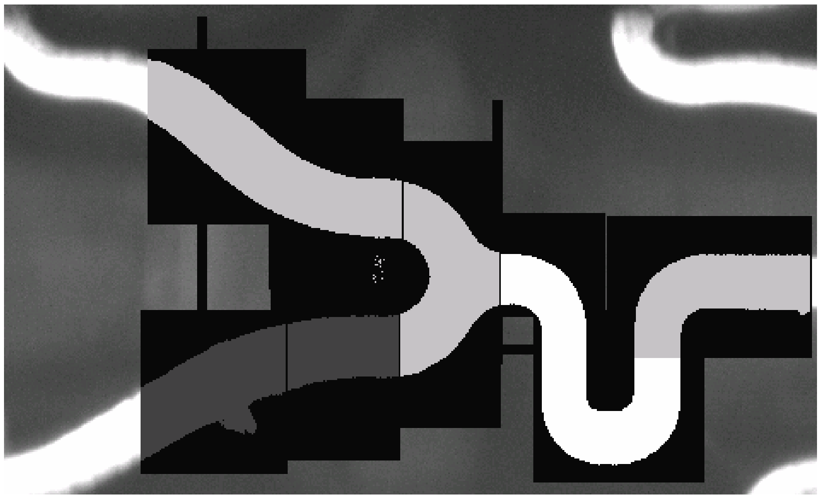


Figure 18: Strut Blob Sensor Layout with Defect

3.6.2. Black Space Sensor Package

The Black Space Sensor Package was designed to be very similar to the Strut Sensor Package. It used the reference blobs described above to align a series of Blob SoftSensors and Blob Selectors. These Intensity SoftSensors inspected the dark area between the stent's struts for defects.

As in the Strut Sensor Package, Blob SoftSensors and Blob Selectors are used to detect light blobs in the dark background and calculate the area and perimeter of these blobs. By defining these areas as shown in Figure 19, it became possible to inspect the space between the stent struts.

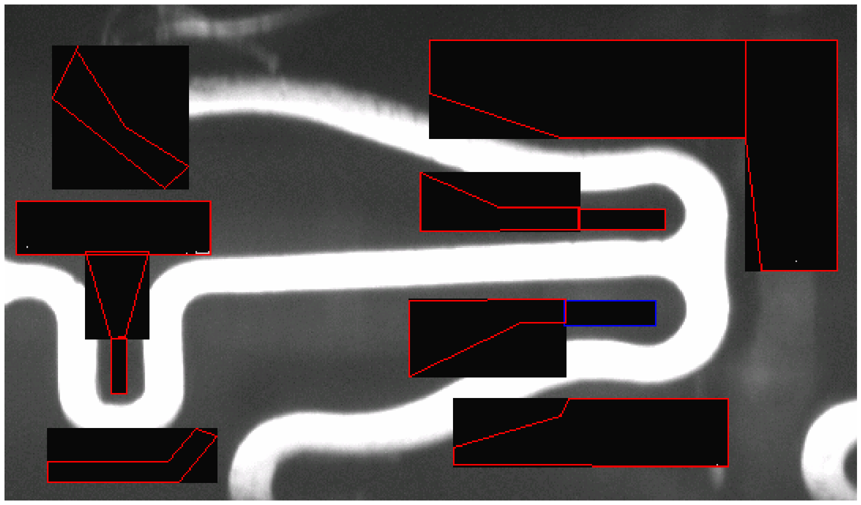


Figure 19: Black Space Blob SoftSensor Layout

As the figure shows, this space between the struts appears as a very dark background. By setting a specific contrast for the Blob SoftSensors, very fine defects that occurred in this area could be detected. These fine defects showed up as very bright spots on the inspection screen. After the defects were detected, it was important to determine if they were severe enough to reject the feature. This would allow minor defects and mandrel noise to pass without disrupting the inspection. The DVT script for the inspection uses the areas of all of the detected blobs to calculate the sum of the areas. It then compared this area to a threshold value. Based on the defect specifications, it was determined that a defect consists of 20 or more pixels. If the sum of the light blobs had an area greater than this value, the feature would be failed. Figure 20 shows a stent with a defect that has been deemed too large.

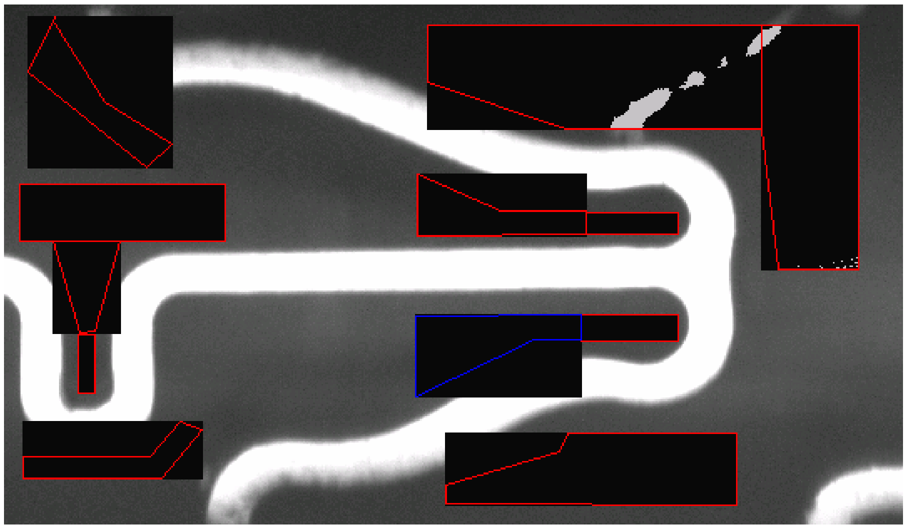
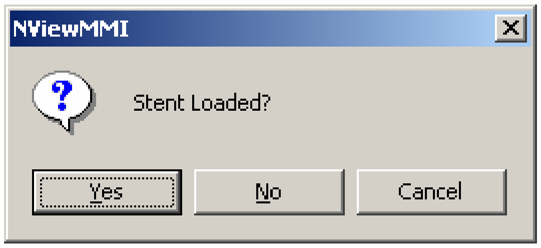


Figure 20: Black Space Blob SoftSensor Layout with Defect

Calculating the sum of the blob areas allowed the inspection to avoid detecting mandrel noise and acceptable defects that would result in false positives.

3.7. Managing the System

To manage the inspection system, a user interface was created using G-Code prompting. The inspection system interfaced with the user through a series of prompts. When To start the system, the user was prompted to load the stent into the CNC fixture. This prompt is shown in Figure 21.

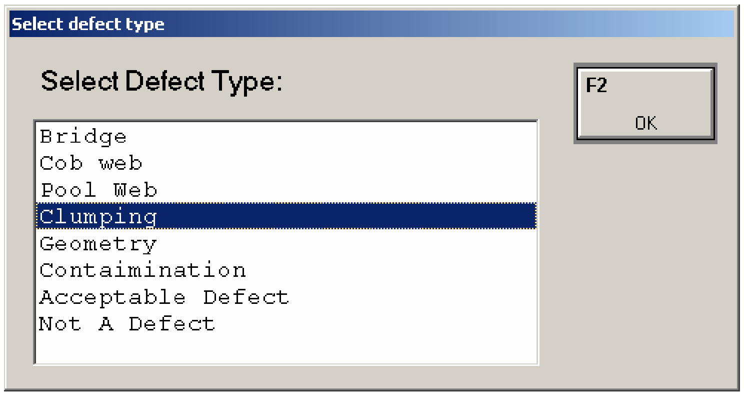


NViewMMI

Stent Loaded?

Figure 21: Stent Loading Prompt

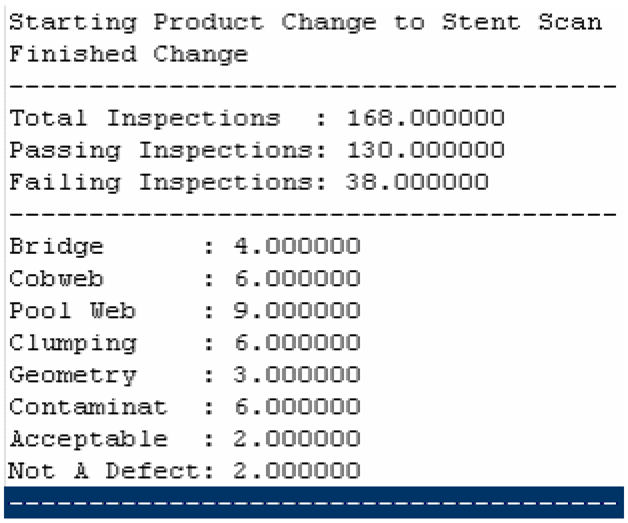
The next prompt appeared whenever a potential defect was found. The prompt would display a list of possible defect types. The inspection system automatically selects one defect type based on the sensor that failed and the conditions of the failure. This defect type is highlighted in the list and the operator has the option to correct any misclassifications by selecting another defect type. This prompt is shown in Figure 22.



```
Bridge
Cob web
Pool Web
Clumping
Geometry
Contaimination
Acceptable Defect
Not A Defect
```

Figure 22: Potential Defect Prompt

At the end of the inspection, a summary was generated by the inspection system and the operator is given the option to load and inspect another stent. This report was displayed in the NView MMI Message Box and contains all of the information regarding the defects found and their classification. An example of this summary is shown in Figure 23.



```
Starting Product Change to Stent Scan
Finished Change
--------------------------------------
Total Inspections  : 168.000000
Passing Inspections: 130.000000
Failing Inspections: 38.000000
--------------------------------------
Bridge       : 4.000000
Cobweb       : 6.000000
Pool Web     : 9.000000
Clumping     : 6.000000
Geometry     : 3.000000
Containinat  : 6.000000
Acceptable   : 2.000000
Not A Defect: 2.000000
--------------------------------------
```

Figure 23: End of Inspection Summary

3.8. Deliverables

The deliverables for this project were the source code for the inspection system, a software description detailing the steps needed to recreate the sensor packages, a hardware description detailing the steps needed to adjust the machine vision console for to run an inspection, and a users guide detailing how to properly run an inspection using the software and hardware.

The source code for the inspection system is divided between a DVT script that runs that actual inspections and a G-Code document that controls the CNC motion. The code for the G-Code program is located in Appendix H and the code for the DVT script can be found in Appendix I.

The software description contains the detailed information needed to recreate all of the sensors used for the inspection. This description can be found in Appendix E.

The hardware description contains all of the information needed to adjust and rewire the console to run an inspection. The hardware description can be found in Appendix G.

4. Results and Analysis

After the automated inspection process had been designed, it needed to be tested. The tests generated the experimental results that were used to analyze the accuracy, effectiveness, and efficiency of the system. The purpose of automating the inspection of stents was to create a system that would replace the manual inspection process that was being used. To do this, it needed to be proved that the automated inspection process could provide an inspection that was as accurate or more accurate than the manual inspection. It also needed to be proven that the inspection could run in a timely and efficient manner. The results of these tests, the required analysis of these results, and summaries of the major changes made to the original design are discussed in this section.

4.1. Experimental Results

To test the accuracy and flexibility of the inspection, it was necessary to use a variety of stents. Originally, the inspection was going to be tested with 20 different stents. Fifteen of the stents were manufactured to specifically contain known defects. They were expected to fail and test the accuracy and sensitivity of the system. Five of the stents were manufactured normally and it was assumed that they would pass inspection. The stents that were expected to pass the inspection would test the robustness and flexibility of the inspection.

During the inspection testing, it was found that only 15 of the stents could be inspected. The 5 omitted stents were excluded because they contained gross geometry defects. Since these defects occurred in the beginning ring, they caused the inspection alignment to fail. This meant that the inspection could not find the starting point in the geometry and resulted in the inspection failing the stents. These failures proved that the inspection could accurately detect geometry defects and fail the geometrically defective stents. Two of the remaining 15 stents also contained gross geometry defects. These defects were in the middle rings of the stent and resulted in the stent failing halfway through the inspection. The detailed results for these two stents and the remaining 13 stents are shown in Appendix H. The overall results from the inspections are shown below.

Type	Detailed Reason	Count	Percent
Total Stents		15	100.00%
Total Inspections		2520	100.00%
False Triggers		190	7.54%
	Mandrel Noise	28	1.11%
	Black Space Blobs > Threshold	79	3.13%
	Clump Area > Tolerance	10	0.40%
	Acceptable Clumping Strut Area	1	0.04%
	Strut Area > Threshold	29	1.15%
	Strut Perimeter > Threshold	11	0.44%
	Alignment Error	32	1.27%

True Defects		334	13.25%
	Black Space Blobs > Threshold	290	11.51%
	Clump Area > Tolerance	13	0.52%
	Acceptable Clumping Strut Area	10	0.40%
	Strut Area > Threshold	3	0.12%
	Strut Perimeter > Threshold	3	0.12%
	Alignment Error	15	0.60%
Missed Defects		2	0.08%
	Black Teflon Flake	1	0.04%
	Thin Cobweb	1	0.04%

4.2. Experimental Analysis

As the results table above shows, the automated stent inspection system works very close to the Product Specifications outlined above. It performs the inspection of an 18 mm stent in less then 1.5 minutes, identifies less than 8% false positives, and missed 0.08% true defects.

The stent inspection time is 30 seconds over the time defined in the Product Specifications. During the design process, it was determined that additional time was going to be needed to ensure full coverage of the stent. This final inspection time was deemed acceptable by Guidant. Inspection accuracy and reliability were the most important Product Specifications. The inspection time could exceed 1 minute as long as it meant increased accuracy.

During an inspection, less than the specified 20% false positives were identified. This is well within the predefined specifications. Ideally, this number would be as close to zero as possible and given more time it could be reduced. However, design preference was given towards creating an inspection that would allow no defects to slip through.

The inspections performed on the test stents resulted in two defects being missed. This was very alarming until an image of the defects was viewed. Both defects were found to be minor and using the inspection system defined above, impossible to detect.

The first of these defects was a black Teflon contaminant on the strut of the stent. It is shown in Figure 24. The black surface of the contaminant makes it impossible to detect using this particular inspection process. The contaminant is the same material and color as the mandrel, so lowering the sensor thresholds to detect a black contaminant would result in all inspections triggering false positives. This would result in the inspection being rendered useless because it would identify a false defect on every stent feature inspected.

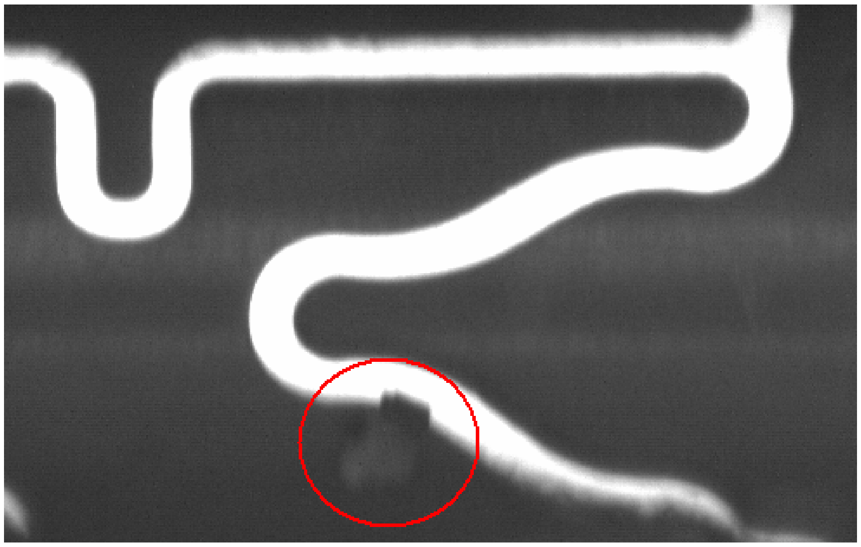


Figure 24: Black Teflon Contaminant

The second defect that the inspection missed was a very small cob web. This cob web is shown in Figure 25. As the figure shows, the intensity of this cob web is very close to the intensity of the mandrel surface. This makes it impossible for the Blob SoftSensor, shown in Figure 26, to identify this defect at the current threshold values. The inspection threshold could be reduced so that this defect could be caught, but the sensor robustness would be compromised. The inspection would become so sensitive that it would pick up an excessive amount of mandrel noise and produce a false defect on every stent feature.

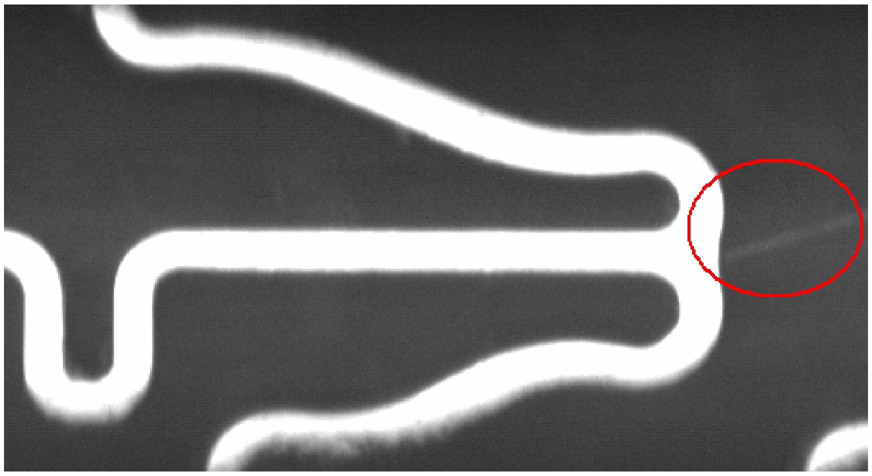


Figure 25: Missed Cob Web

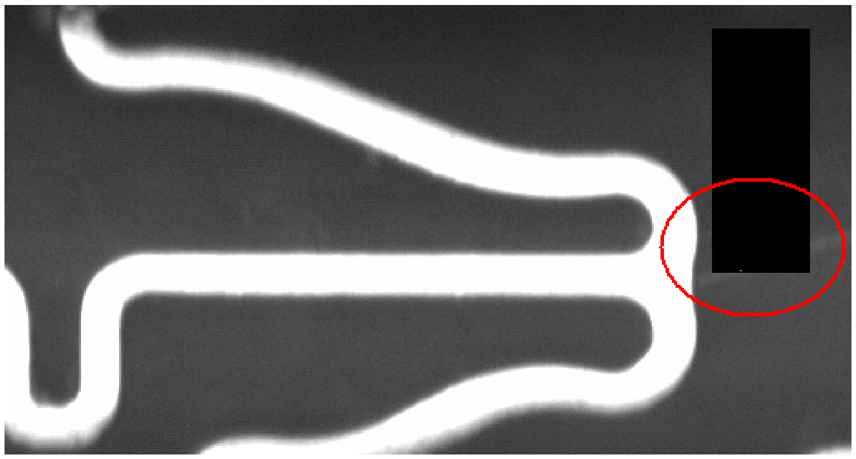


Figure 26: Missed Cob Web with Blob SoftSensor

4.3. Design Issues

During the design process, it became obvious that several of the sensor packages and design ideas needed to be modified or replaced to combat the problems that arose within the inspection. It was important to document the original design ideas, the changes made to them, and the reasons the changes were made. This makes it easier for other people within Guidant Corporation to continue the work that was started on this project. Documenting any dead ends found in the design process allows the employees of Guidant to upgrade the inspection without encountering the same problems that this project faced.

4.3.1. Inspection Time

The first major challenge faced during the creation of the inspection was that the inspection took too long to complete. The initial test inspection took over 7 minutes to complete. This was much greater than expected and unacceptable. To combat this problem, different sensors were used for stent alignment.

4.3.1.1. Template Match SoftSensors

A major change made to the inspection design was to reduce the number of Template Match SoftSensors used. Originally, the design called for a combination of Template Match SoftSensors and Rotational SoftSensors to be used to align the Strut and Black Space Sensor Packages. The process was very similar to the alignment process described above and was used for each sensor package. This meant that there were 15 Template Match SoftSensors and more than 15 Rotational SoftSensors being used for the inspection. For each feature, the Template Match SoftSensor would search for the predefined Template Region within the Scan Region. Once it had found this feature, it would center it within the screen. Then, the Rotational SoftSensor would align on a stent strut and calculate the percent rotation between the current feature and the predefined feature. This sensor package is shown in Figure 27.

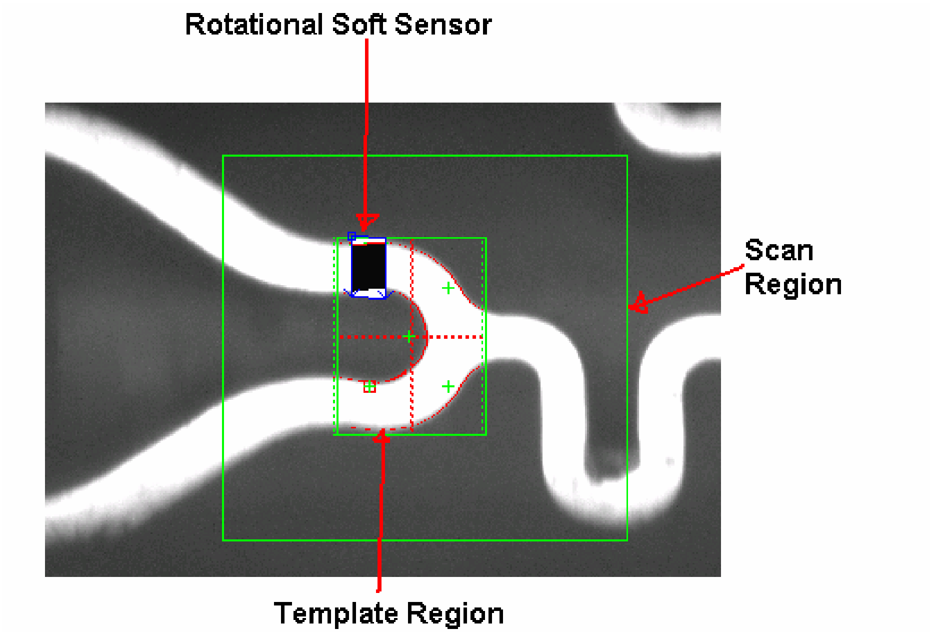


Figure 27: Template Match and Rotational SoftSensors

It was discovered that a complete inspection of an 18 mm stent took over 7 minutes to complete. As shown by the Product Specifications defined above, this was unacceptable. After much testing and consultation with the project liaison Jason Van Sciver, it was determined that the cause of this lengthy inspection was the Template Match SoftSensors. The original design called for a Template Match and a Rotational SoftSensor to be used for every Black Space and Strut Sensor Package, along with the Alignment sensor packages. By changing these sensors to the current combination of Blob SoftSensors, the sensors could still be properly aligned and 14 Template Match and Rotational SoftSensors were eliminated from the design. This reduced the inspection time to 1.5 minutes.

4.3.2. False Positives

The second major problem that the inspection faced was to find a way to combat mandrel noise. Mandrel noise consists of light spots that appear on the dark background during the inspection. These bright spots were sometimes detected by the inspection and would result in a false positive.

4.3.2.1. Internal Triggering

One problem with the original inspection process that contributed to the excessive false positives was that it was difficult to properly synchronize the CNC fixture with the DVT camera. The process was first designed using internal triggering. The camera was programmed to image the stent every 175 milliseconds. This proved to be very unreliable and resulted in pictures that were off center. Since the camera was imaging the stent at such a high rate, any deviation in the amount of time that it took to inspect the stent feature and rotate the stent resulted in a misalignment. This caused errors in the inspection and returned too many false positives.

To rectify this problem, external triggering was used. The camera was wired to the CNC fixture and programmed to wait for an input from the fixture to take a picture. The new process starts with the CNC fixture rotating the stent to the predefined starting point. When it reaches this point, it ceases all movement and signals to the camera to take a picture. After an image is taken, the camera inspects the stent feature. When the inspection is completed, it signals back to the CNC fixture and the fixture increments the stent to the next feature. This new process is a little more deliberate than the previous one and requires more physical wiring, but less software communication. With this lack of additional software handshaking, external triggering is more robust than the original design and less prone to misaligning the stent. This made it the best choice for the design. A more detailed description of the hardware changes required to implement external triggering is located in Appendix G.

4.3.2.2. Black Space Intensity SoftSensors

Originally, the Black Space Sensor Package was designed using Intensity SoftSensors. Intensity sensors are very useful because they can be used to detect black and white pixel change within a predefined area. By defining these areas as shown in Figure 28, it became possible to inspect the space between the stent struts.

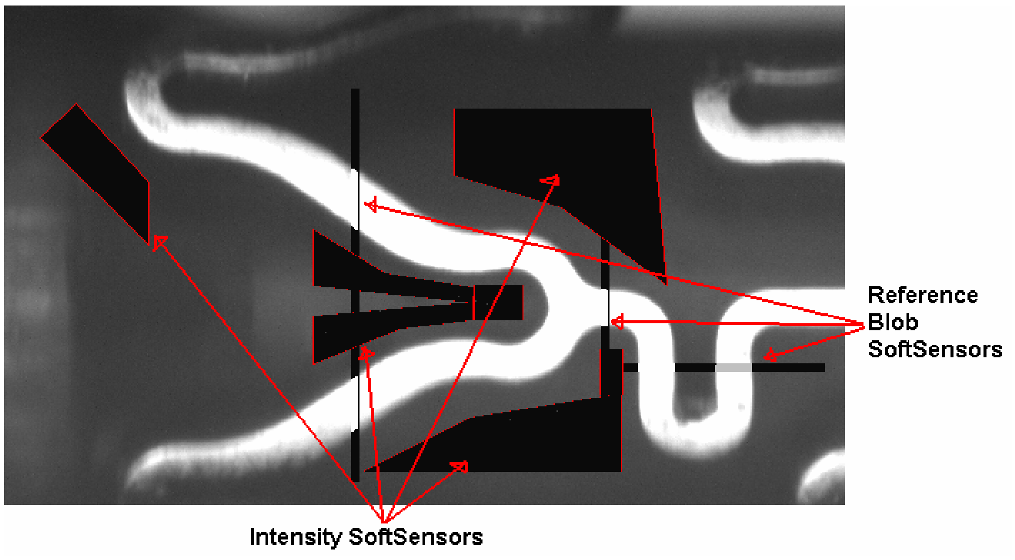


Figure 28: Intensity SoftSensor Layout

As the figure shows, this space between the struts appears as a very dark background. By setting a specific contrast for the intensity sensors, very fine defects that occurred in this area could be detected. These fine defects showed up as very bright spots on the inspection screen. Figure 29 shows a stent with a defect in between the struts and the intensity sensors that identified it.

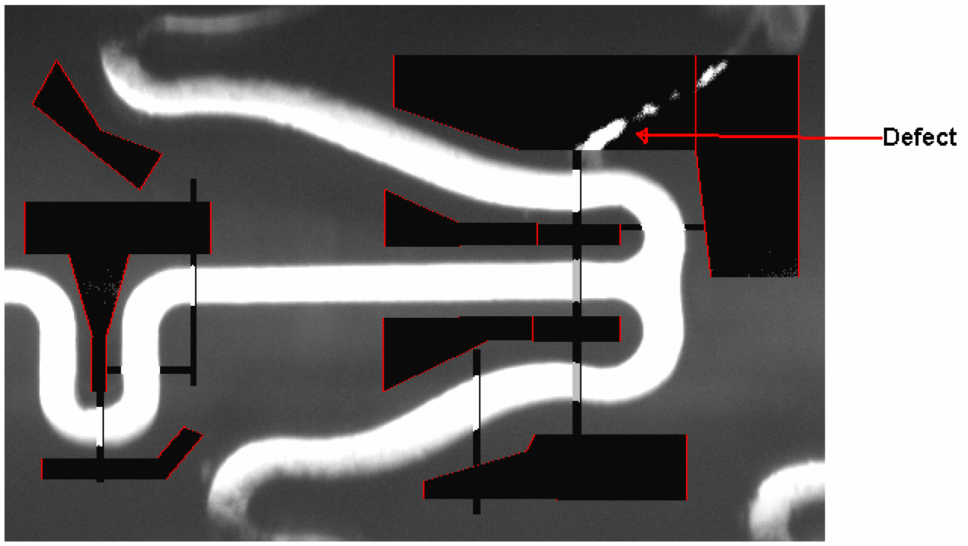


Figure 29: Intensity SoftSensors with Defect

This change in pixel intensity tripped the Intensity SoftSensor and alerted the system that a defect had been found.

The problem with Intensity SoftSensors was that they did not record or save any of the information regarding the light pixels that they identified. During the design process, it became necessary to reduce the sensitivity and increase the robustness of the inspection. It proved impossible to do this with Intensity SoftSensors because they recorded no information regarding the pixels they observed. Part of the inspection was to determine the sum of all of the bright pixels in a search area. If this sum was above the specification of 10 pixels, it was determined that the feature was defective. By replacing the Intensity SoftSensors with Blob SoftSensors and Blob Selectors, the sensor package was able to store data regarding the area and perimeter of all of the light pixels in the scan area. This allowed the inspection to calculate the sum of the light pixels and added a great deal of robustness to the design.

4.3.2.3. Linear Intensity SoftSensor

Another minor change that was made during the development of the inspection was that the Intensity SoftSensor responsible for finding the edge of the stent was functioning incorrectly. Since the SoftSensor is designed to detect when a change from dark pixels to bright pixels occurs, it could be easily fooled into finding a false edge. Any bright spots or “noise” on the mandrel was being detected and labeled as the edge of the stent. This caused the inspection to go into an endless loop while it looked for the starting point. The initial Intensity SoftSensor layout is shown in Figure 30.

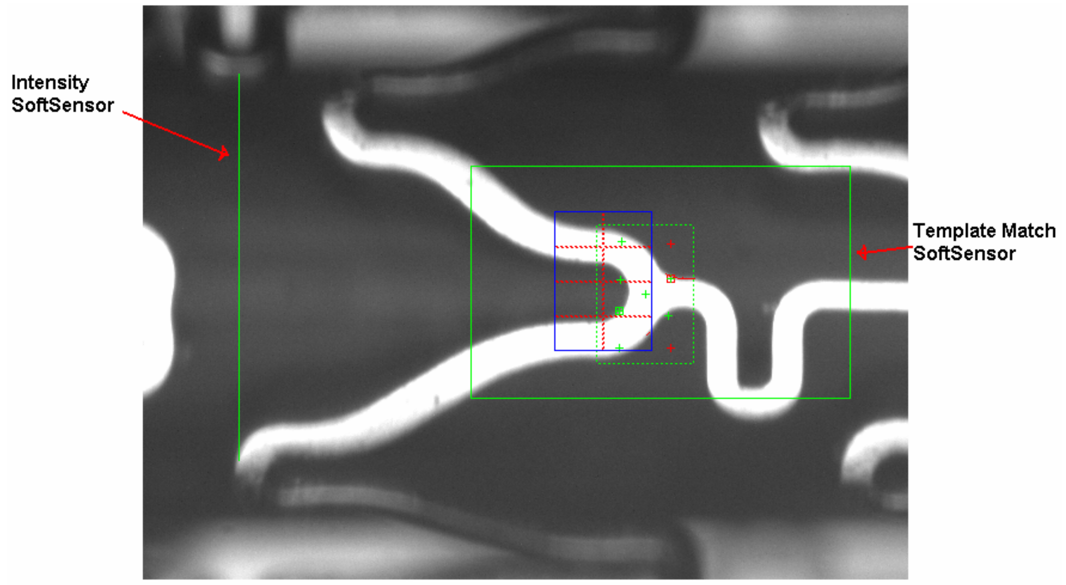


Figure 30: Original Intensity SoftSensor Layout

To rectify this problem, the area of the Intensity SoftSensor was increased. The Intensity SoftSensor works by detecting a percentage of light pixels in a specified area. When the percentage of light pixels rises above a predefined threshold, it returns a PASS. Light spots on the mandrel were causing this sensor to return an incorrect PASS because the search area was not large enough to allow for small amounts of bright pixels of the mandrel. When the search area was increased, the light pixels that represented the light spots on the mandrel became a smaller percent of the overall area. This reduced percentage was well below the predefined threshold. By increasing the area, the errors caused by small amounts of mandrel noise were eliminated. The final layout for the Intensity SoftSensor is shown in Figure 31.

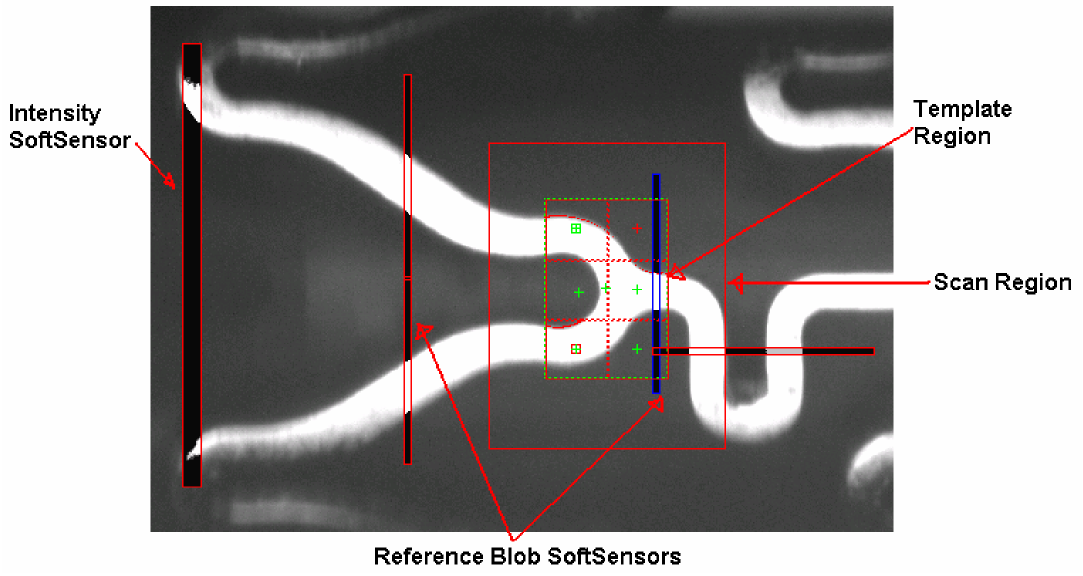


Figure 31: Final Intensity SoftSensor Layout

4.3.2.4. Ridged Mandrel

The major cause of the false positives that the inspection was generating was image noise from the mandrel. This noise was caused by the structure of the mandrel itself. Initial inspections were performed with a ridged mandrel. The ridges in the mandrel caused most of the noise that were causing the inspection's false positives.

Mandrels are made of darkened carbide rods that are cut to a set length and diameter. Carbide is used because it does not lose shape and suffer from run out. The carbide mandrels are either perfectly shaped or broken, never bent or warped. The mandrels are then coated with a thin layer of Teflon. The Teflon is used to facilitate the loading/unloading of the stent from the mandrel and increase the darkness of the mandrel surface.

Originally, a ridged mandrel was used to hold the stent during the inspection. The mandrel surface contained small ridges that minimized the amount of contact between the stent and mandrel while effectively holding the stent in place. The problem with these mandrels was that the ridges made it difficult to uniformly coat them with Teflon. Creases and light spots would appear on the mandrel surface near the ridges. These bright spots would be detected by the inspection and classified as defects. An example of mandrel noise caused by a ridge is shown in Figure 32.

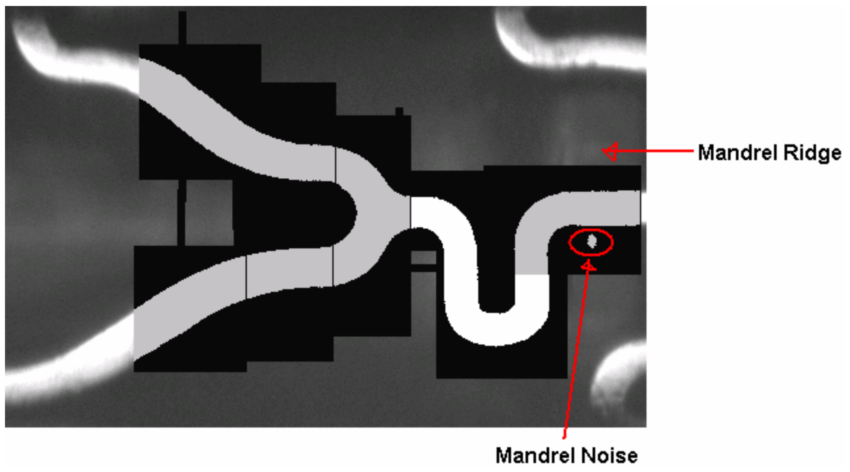


Figure 32: Example of Mandrel Noise

To solve the problems caused by the ridged mandrel, a flat mandrel was substituted into the inspection. The flat mandrel was uniformly coated with Teflon, so the bright spots that caused mandrel noise were greatly reduced. It was found that changing mandrel types reduced the number of false positives by nearly 33%.

Another added benefit of the flat mandrel was that it made it easier for the inspection to detect contamination. Contamination consists of a foreign particle exists on the stent. Usually, contamination is found in the form a strand being woven in and out of the stent geometry. An example of contamination is shown in Figure 33.

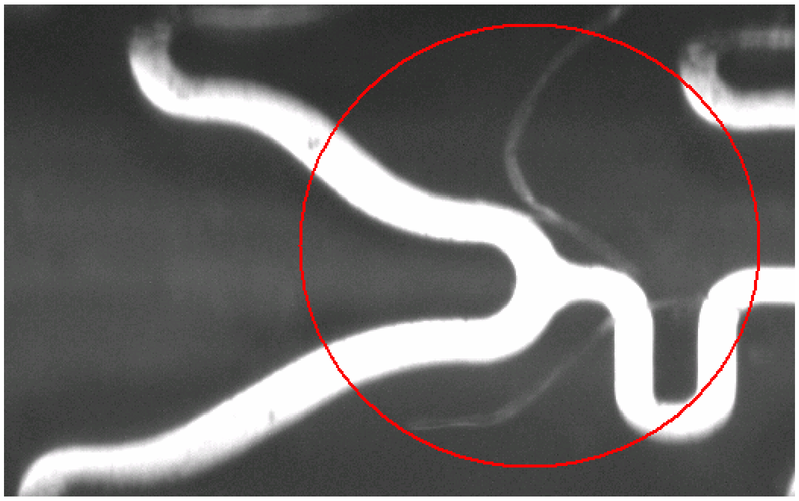


Figure 33: Contamination with Ridged Mandrel

This made it very hard to detect when using the ridged mandrel. Using the flat mandrel forced the contamination to the surface of the stent and made it much easier to detect. Figure 34 shows an example of contamination being forced to the surface of the stent.

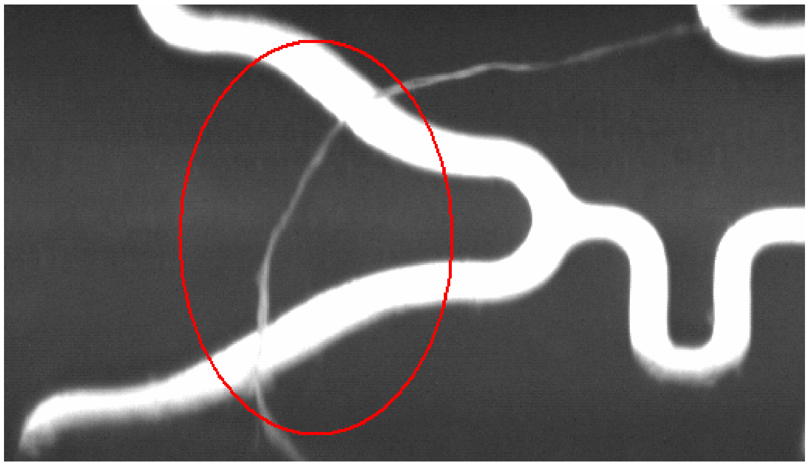


Figure 34: Contamination with Flat Mandrel

5. Conclusions and Recommendations

The goal of this project was to design an automated, machine vision based stent inspection system. This inspection system was the first prototype of a system that Guidant may eventually use to replace the manual inspection for their drug-eluting stents.

5.1. Design Conclusions

The results, while meeting the main objective of the project, show that it was very difficult to produce an inspection system that does not pass any defective stents, fail any acceptable stents without operator intervention, and inspects at a rate of 18 mm per 1 minute. Creating a sensor package that was sensitive enough to catch 100% of the defects resulted in the inspection time and the number of false positives increasing. A balance of these three objectives was achieved and the result of this project was an efficient yet accurate inspection created for Guidant Corporation.

Further work and testing is required before this inspection system can be fully implemented. During the testing of the inspection, it was found that the system responds negatively to significant changes in the stent geometry or contamination caused by flakes from the Teflon mandrel coating. These minor issues, as well as the flexibility and robustness of the system will need to be addressed to further improve upon an already accurate and successful inspection system

Despite the conclusions drawn above, this project was viewed as a success. When a stent feature is being inspected, the chances of a true defect being missed or a false positive being identified are 0.08% and 7.54% respectively. These numbers are very promising. Based on the size and variety of defects found on a stent, these numbers can be deemed very acceptable. The specifications laid out at the beginning of the project were quite lofty and this project came very close to completing them. The inspection system created for Guidant Corporation is a very good first prototype and, with a little more time, can be easily upgraded to achieve the desired accuracy and flexibility.

5.2. Design Recommendations

Based on the results generated in the previous tests of the inspection, it was determined that additional testing using production stents are required. Also, upgrades to the mandrel, lighting system, inspection algorithm, and user interface are needed to reduce the number of false positives generated by the inspection and increase robustness of the system.

5.2.1. Additional Testing

Further testing of the current algorithm is also required. The testing performed on this inspection was limited to a batch of 25 stents. 5 of these stents were used to program the inspection and the other 20 were used to test its accuracy. All of the stents used only had a polymer coat, so some testing would need to be done on the drug coated stents from the manufacturing line. Testing stents off of the manufacturing line would validate the algorithm's sensors. To further evaluate the capability of the inspection, it will need to be tested on more stents with wider variety of defects.

5.2.2. Mandrel Upgrade

During the inspection testing, problems arose with loading the stent onto the flat mandrel. Since the mandrel is flat, the interior of the stent could catch against the edge of the Teflon coating applied to the mandrel. In rare cases, the coating would flake off and contaminate the stent or one of the crowns would slip under the Teflon. One such defect occurred during the testing and proved impossible to detect.

To solve this problem, Guidant could make minor adjustments to the flat mandrels. By adding small grooves to either end of the mandrel, they could remove the possibility of the stent catching on the end of the Teflon coating. The proposed mandrel upgrade is shown in Figure 35. This upgrade would also make loading and unloading the stent less damaging, since there would be a taper for the leading ring of the stent rather than a flat edge of the 0.002" Teflon coating.

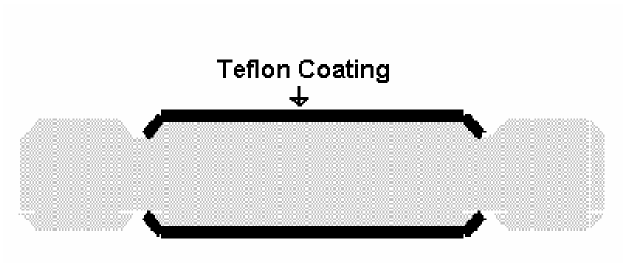


Figure 35: Proposed Mandrel Upgrade

5.2.3. Lighting Upgrades

One of the problems that caused some of the false positives was shadowing. Shadowing occurs when the lighting of the inspection console is centered at one location. The fixture illuminates the center of the stent but fails to properly adjust to the stent's surface contours. Since the lighting on the inspection console used for this inspection was located directly above the viewing area, it highlighted the center of the viewing area well but left shadows near the edges of the image. These shadows were inconsistent and varied from feature to feature. The shaded area could appear under the sensors as dark pixels, light pixels, or combination of the two. This combination of light and dark pixels could be identified as a defect and sometimes resulted in a false positive being generated. This inconsistent intensity of the edge struts made it impossible to predict if the shadows would interfere with the inspection sensors or not. Reducing shadowing would produce a more consistently lit image for inspection and reduce the number of false positives generated by the inspection. An example of shadowing is shown in Figure 36.

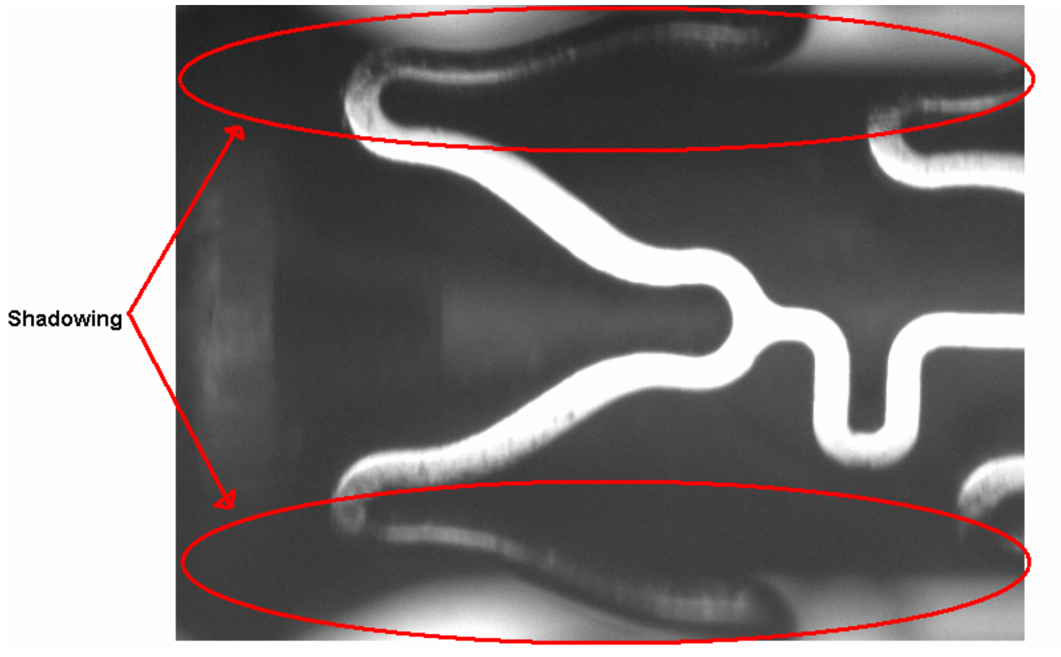


Figure 36: Example of Shadowing

One solution to reduce shadowing would be to switch to a lighting fixture that illuminates more of the stent. One way to do this would be to reflect the light into the viewing area at many angles. This would result in a more consistent intensity on the edge of the viewing area and reduce the number of false positives generated by shadowing. This would also assist in catching cobwebs that move from one feature to the next, since they would be directly illuminated without the reflection of the stent behind them. The proposed upgrade to the lighting system is shown in Figure 37.

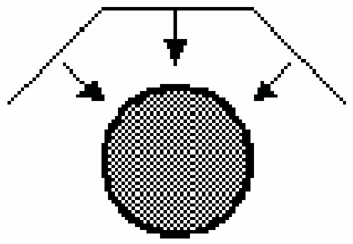


Figure 37: Proposed Lighting Upgrade

5.2.4. Algorithm Upgrade

Currently, the inspection possesses the capability to inspect the stent and identify defects, but not classify them. Upgrading this area of the inspection algorithm would prove very useful. Programming the inspection to identify and classify the defect would negate the need for supervision from an operator and make the inspection fully automated. Since this algorithm upgrade would be used to fully automate the inspection, it should only be performed after the number of false positives has been significantly lowered. This would ensure that the inspection would be both automated and accurate.

5.2.5. User Interface Upgrade

To manage the inspection system, a user interface was created using Visual Basic. The interface has already been developed and tested in Visual Basic and it is predicted that it will serve as an upgrade of the current UI when implemented. Unfortunately, the driver class files required for communication between the Visual Basic, CNC, and DVT systems could not be obtained in time for proper implementation. This made it impossible to compile the program and run tests to see if it communicates with the camera and CNC controller as expected. By obtaining these files and compiling them with the interface described below, the UI can be easily upgraded.

This user interface runs an inspection initialization that checks the system to make sure that the camera and CNC fixture are properly connected. It also allows the operator to start/abort the inspection, view and save an image of any features that failed inspection, accept/reject the feature, and classify the defect by type. The main screen for the user interface is shown in Figure 38.

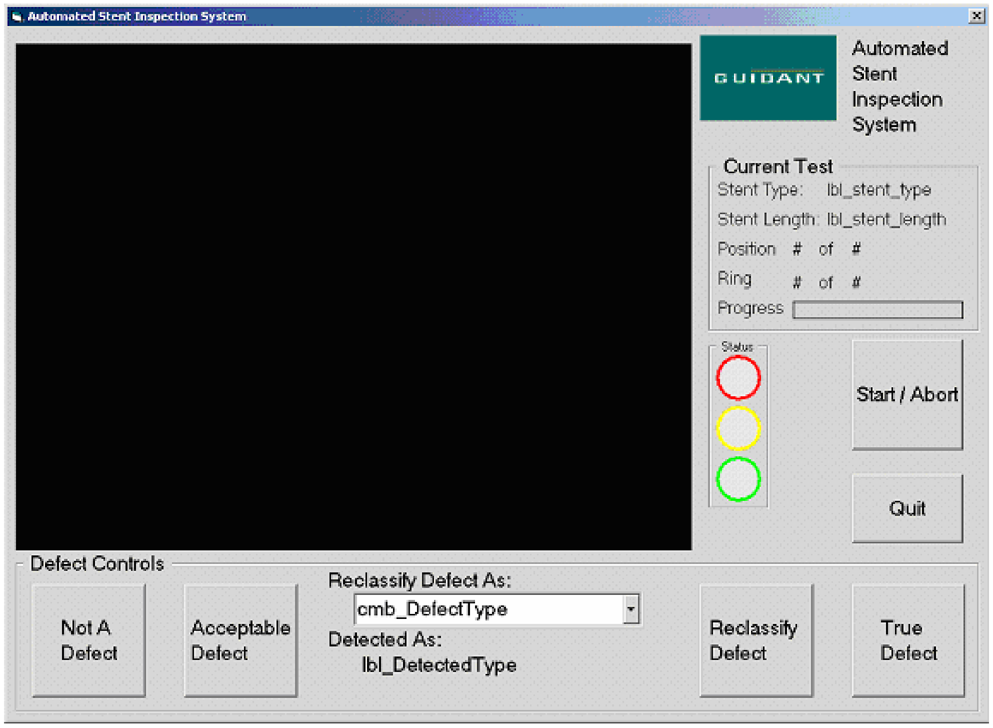


Figure 38: User Interface Main Screen

When then inspection is first started, the user interface initialization window is the first to load. It consists of a progress bar and a status update. The system is checking to make sure that the camera and CNC fixture are properly connected and that they are communicating with each other. The initialization screen is shown in Figure 39.

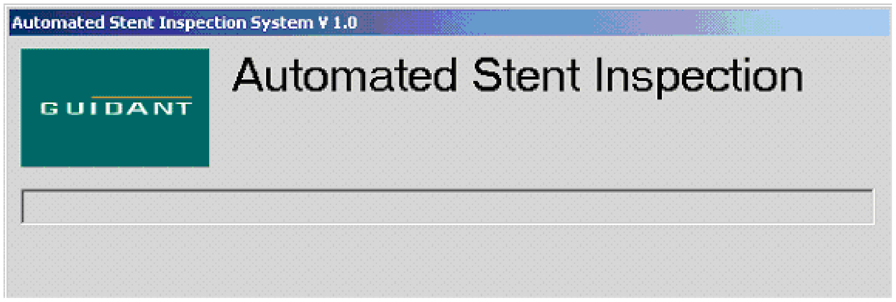

Automated Stent Inspection System V 1.0
GUIDANT
Automated Stent Inspection

Figure 39: Initialization Window

The Start/Abort button on the user interface is used to begin a new inspection or halt an inspection in process. To start an inspection, the operator needs to load the stent and mandrel into the CNC chuck. They can then press the Start/Abort button and the inspection will begin. Pressing the button during an inspection results in the inspection being stopped, the CNC fixture being returned to the "Home" position, and the tailstock being opened.

As the inspection is running, the user interface is displaying images of the feature that is being inspected. This is updated five times a second, so the images are changing too fast for the operator to see any details. If a defect is found, the inspection freezes. This results in an image of the defect and feature that it was found on being displayed on the user interface. The operator can then view the potential defect and save the image.

Before the inspection can be resumed, the operator is required to either accept or reject the part. To do this, the operator can select one of four options. First, the operator can accept the inspection's assessment of the stent. This means that the operator agrees that the potential defect is serious enough to fail the stent and that it has been classified correctly. For this option, the operator must push the "True Defect" button. The second option that the operator has is to reject the stent, but reclassify the defect. This means that the inspection has correctly identified a defect but incorrectly classified it. To do this, the operator must select a new defect from the drop box and push the "Reclassify Defect" button. The third option is for the operator to push the "Acceptable Defect" button. This means that the potential defect is a defect, but that it is small enough to be disregarded. The feature is reclassified as passing and the inspection continues. Lastly, the operator can disagree with the inspection. This means that the feature in question contains no actual defect and the inspection misidentified it. To do this, the operator must push the "Not a Defect" button.

After an inspection passes, fails, or is aborted, a report is generated to contain the results of the inspection. This report is in the form of an MS Access Database and contains all the information about of the potential defects that have been classified as "Acceptable Defect", "True Defect", "Not a Defect", or reclassified. This makes it easier to monitor the accuracy of the inspection. It also facilitates the reconfiguring the system. If one part of the inspection is excessively misidentifying defects, it can be changed without affecting the other parts of the inspection. The report is also useful because it allows the operator to observe patterns within a batch of stents. For example, if numerous stents within a batch are all found to be failing for excessive clumping, it will show up in the report and make it easier for the operator to observe this trend and report the problem to production. Generating an inspection report in the form of a MS Access Database makes it easier for Guidant Corporation to track the accuracy of the stent inspection and production.

6. Bibliography

1. “A Few Quotes From... Silicon Valley History.” Gromov, Gregory.
. Retrieved November 6, 2004, from the World Wide Web:
http://www.netvalley.com/svhistory.html

2. “Fred Terman, the Father of Silicon Valley.” (May 1985) Tajnai, Carolyne. Retrieved November 7, 2004, from the World Wide Web:
http://www.netvalley.com/archives/mirrors/terman.html

3. “Guidant Gains Immediate Access to Drug Eluting Stent Market.” (Feb 24, 2004). Retrieved November 7, 2004, from the World Wide Web:
http://www.guidant.com/news/400/web_release/nr_000451.shtml

4. “Stent.” Retrieved November 9, 2004, from the World Wide Web:
http://en.wikipedia.org/wiki/Stent

5. “Quality Every Step of the Way.” (Summer 2004). Retrieved November 9, 2004, from the World Wide Web:
http://www.lifebeatonline.com/winter2004/news.shtml

6. “The New ‘Coated’ Stents – What’s the Story?” (July 7, 2004). Kulick, Daniel. Retrieved November 9, 2004, from the World Wide Web:
http://www.medicinenet.com/script/main/art.asp?articlekey=20723

7. **“**Welcome to Silicon Valley Today.” Unknown Author. Retrieved November 7, 2004, from the World Wide Web: http://www.svtoday.com/svt/newspage.htm

8. Cognex Corporation. Retrieved November 7, 2004, from the World Wide Web:
http://www.cognex.com/

9. Machine Vision Online Community. Retrieved November 7, 2004, from the World Wide Web: http://www.machinevisiononline.org/

10. Advanced Illumination Online Catalog. Retrieved November 7, 2004, from the World Wide Web:
http://www.advancedillumination.com/category/RL.html#dark_field

11. “Machine Vision Inspection” Retrieved November 7, 2004, from the World Wide Web:
http://www.it.northropgrumman.com/home.asp?bid=6083

12. “Guidant Corporation: About Us” Retrieved November 7, 2004, from the World Wide Web:
http://www.guidant.com/about/

7. Appendix A: Background Information

A significant part of our background research was learning about our project site and sponsor company. Completing this research gave us a better understanding of the environment that we would be working in and the type of products produced by Guidant Corporation. By understanding the goals of our sponsor company, we were able to define our project objectives to meet the goals of the entire company, our project liaison, and project advisors.

7.1. Silicon Valley

"Silicon Valley is the only place on *Earth* not trying to figure out how to become **Silicon Valley**." (Robert Metcalfe, 1) Today Silicon Valley is one of the most famous industrial parks in the world. It houses many of the world's top high-tech industries and is famous for the production of semiconductor chips.

The valley that is contained between the San Francisco Bay, Santa Cruz Mountains, and the Coast Range hasn't always been the center of industry that is today. In the early 1900's, the valley was known as the Valley of Heart's Delight and it was famous for its fruit orchards, not its silicon chips. It took the vision of one man, Professor Frederick Emmons Terman of Stanford University, to turn the Valley of Heart's Delight into Silicon Valley (Tajnai, 2).

During the 1930's Prof. Terman was a professor in the Department of Electrical Engineering at Stanford University. He became concerned with the lack of employment opportunities for graduates of the Stanford Engineering Program (Tajnai, 2). To solve this problem, he began plans to establish an industry for radio technology in Stanford's backyard. This industry would provide employment for Stanford's graduates and work in the growing field of radios.

Silicon Valley was officially founded in 1946 (Tajnai, 2). Prof. Terman and Stanford worked together to create a West Coast center of innovation. Their goal was to create an industrial park that could compete with similar centers on the East Coast and provide economic development to the area. Stanford began leasing the off-campus lands that it owned in the 1950's and in 1951; Varian Associates signed the first lease. Varian also moved into the first industrial building in 1953. Soon after, companies like Eastman Kodak, General Electric, Admiral Corporation, Shockley Transistor Laboratory of Beckman Instruments, Lockheed, Hewlett-Packard, and Preformed Line Products followed. Hewlett-Packard is especially important to Silicon Valley's history, as Terman was responsible for bringing the two co-founders of HP together. Stanford Electrical Engineering graduates Hewlett and Packard worked very closely with Terman to form HP and Silicon Valley (Tajnai, 2). These companies, along with others started by Stanford graduates, helped to form the core of what is present day Silicon Valley.

Today, Silicon Valley plays host to some of the World's largest and most successful hi-tech companies. With companies like Netscape, National Semiconductor, Intel, Adobe, Adaptec, Guidant Corporation, and Computer Curriculum Corporation, as well as universities like Stanford University, San Jose

State, and Santa Clara University, Silicon Valley has become a synonym for technology and industrial efficiency. (Unknown, 5)

7.2. Guidant Corporation

Since its incorporation in 1994, Guidant Corporation has been a pioneer in the design and production of cardiovascular medical products. It is located all over the world and has Corporate Headquarters in Indianapolis, Japan, Belgium, Hong Kong, Canada, Brazil, Austria, Switzerland, and the Czech Republic. They also have Operating Locations throughout the United States, Puerto Rico, and Ireland. The company has been split into four business groups. By dividing the company into these four specific business groups, Guidant has been able to maintain the personal and innovative atmosphere of a small company, while growing into a much larger one. The business groups are able to work intimately with customers and provide immediate responses to their needs.

Since 1994, Guidant has grown into a $3.6 billion company and now employs 12,000 employees. These are all a product of the success that each business group is having within the Cardiac Surgery, Endovascular Solutions, Cardiac Rhythm Management, and Vascular Intervention markets. (Unknown, 12)

Recently, Guidant has increased their production and market share by entering in an agreement with Cordis Corporation, a Johnson-Johnson company. In February 2004, the two companies agreed to work together to design, manufacture, and promote Cordis' CYPHER™ Sirolimus-eluting Coronary Stent. They will combining Cordis' CYPHER Stent product line with Guidant's MULTI-LINK VISION® Stent Delivery System to more efficiently design, produce, and inspect drug eluting stents. This partnership gives Guidant access to the US drug eluting stent market, a market that is expected to generate close to $3 billion in revenue. (Unknown, 3)

This partnership has created a feeling of optimism within Guidant Corporation. This feeling was recently echoed by Dana G. Mead, Jr., the president Guidant's Vascular Intervention Business Group.

'"We are enthusiastic about expanding our product offering with a drug eluting stent that has consistently demonstrated exceptional clinical results. In four years of clinical use in over half a million patients, the CYPHER Stent has proven to be safe and effective. The combination of our market-leading metallic stent – the MULTI-LINK VISION® Coronary Stent System – and an expansive product portfolio will allow Guidant to provide a full range of proven interventional therapies to physicians and patients," said Dana G. Mead, Jr., president, Guidant Vascular Intervention.' (Unknown, 3)

By entering a partnership with Cordis Corporation, one of the country's leaders in stent research and production, Guidant is expanding their market share and increasing their revenue. This increases Guidant's contributions to the medical industry and helps to continue the prevention of cardiac disease in the United States.

8. Appendix B: Schedule

Monday	Tuesday	Wednesday	Thursday	Friday
January 3 • Trial Run to Guidant (All) • Arrange to Meet Liaison (All)	**January 4** • Tour of Facility (All) • Introductory Meeting with Liaison (All)	**January 5** • Study Manual Stent Inspection Process (All)	**January 6** • DVT Software Tutorials (All)	**January 7** • DVT Software Tutorials (All) • Introduction to Vision Console (All)
January 10 • Mandatory CBT Training (All)	**January 11** • Mandatory Haz Comm Training (All) • Create Schedule (All)	**January 12** • Review of Existing Code (Jay/Ben) • Update Proposal (Erik)	**January 13** • Review of Existing Code (Jay/Ben) • Update Proposal (Erik)	**January 14** • Review of Existing Code (Jay/Ben) • Complete Lit Review of Final Report (Erik)
January 17 • Automated Stent Load/Unload Function (Jay/Ben) • Outline of Procedure for Final Report (Erik)	**January 18** • Automated Stent Load/Unload Function (Jay/Ben) • Outline of Procedure for Final Report (Erik)	**January 19** • Automated Stent Load/Unload Function (Jay/Ben) • Update Procedure for Final Report (Erik)	**January 20** • Automated Stent Load/Unload Function (Jay/Ben) • Update Procedure for Final Report (Erik)	**January 21** • Code review 1 for Load/Unload Function (All) • Revision 1 Procedure for Final Report (All)
January 24 • Automated Stent Load/Unload Function (Jay/Ben) • Outline of Procedure for Final Report (Erik)	**January 25** • Automated Stent Load/Unload Function (Jay/Ben) • Outline of Procedure for Final Report (Erik)	**January 26** • Automated Stent Load/Unload Function (Jay/Ben) • Update Procedure for Final Report (Erik)	**January 27** • Automated Stent Load/Unload Function (Jay/Ben) • Update Procedure for Final Report (Erik)	**January 28** • Code review 2 for Load/Unload Function (All) • Revision 2 Procedure for Final Report (All)
January 31 • Black Space Inspection Module (Ben) • Strut Inspection	**February 1** • Black Space Inspection Module (Ben) • Strut Inspection	**February 2** • Black Space Inspection Module (Ben) • Strut Inspection	**February 3** • Black Space Inspection Module (Ben) • Strut Inspection Module (Erik)	**February 4** • Code review 1 for Load/Unload Function (All) • Code review

Module (Erik) • Stent Alignment Module (Jay)	Module (Erik) • Stent Alignment Module (Jay)	Module (Erik) • Stent Alignment Module (Jay)	• Stent Alignment Module (Jay) • Update Procedure for Final Report (Erik)	1 for Strut Inspection Module (All) • Code review for Black Space Inspection Module (All) • Revision 3 Procedure for Final Report (All)
February 7 • Black Space Inspection Module (Ben) • Strut Inspection Module (Erik) • Stent Alignment Module (Jay)	**February 8** • Black Space Inspection Module (Ben) • Strut Inspection Module (Erik) • Stent Alignment Module (Jay)	**February 9** • Black Space Inspection Module (Ben) • Strut Inspection Module (Erik) • Stent Alignment Module (Jay)	**February 10** • Black Space Inspection Module (Ben) • Strut Inspection Module (Erik) • Stent Alignment Module (Jay) • Update Procedure for Final Report (Erik)	**February 11** • Code review 2 for Load/Unload Function (All) • Code review 2 for Strut Inspection Module (All) • Code review for Black Space Inspection Module (All) • Revision 4 Procedure for Final Report (All)
February 14 • System Testing (Ben) • Sensor Updates (Jay) • Outline of Results and Analysis for Final Report (Erik)	**February 15** • System Testing (Ben) • Sensor Updates (Jay) • Outline of Results and Analysis for Final Report (Erik)	**February 16** • System Testing (Ben) • Sensor Updates (Jay) • Write Results and Analysis for Final Report (Erik) • Update Presentation (Erik)	**February 17** • System Testing (Ben) • Sensor Updates (Jay) • Write Results and Analysis for Final Report (Erik) • Update Presentation (Erik)	**February 18** • Sensor Accuracy Review (All) • Revision 1 of Results and Analysis for Final Report (All) • Revision of Presentation (All)
February 21 • Final System Testing (All) • Update Results and Analysis for Final Report (Erik)	**February 22** • Final System Testing (All) • Update Results and Analysis for Final Report (Erik)	**February 23** • Update Presentation (Jay) • Users Manual (Ben) • Revision 2 of Results and Analysis for	**February 24** • User Interface (Jay) • Write Conclusions and Recommendations for Final Report (Erik) • Users Manual	**February 25** • Write Executive Summary (Erik) • Revision of Users Manual (All) • Revision of Final Report

		Final Report (All)	(Ben)	(All) • User Interface (Jay)
February 28 • Presentation Rehearsal (All)	**March 1** • Presentation Rehearsal (All)	**March 2** • Final Presentation at Guidant (All)	**March 3** • Final Presentation at SRI (All)	**March 4** • Final Day at Guidant (All)

9. Appendix C: G-Code Definition

What is G Code?

G codes are the specific syntax used for preparatory functions such as the type of movement, acceleration, and work in conjunction with F & M words. In the following table is a summary of the G Codes and their associated F and M words.

Command	Description	Comments
G0	Rapid Traverse	Used to rapidly move from point to point, where the path taken is not critical.
G1	Linear Interpolation	Interpolates motion and synchronizes it on all axes commanded to move.
G4	Dwell	Pause for the number of seconds indicated in the next F word.
G64	Set Linear Acceleration Mode	The acceleration is linear and constant over the time allotted versus sinusoidal.
G68	Acceleration/Deceleration Rate Based	The acceleration and deceleration will vary in accordance to the desired rate of movement, rather than time to move.
G82	Clear Software Home	Will eliminate any software coordinate systems current active and use hardware coordinates.
G90	Absolute Programming Mode	Movements in this mode move to a specific place in the coordinate plane based upon a fixed reference point.
G91	Incremental Programming Mode	Movements in this mode move based on the current position rather than a fixed reference.
G109	Deceleration to Zero Velocity	During consecutive movements the fixture must come to a complete stop rather than chain movements together.
M0	Program Stop	The motion will stop at this point until the Cycle Start button is pressed
M1	Optional Program Stop	If optional stops are enabled this will halt motion until the Cycle Start button is pressed, otherwise it has no effect.
M2	End of Program	This will terminate the program.
F	Linear Feed Rate (When used with a G0 or G1)	This is the velocity in Inches/Minute that the fixture will accelerate towards as a maximum speed.
F	Dwell Time (When used with a G4)	This is the time in seconds for the system to pause after a G4 command.

10. Appendix D: DVT SoftSensors

What are DVT SoftSensors?

DVT SoftSensors are tools used by FrameWork to manipulate and inspect an image from a DVT camera. The basic tools and their purposes are described below.

Sensor Type	Description
Translational SoftSensor	Translational SoftSensors are used primarily to locate objects to be inspected in the Sampled Image Display.
Rotational SoftSensor	Rotation SoftSensors are designed to compute the angle of rotation of objects in the image. Like Translation SoftSensors, Rotation SoftSensors allow for different positions of an object without failing the part for being in an unexpected position or location.
Intensity SoftSensor	Checks contrast and bright area and compare to defined threshold values for maximum and minimum values for each of these measurements.
EdgeCount SoftSensor	Used to count edge transitions based on changing intensity levels. The SoftSensor outputs the total number of edges in the Results Table and this output can be limited with Warn and Pass Parameters.
FeatureCount SoftSensor	Used to count regions known as features of light and/or dark pixels along the SoftSensor path. The SoftSensor outputs the total number of features in the Results Table and this output can be limited with Warn and Pass Parameters.
Measurement SoftSensor	Measurement SoftSensors are used to compute the distance between two points of origin or the area of a selected region.
Math SoftSensors	Use reference points to calculate *Distance, Intersection, Angle, Midpoint, Midline, Line Through Two Points, Perpendicular Line, Scale Factor, Coordinate Transform, Coordinate System,* and *New Coordinate Transform.*
Readers	The Reader SoftSensors are the 2-D reader (good for DataMatrix, Vericode and Snowflake), bar code reader, and the OCR reader. They are designed to read labels, codes, etc. to verify that the right part is in front of the SmartImage Sensor or simply to decode the information.
Blob SoftSensor	Blobs are areas of connected pixels of similar intensity. The Blob Generator will locate these blobs of either dark or light pixels and allow further analysis.
Blob Selector	Analyze blobs found by a given Blob SoftSensor and compare against predefined criteria including area, perimeter, angular rotation, and center of mass.
Template Match	Searches within a defined *Scan Region* for a minimum

SoftSensor	error match to the *Template Region* that was learned previously
ObjectFind SoftSensor	Similar to *Template Match*, it searches within a defined area for a known object.
Segmentation	Similar to a Blob SoftSensor, Segmentation is the process of separating pixels into distinct blobs. Each set has its own characteristics including color, pixel size, and center.
Smart Link	The SmartLink SoftSensor is used to send information from a SmartImage Sensor to the SmartLink Display device. The information from this SoftSensor is displayed in a Table in the SmartLink display. The table must be configured on both in the FrameWork and in SmartLink. When defining a table in the SmartLink, the IP address of the SmartImage sensor and a Table ID must be defined.
Script	Scripts are basically programmable tools used in SmartImage Sensors. Each script is designated as a class that can contain a number of static user-defined functions, with one required method that will be the first to execute when the script is initialized (the name of this method is different between a Foreground Script and a Background Script and will be detailed below). They are designed to be fully customizable to the application's needs. Unlike other parameters within FrameWork, Scripts have no predefined purpose. They are created as an empty tool that is shaped to perform the required tasks according to the user needs.

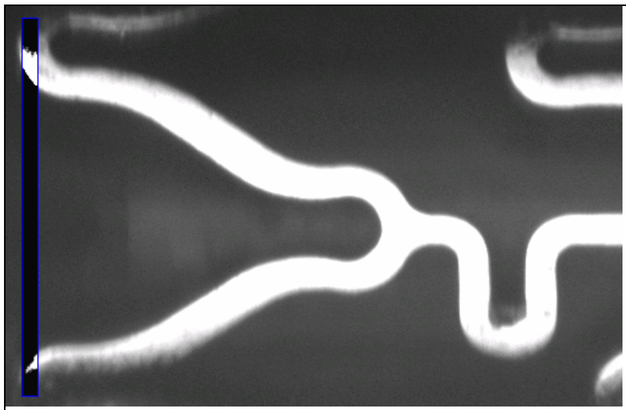
11. Appendix E: Software Description

The individual sensor packages that were used to automate the stent inspection are shown below. There are three different classifications for these sensor packages. They are Alignment, Webbing, and Clumping. Alignment sensors are used to find the edges and inspection starting points within the stent geometry. Webbing sensors are used to scan the area between stent struts for defects. Clumping sensors are used to scan the actual stent surface and edges for defects. The information needed to recreate each sensor package is displayed in the tables below. All sensors are named using the following convention.

Sensor Naming Convention: type_ feature#_ classification _name
Type:
Blob Sensor = blo
Intensity Sensor = int
Template Sensor = tmp
Blob Selector = bls
Feature #:
Feature 1, 5 = 1
Feature 2, 4, 8, 10 = 2
Feature 3 = 3
Feature 6, 12 = 6
Feature 7, 9 = 7
Feature 11 = 11
Classification:
Alignment = a
Webbing = w
Clumping = c
Threshold = t

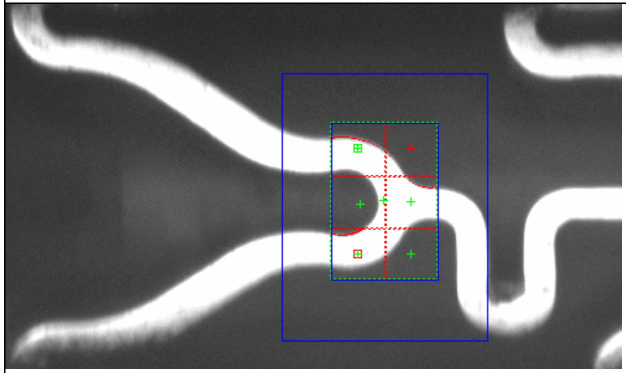
11.1. *Stent Alignment*

The first step in the inspection algorithm is to properly align the stent for inspection.



int_1_a_edge_seek		
Sensor Shape	**X**	**Y**
Point 0	114	66
Point 1	127	66
Point 2	127	378
Point 3	114	378
Point 4	114	66
Sensor Thresholds	**Type**	**Value**
Threshold Level	Fixed Value	65
Min Bright Area	Warn/Pass	2 %

Sensor Type: Intensity Soft Sensor
Package Type: Alignment
Purpose: This sensor is the first sensor that we use. As the CNC fixture is slowly moving the stent horizontally, this sensor scans the area. The sensor is scanning the area for bright pixels. When the percentage of bright pixels in the search area reaches 2%, the sensor will return a PASS. It returns a FAIL otherwise
Pass: A PASS signifies that the edge of the stent has been found. It signals to the CNC fixture to cease all horizontal movement and begin slowly rotating the stent.
Fail: A FAIL signifies that the edge has not been found. It signals to the CNC fixture to continue horizontal movement.

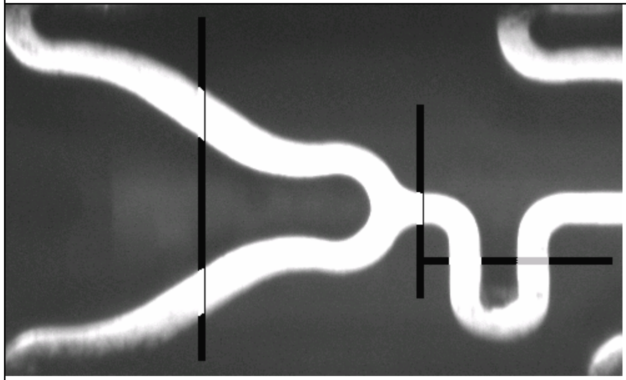


tmp_1_a_fidseek		
Sensor Shape	**X**	**Y**
Outer Rectangle	330	136
	496	351

Sensor Type: Template Match Soft Sensor
Package Type: Alignment
Purpose: This sensor searches for the inspection starting point after the stent edge has been found. As the stent rotates, it scans the search area (outer box) for a match to the defined template (inner box). It will return a PASS when a feature that matches the template is found. The sensor allows for an error of 10% for this template matching. It return a FAIL if no matches are found.

Inner Rectangle	370	176
	456	302
Sensor Thresholds	**Type**	**Value**
Threshold Level	Fixed Value	65
Max Total Area	Warn/Pass	10%

Pass: A PASS signifies that the starting point has been found. It signals to the CNC fixture to cease rotation.
Fail: A FAIL signifies that the starting point has not been found. It signals to the CNC fixture to continue rotation.



Sensor Shape	**X**	**Y**	**Positional Reference**
blo_1_a_topstr	270	88	blo_1_a_key
	275	232	
blo_1_a_botstr	270	230	blo_1_a_key
	275	362	
blo_1_a_fid	445	158	None
	450	312	
blo_1_a_key	445	280	blo_1_a_fid
	601	285	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

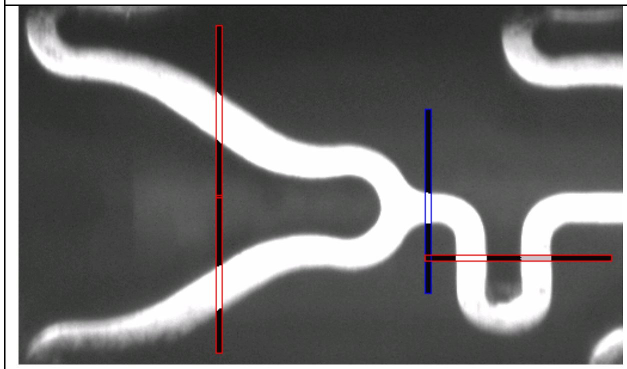
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors make the fine adjustments required to center the starting point. This adjustment runs in two stages. First it performs a coarse adjustment based on the vertical Blob Sensors. Then, it performs a fine adjustment based on both the horizontal and vertical Blob Sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for centering the starting point.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes software reset and returns to the Template Sensor described above.

11.2. Feature Inspection: Beginning Ring

The inspection starts by incrementing through the features of the first ring. A Webbing Inspection and a Clumping Inspection is performed for each feature. After each inspection is performed on a feature, the stent is rotated 0.01963 inches (based on 0.075 inch outer diameter) to the next feature. Once all 12 features have been inspected, the stent is moved 0.05286 inches horizontally to focus on the next ring. Blob Sensors from above are then used to realign the stent to the next starting point.

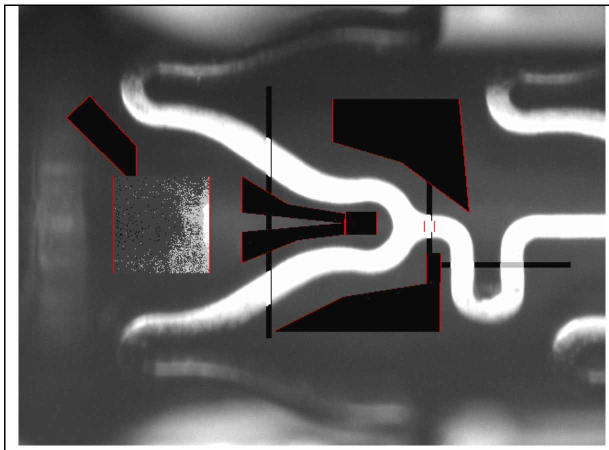
Feature 1

Feature 1 occurs three times within Ring 1. It uses the same reference and inspection sensors as Feature 5.



Sensor Shape	**X**	**Y**	**Positional Reference**
blo_1_a_topstr	270	88	blo_1_a_key
	275	232	
blo_1_a_botstr	270	230	blo_1_a_key
	275	362	
blo_1_a_fid	445	158	None
	450	312	
blo_1_a_key	445	280	blo_1_a_fid
	601	285	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

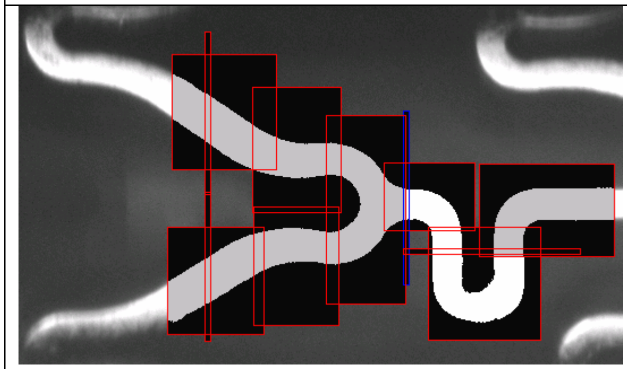
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Shape	X	Y	Positional Reference
blo_1_w_1			blo_1_a_botstr
Point 0	356	228	
Point 1	385	228	
Point 2	385	246	
Point 3	356	246	
Point 4	356	228	
blo_1_w_2			blo_1_a_botstr
Point 0	355	237	
Point 1	243	225	
Point 2	243	200	
Point 3	294	217	
Point 4	356	227	
Point 5	355	237	
blo_1_w_3			blo_1_a_botstr
Point 0	355	237	
Point 1	355	249	
Point 2	305	255	
Point 3	243	265	
Point 4	243	247	
Point 5	355	237	
blo_1_w_4			blo_1_a_key
Point 0	342	148	
Point 1	342	102	
Point 2	480	102	
Point 3	491	225	
Point 4	416	170	
Point 5	342	148	
blo_1_w_5			blo_1_a_key
Point 0	444	270	
Point 1	460	270	
Point 2	460	301	
Point 3	444	301	
Point 4	444	270	
blo_1_w_6			blo_1_a_key
Point 0	459	301	
Point 1	459	355	
Point 2	279	355	
Point 3	352	318	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 4	459	301	
blo_1_w_7			blo_1_a_topstr
Point 0	128	197	
Point 1	128	153	
Point 2	77	98	
Point 3	52	122	
Point 4	128	197	
int_1_t_dark			blo_1_a_botstr
Point 0	103	186	
Point 1	208	186	
Point 2	208	291	
Point 3	103	291	
Point 4	103	186	
int_1_t_light			blo_1_a_fid
Point 0	442	235	
Point 1	453	235	
Point 2	453	246	
Point 3	442	246	
Point 4	442	235	

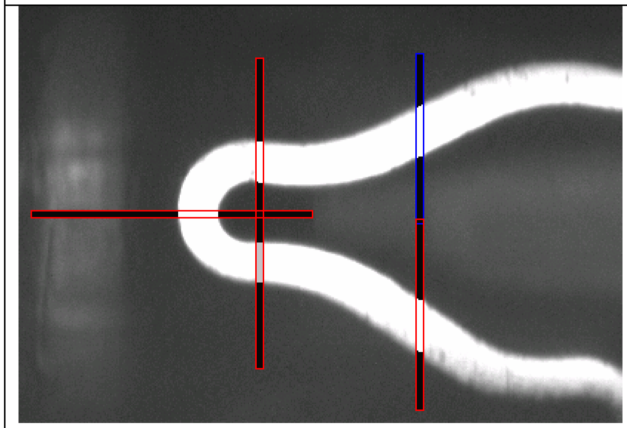


Sensor Shape	X	Y	Positional Reference
blo_1_c_ts	312	137	blo_1_a_botstr
	390	248	
blo_1_c_ts2	241	108	blo_1_a_botstr
	333	210	
blo_1_c_bs	313	243	blo_1_a_botstr
	388	348	
blo_1_c_bs2	237	267	blo_1_a_botstr
	322	356	
blo_1_c_ms	428	204	blo_1_a_botstr
	508	264	
blo_1_c_ms2	512	205	blo_1_a_key
	628	287	
blo_1_c_crown	377	162	blo_1_a_botstr
	447	329	
blo_1_c_kh	467	261	blo_1_a_botstr
	566	361	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

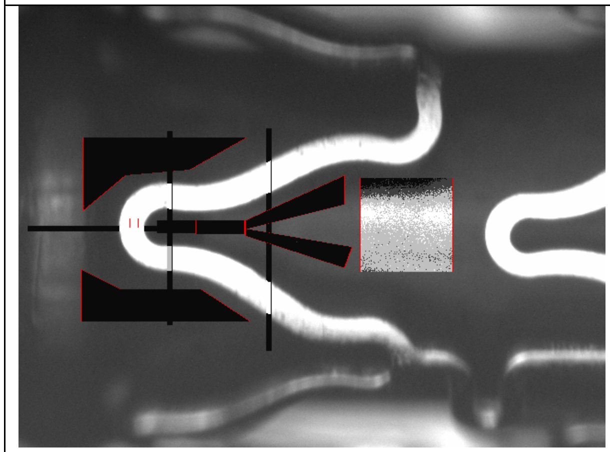
Feature 2

Feature 2 occurs three times within Ring 1. It uses the same reference and inspection sensors as Features 4, 8, and 10.



Sensor Shape	X	Y	Position Reference
blo_2_a_topstr	270	100	None
	275	240	
blo_2_a_botstr	270	245	blo_2_a_topstr
	275	374	
blo_2_a_neck	162	136	blo_2_a_botstr
	167	346	
blo_2_a_crown	10	239	blo_2_a_neck
	200	244	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

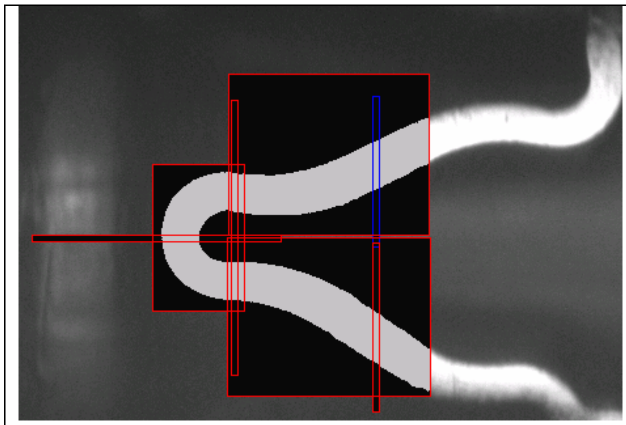
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed

Sensor Shape	X	Y	Position Reference
blo_2_w_1			blo_2_a_crown
Point 0	156	233	
Point 1	192	233	
Point 2	192	246	
Point 3	156	246	
Point 4	156	233	
blo_2_w_2			blo_2_a_crown
Point 0	193	233	
Point 1	246	233	
Point 2	246	247	
Point 3	193	247	
Point 4	193	233	
blo_2_w_3			blo_2_a_crown
Point 0	247	249	
Point 1	355	284	
Point 2	363	262	
Point 3	247	242	
Point 4	247	249	
blo_2_w_4			blo_2_a_crown
Point 0	247	233	
Point 1	355	184	
Point 2	355	215	
Point 3	247	242	
Point 4	247	233	
blo_2_w_5			blo_2_a_crown
Point 0	90	160	
Point 1	70	222	
Point 2	117	183	
Point 3	187	177	
Point 4	247	143	
Point 5	90	160	
blo_2_w_6			blo_2_a_crown
Point 0	68	286	
Point 1	111	308	
Point 2	199	308	
Point 3	251	342	
Point 4	68	342	
Point 5	68	286	
int_2_t_dark			blo_2_a_crown
Point 0	372	187	
Point 1	473	187	
Point 2	473	288	
Point 3	372	288	
Point 4	372	187	
int_2_t_light			blo_2_a_crown
Point 0	121	231	
Point 1	130	231	
Point 2	130	240	
Point 3	121	240	
Point 4	121	231	

and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

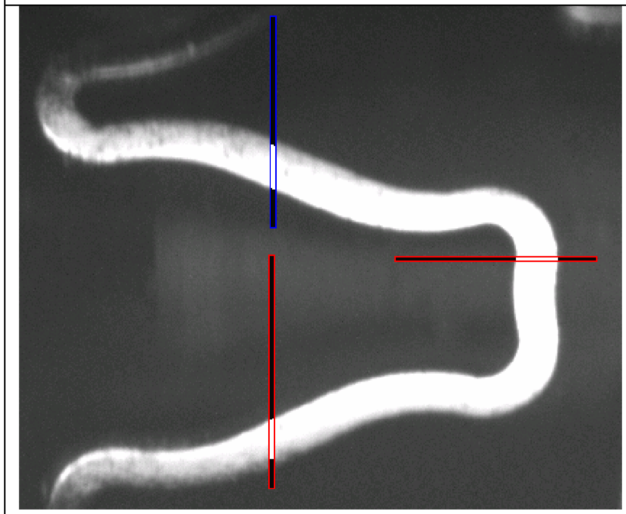


Sensor Shape	X	Y	Position Reference
blo_2_c_ts	160	116	blo_2_a_crown
	313	239	
blo_2_c_bs	159	251	blo_2_a_crown
	314	357	
blo_2_c_crown	102	185	blo_2_a_crown
	172	297	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

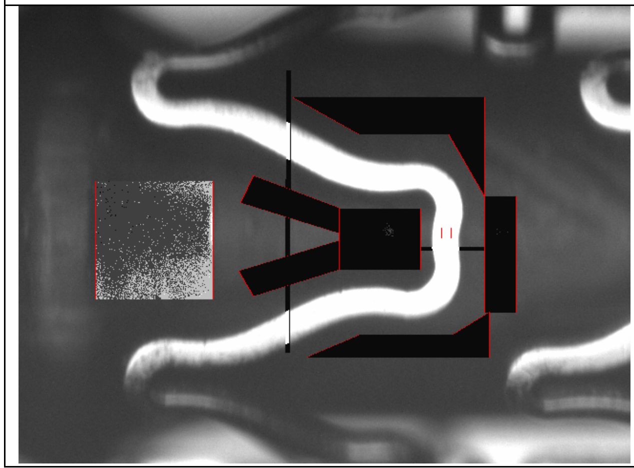
Feature 3

Feature 3 occurs three times within Ring 1. It uses the same reference sensors as Feature 11.



Sensor Shape	X	Y	Position Reference
blo_11_a_topstr	280	68	None
	284	214	
blo_11_a_botstr	279	233	blo_11_a_topstr
	283	394	
blo_11_a_crown	366	234	blo_11_a_crown
	505	237	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

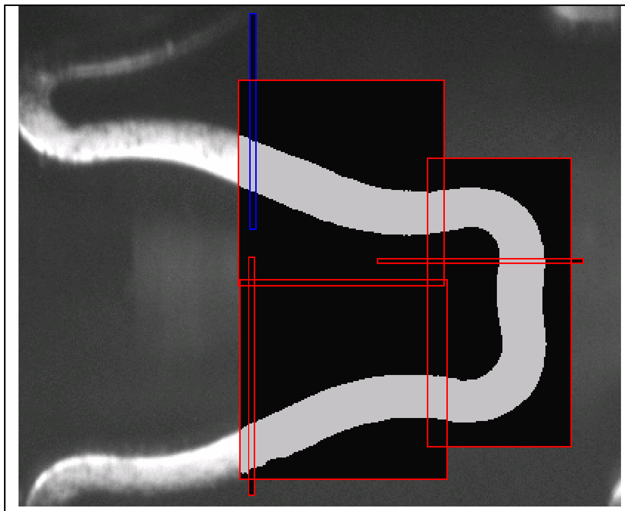
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free

Sensor Shape	X	Y	Position Reference
blo_3_w_1			blo_11_a_crown
Point 0	335	213	
Point 1	415	213	
Point 2	415	276	
Point 3	335	276	
Point 4	335	213	
blo_3_w_2			blo_11_a_crown
Point 0	335	276	
Point 1	245	304	
Point 2	230	278	
Point 3	335	246	
Point 4	335	276	
blo_3_w_3			blo_11_a_crown
Point 0	335	212	
Point 1	245	178	
Point 2	232	206	
Point 3	335	238	
Point 4	335	212	
blo_3_w_4			blo_11_a_crown
Point 0	487	200	
Point 1	487	96	
Point 2	287	96	
Point 3	355	134	
Point 4	449	134	
Point 5	487	200	
blo_3_w_5			blo_11_a_crown
Point 0	492	200	
Point 1	520	200	
Point 2	520	321	
Point 3	492	321	
Point 4	492	200	
blo_3_w_6			blo_11_a_crown
Point 0	302	368	
Point 1	492	368	
Point 2	492	321	
Point 3	453	345	
Point 4	356	345	
Point 5	302	368	
int_3_t_dark			blo_11_a_crown
Point 0	80	184	
Point 1	203	184	
Point 2	203	307	
Point 3	80	307	
Point 4	80	184	
int_3_t_light			blo_11_a_crown
Point 0	442	233	
Point 1	452	233	
Point 2	452	243	
Point 3	442	243	
Point 4	442	233	

of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

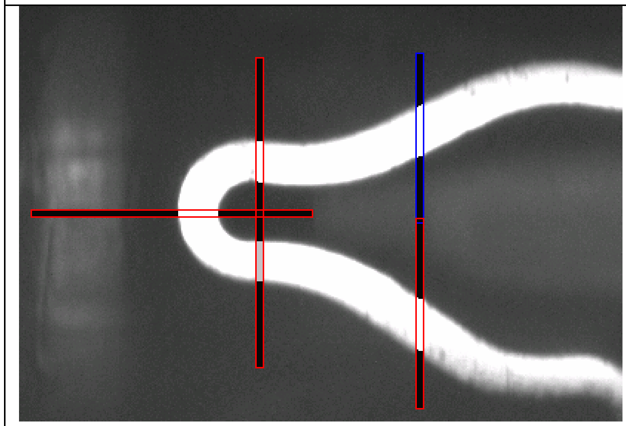


Sensor Shape	X	Y	Position Reference
blo_3_c_ts	273	248	blo_11_a_crown
	413	383	
blo_3_c_bs	272	113	blo_11_a_crown
	411	252	
blo_3_c_crown	400	166	blo_11_a_crown
	497	361	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

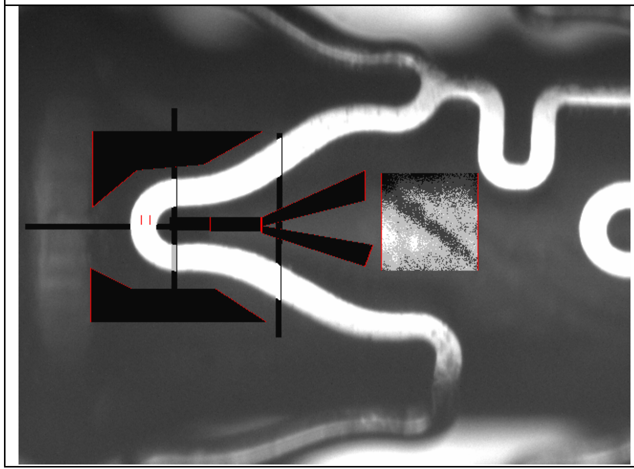
Feature 4

Feature 4 occurs three times within Ring 1. It uses the same reference and inspection sensors as Features 2, 8, and 10.



Sensor Shape	X	Y	Position Reference
blo_2_a_topstr	270	100	None
	275	240	
blo_2_a_botstr	270	245	blo_2_a_topstr
	275	374	
blo_2_a_neck	162	136	blo_2_a_botstr
	167	346	
blo_2_a_crown	10	239	blo_2_a_neck
	200	244	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

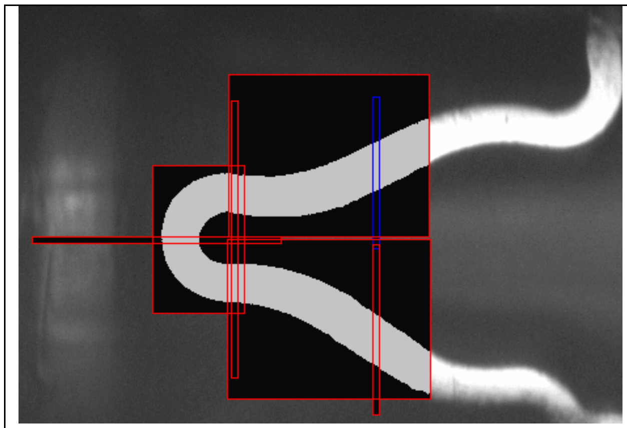
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed

Sensor Shape	X	Y	Position Reference
blo_2_w_1			blo_2_a_crown
Point 0	156	233	
Point 1	192	233	
Point 2	192	246	
Point 3	156	246	
Point 4	156	233	
blo_2_w_2			blo_2_a_crown
Point 0	193	233	
Point 1	246	233	
Point 2	246	247	
Point 3	193	247	
Point 4	193	233	
blo_2_w_3			blo_2_a_crown
Point 0	247	249	
Point 1	355	284	
Point 2	363	262	
Point 3	247	242	
Point 4	247	249	
blo_2_w_4			blo_2_a_crown
Point 0	247	233	
Point 1	355	184	
Point 2	355	215	
Point 3	247	242	
Point 4	247	233	
blo_2_w_5			blo_2_a_crown
Point 0	90	160	
Point 1	70	222	
Point 2	117	183	
Point 3	187	177	
Point 4	247	143	
Point 5	90	160	
blo_2_w_6			blo_2_a_crown
Point 0	68	286	
Point 1	111	308	
Point 2	199	308	
Point 3	251	342	
Point 4	68	342	
Point 5	68	286	
int_2_t_dark			blo_2_a_crown
Point 0	372	187	
Point 1	473	187	
Point 2	473	288	
Point 3	372	288	
Point 4	372	187	
int_2_t_light			blo_2_a_crown
Point 0	121	231	
Point 1	130	231	
Point 2	130	240	
Point 3	121	240	
Point 4	121	231	

and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.



Sensor Shape	X	Y	Position Reference
blo_2_c_ts	160	116	blo_2_a_crown
	313	239	
blo_2_c_bs	159	251	blo_2_a_crown
	314	357	
blo_2_c_crown	102	185	blo_2_a_crown
	172	297	

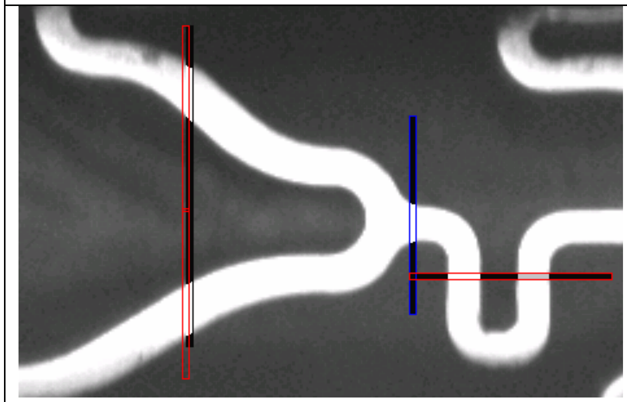
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

11.3. *Feature Inspection: Middle Rings*

The inspection for the middle rings is very similar to the beginning ring. The difference being the types of features found in the rings. It starts each ring by realigning the stent to the starting point and then incrementing through the 12 features.

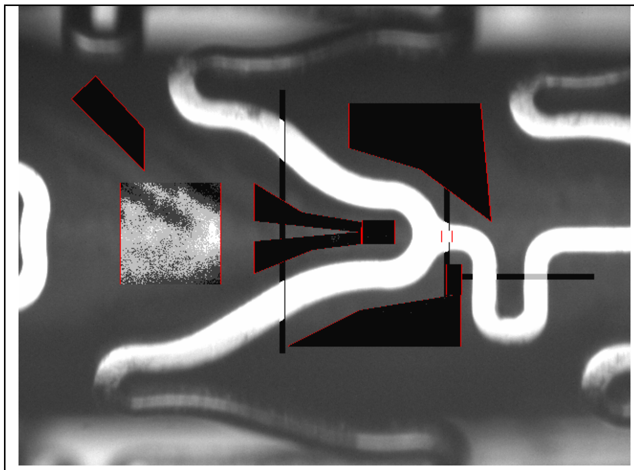
Feature 5

Feature 5 occurs three times within each middle ring. It uses the same reference and inspection sensors as Feature 1.



Sensor Shape	X	Y	Position Reference
blo_1_a_topstr	270	88	blo_1_a_key
	275	232	
blo_1_a_botstr	270	230	blo_1_a_key
	275	362	
blo_1_a_fid	445	158	None
	450	312	
blo_1_a_key	445	280	blo_1_a_fid
	601	285	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

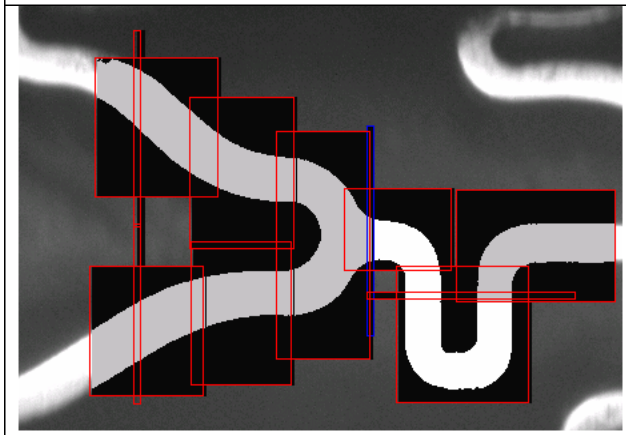
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Shape	X	Y	Positional Reference
blo_1_w_1			blo_1_a_botstr
Point 0	356	228	
Point 1	385	228	
Point 2	385	246	
Point 3	356	246	
Point 4	356	228	
blo_1_w_2			blo_1_a_botstr
Point 0	355	237	
Point 1	243	225	
Point 2	243	200	
Point 3	294	217	
Point 4	356	227	
Point 5	355	237	
blo_1_w_3			blo_1_a_botstr
Point 0	355	237	
Point 1	355	249	
Point 2	305	255	
Point 3	243	265	
Point 4	243	247	
Point 5	355	237	
blo_1_w_4			blo_1_a_key
Point 0	342	148	
Point 1	342	102	
Point 2	480	102	
Point 3	491	225	
Point 4	416	170	
Point 5	342	148	
blo_1_w_5			blo_1_a_key
Point 0	444	270	
Point 1	460	270	
Point 2	460	301	
Point 3	444	301	
Point 4	444	270	
blo_1_w_6			blo_1_a_key
Point 0	459	301	
Point 1	459	355	
Point 2	279	355	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 3	352	318	
Point 4	459	301	
blo_1_w_7			blo_1_a_topstr
Point 0	128	197	
Point 1	128	153	
Point 2	77	98	
Point 3	52	122	
Point 4	128	197	
int_1_t_dark			blo_1_a_botstr
Point 0	103	186	
Point 1	208	186	
Point 2	208	291	
Point 3	103	291	
Point 4	103	186	
int_1_t_light			blo_1_a_fid
Point 0	442	235	
Point 1	453	235	
Point 2	453	246	
Point 3	442	246	
Point 4	442	235	

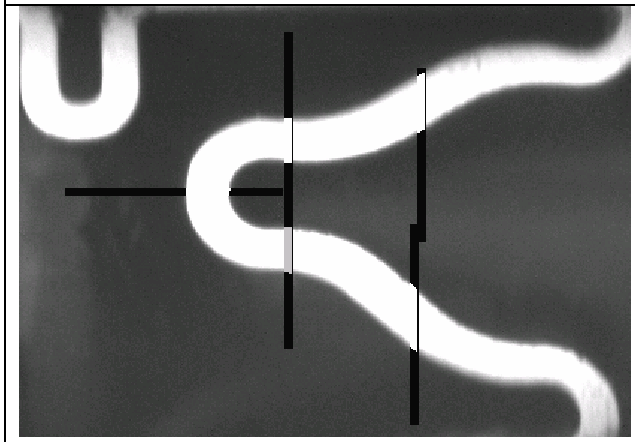


Sensor Shape	X	Y	Position Reference
blo_1_c_ts	312	137	blo_1_a_botstr
	390	248	
blo_1_c_ts2	241	108	blo_1_a_botstr
	333	210	
blo_1_c_bs	313	243	blo_1_a_botstr
	388	348	
blo_1_c_bs2	237	267	blo_1_a_botstr
	322	356	
blo_1_c_ms	428	204	blo_1_a_botstr
	508	264	
blo_1_c_ms2	512	205	blo_1_a_key
	631	267	
blo_1_c_crown	377	162	blo_1_a_botstr
	447	329	
blo_1_c_kh	467	261	blo_1_a_botstr
	566	361	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

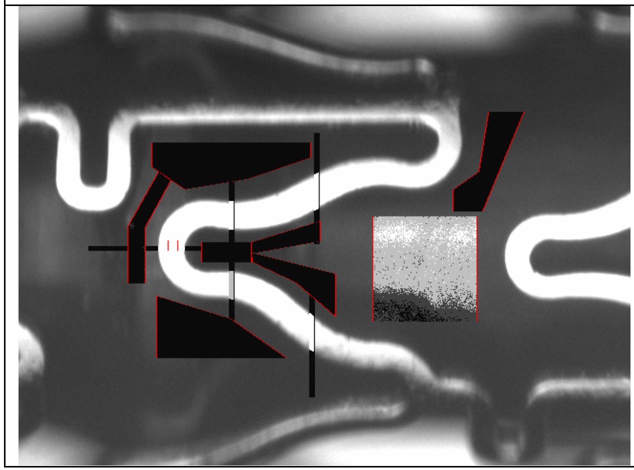
Feature 6

Feature 6 occurs three times within each middle ring. It uses the same reference and inspection sensors as Feature 12.



Sensor Shape	X	Y	Position Reference
blo_6_a_topstr	309	133	None
	314	248	
blo_6_a_botstr	304	275	blo_6_a_topstr
	309	408	
blo_6_a_crown	73	251	blo_6_a_neck
	218	256	
blo_6_a_neck	220	151	blo_6_a_botstr
	225	361	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



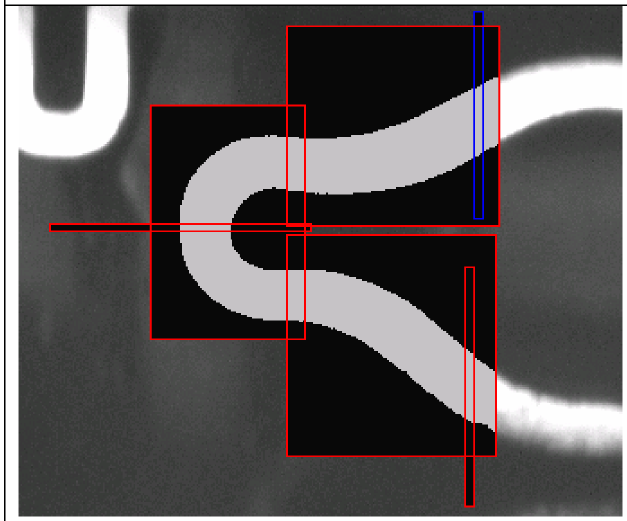
Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed

Sensor Shape	X	Y	Position Reference
blo_6_w_1			blo_6_a_crown
Point 0	196	247	
Point 1	243	247	
Point 2	243	267	
Point 3	196	267	
Point 4	196	247	
blo_6_w_2			blo_6_a_crown
Point 0	243	258	
Point 1	244	267	
Point 2	289	288	
Point 3	331	324	
Point 4	331	280	
Point 5	243	258	
blo_6_w_3			blo_6_a_crown
Point 0	243	247	
Point 1	315	225	
Point 2	315	245	
Point 3	243	258	
Point 4	243	247	
blo_6_w_4			blo_6_a_crown
Point 0	491	111	
Point 1	528	111	
Point 2	485	214	
Point 3	454	214	
Point 4	454	192	
Point 5	481	173	
Point 6	491	111	
blo_6_w_5			blo_6_a_crown
Point 0	139	145	
Point 1	139	168	
Point 2	173	191	
Point 3	240	180	
Point 4	305	158	
Point 5	305	145	
Point 6	139	145	
blo_6_w_6			blo_6_a_crown
Point 0	144	303	
Point 1	144	367	
Point 2	279	367	
Point 3	222	327	
Point 4	144	303	
blo_6_w_7			blo_6_a_crown
Point 0	132	289	
Point 1	114	289	
Point 2	114	230	
Point 3	146	173	
Point 4	159	182	
Point 5	132	231	
Point 6	132	289	
int_6_t_dark			blo_6_a_crown

and the stent is rotated to the next feature.

Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 1	479	220	
Point 2	479	329	
Point 3	370	329	
Point 4	370	220	
int_6_t_light			blo_6_a_crown
Point 0	156	245	
Point 1	166	245	
Point 2	166	255	
Point 3	156	255	
Point 4	156	245	

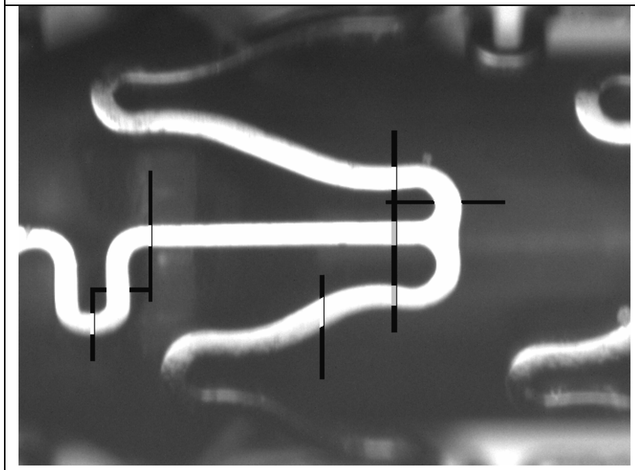


Sensor Shape	X	Y	Position Reference
blo_6_c_ts	205	141	blo_6_a_crown
	323	252	
blo_6_c_bs	205	257	blo_6_a_crown
	321	380	
blo_6_c_crown	125	185	blo_6_a_crown
	215	315	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

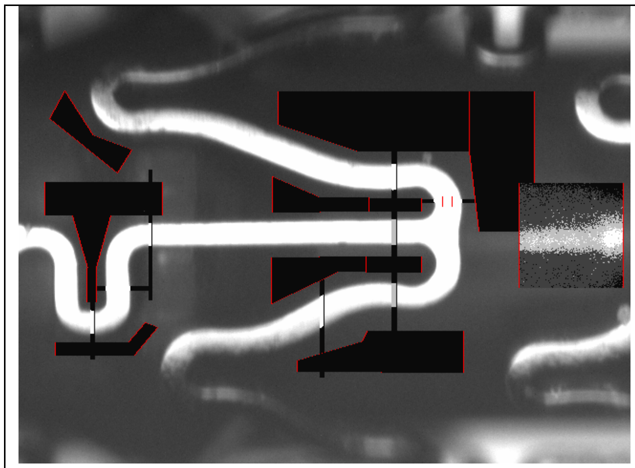
Feature 7

Feature 7 occurs three times within each middle ring. It uses the same reference and inspection sensors as Feature 9.



Sensor Shape	X	Y	Position Reference
blo_7_a_keyhole	75	294	blo_7_a_midstr
	139	298	
blo_7_a_botstr	315	281	None
	319	389	
blo_7_a_midstr	136	172	blo_7_a_m
	139	308	
blo_7_a_m2	390	130	blo_7_a_key2
	395	340	
blo_7_a_key2	75	294	blo_7_a_keyhole
	79	370	
blo_7_a_m	384	203	blo_7_a_botstr
	508	206	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

Sensor Type: Blob Soft Sensors, Blob Selectors
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.

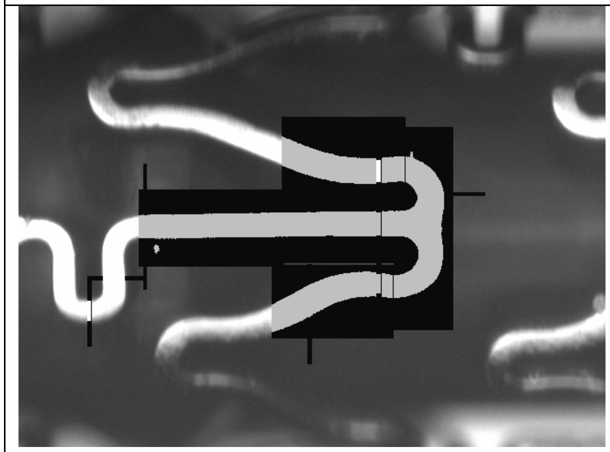


Sensor Shape	X	Y	Position Reference
blo_7_w_1			blo_7_a_m2
Point 0	363	265	
Point 1	413	265	
Point 2	413	277	
Point 3	363	277	
Point 4	363	265	
blo_7_w_2			blo_7_a_m2
Point 0	366	204	
Point 1	413	204	
Point 2	413	213	
Point 3	366	213	
Point 4	366	204	
blo_7_w_3			blo_7_a_m2
Point 0	366	215	
Point 1	265	215	
Point 2	265	185	
Point 3	315	201	
Point 4	366	201	
Point 5	366	215	
blo_7_w_4			blo_7_a_m2
Point 0	363	263	
Point 1	264	268	
Point 2	264	302	
Point 3	335	278	
Point 4	363	278	
Point 5	363	263	
blo_7_w_5			blo_7_a_m2
Point 0	471	90	
Point 1	529	90	
Point 2	529	236	
Point 3	481	236	
Point 4	471	152	
Point 5	471	90	
blo_7_w_6			blo_7_a_m2
Point 0	271	90	
Point 1	271	124	
Point 2	354	152	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 3	471	152			
Point 4	471	90			
Point 5	271	90			
blo_7_w_7			blo_7_a_key2		
Point 0	48	85			
Point 1	32	118			
Point 2	103	175			
Point 3	118	175			
Point 4	78	136			
Point 5	48	85			
blo_7_w_8			blo_7_a_keyhole		
Point 0	27	185			
Point 1	150	185			
Point 2	150	219			
Point 3	27	219			
Point 4	27	185			
blo_7_w_9			blo_7_a_keyhole		
Point 0	60	220			
Point 1	72	275			
Point 2	81	274			
Point 3	92	220			
Point 4	60	220			
blo_7_w_10			blo_7_a_keyhole		
Point 0	72	275			
Point 1	80	275			
Point 2	80	305			
Point 3	72	305			
Point 4	72	275			
blo_7_w_11			blo_7_a_key2		
Point 0	132	332			
Point 1	145	337			
Point 2	121	366			
Point 3	38	366			
Point 4	38	353			
Point 5	114	353			
Point 6	132	332			
blo_7_w_12			blo_7_a_m2		
Point 0	291	383			
Point 1	291	372			
Point 2	359	352			
Point 3	364	341			
Point 4	465	341			
Point 5	465	384			
Point 6	291	383			
int_7_t_dark			blo_7_a_m2		
Point 0	523	186			
Point 1	620	186			
Point 2	620	295			
Point 3	523	295			
Point 4	523	186			
int_7_t_light			blo_7_a_m		
Point 0	443	200			
Point 1	453	200			

Point 2	453	210	
Point 3	443	210	
Point 4	443	200	

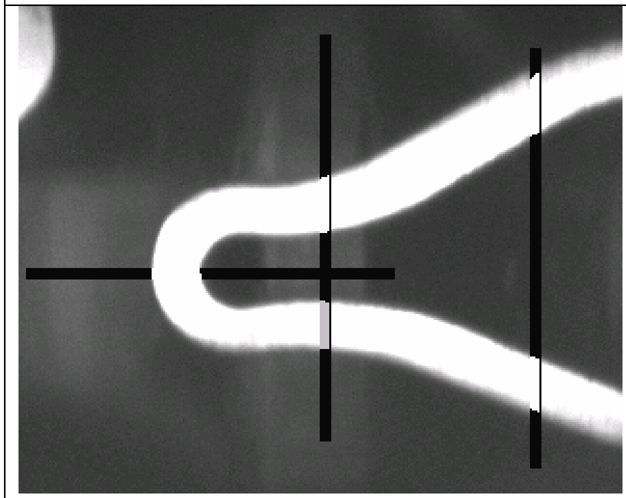


Sensor Shape	X	Y	Position Reference
blo_7_c_ts	287	121	blo_7_a_keyhole
	421	207	
blo_7_c_bs	276	282	blo_7_a_keyhole
	408	361	
blo_7_c_ms	283	208	blo_7_a_keyhole
	414	275	
blo_7_c_ms2	131	200	blo_7_a_keyhole
	289	283	
blo_7_c_crown	409	132	blo_7_a_keyhole
	473	352	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

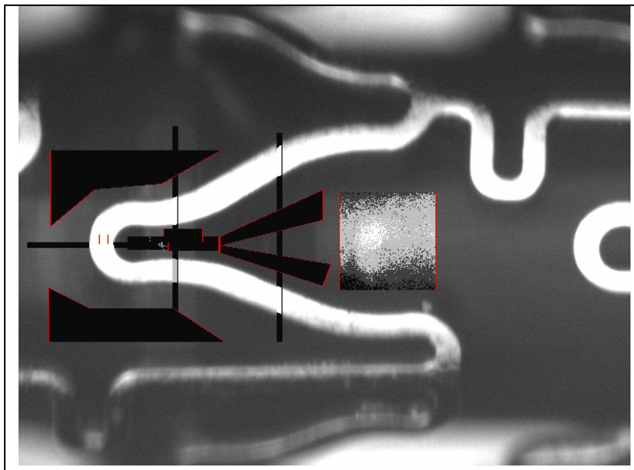
Feature 8

Feature 8 occurs three times within each middle ring. It uses the same reference and inspection sensors as Features 2, 4, and 10.



Sensor Shape	**X**	**Y**	**Position Reference**
blo_2_a_topstr	270	100	None
	275	240	
blo_2_a_botstr	270	245	blo_2_a_topstr
	275	374	
blo_2_a_neck	162	136	blo_2_a_botstr
	167	346	
blo_2_a_crown	10	239	blo_2_a_neck
	200	244	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

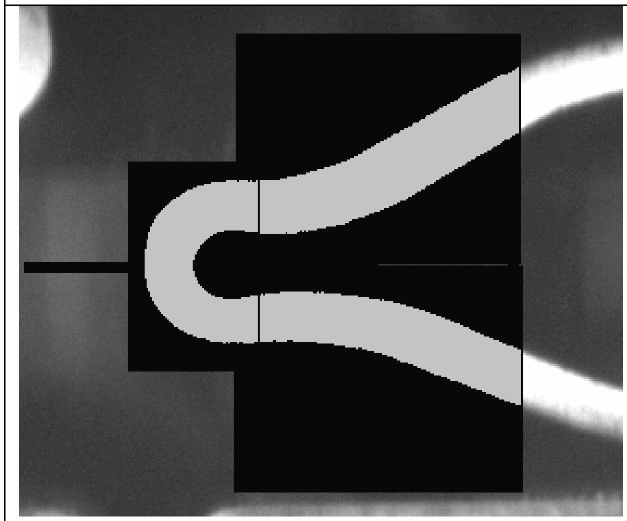
Sensor Type: Blob Soft Sensors, Blob Selectors
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Shape	X	Y	Position Reference
blo_2_w_1			blo_2_a_crown
Point 0	156	233	
Point 1	192	233	
Point 2	192	246	
Point 3	156	246	
Point 4	156	233	
blo_2_w_2			blo_2_a_crown
Point 0	193	233	
Point 1	246	233	
Point 2	246	247	
Point 3	193	247	
Point 4	193	233	
blo_2_w_3			blo_2_a_crown
Point 0	247	249	
Point 1	355	284	
Point 2	363	262	
Point 3	247	242	
Point 4	247	249	
blo_2_w_4			blo_2_a_crown
Point 0	247	233	
Point 1	355	184	
Point 2	355	215	
Point 3	247	242	
Point 4	247	233	
blo_2_w_5			blo_2_a_crown
Point 0	90	160	
Point 1	70	222	
Point 2	117	183	
Point 3	187	177	
Point 4	247	143	
Point 5	90	160	
blo_2_w_6			blo_2_a_crown
Point 0	68	286	
Point 1	111	308	
Point 2	199	308	
Point 3	251	342	
Point 4	68	342	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 5	68	286	
int_2_t_dark			blo_2_a_crown
Point 0	372	187	
Point 1	473	187	
Point 2	473	288	
Point 3	372	288	
Point 4	372	187	
int_2_t_light			blo_2_a_crown
Point 0	121	231	
Point 1	130	231	
Point 2	130	240	
Point 3	121	240	
Point 4	121	231	



Sensor Shape	X	Y	Position Reference
blo_2_c_ts	160	116	blo_2_a_crown
	313	239	
blo_2_c_bs	159	251	blo_2_a_crown
	314	357	
blo_2_c_crown	102	185	blo_2_a_crown
	172	297	

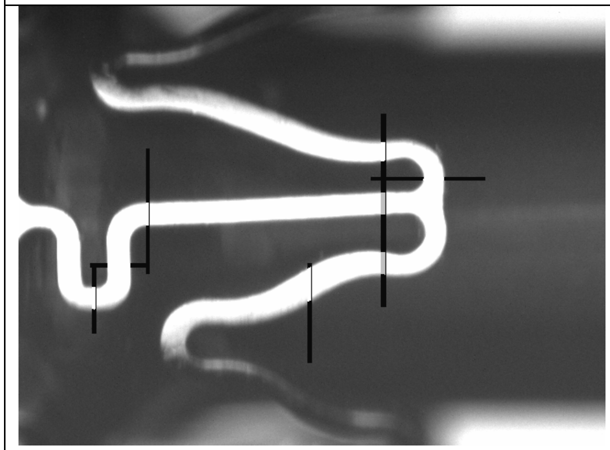
Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

11.4. *Feature Inspection: End Ring*

The inspection for the middle rings is similar to the beginning and middle rings. The differences being the types of features found in the rings and that after the ring has been inspected, the stent must be unloaded. The inspection starts by realigning the stent and then incrementing through the 12 features. The CNC fixture then returns to the "Home" position and open the tailstock.

Feature 9

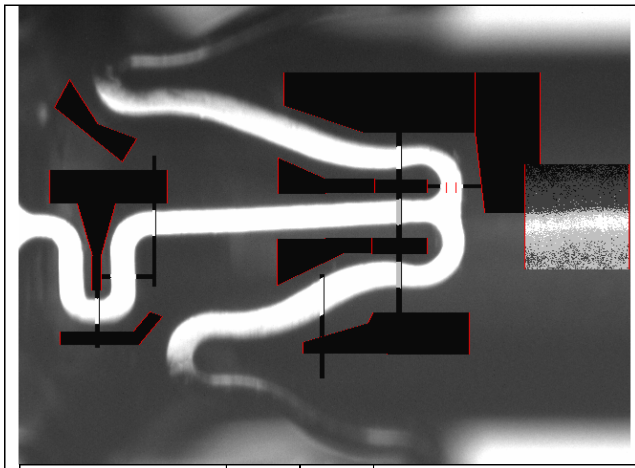
Feature 9 occurs three times within each middle ring. It uses the same reference and inspection sensors as Feature 7.



Sensor Shape	**X**	**Y**	**Position Reference**
blo_7_a_keyhole	75	294	blo_7_a_midstr
	139	298	
blo_7_a_botstr	315	281	None
	319	389	
blo_7_a_midstr	136	172	blo_7_a_m
	139	308	
blo_7_a_m2	390	130	blo_7_a_key2
	395	340	
blo_7_a_key2	75	294	blo_7_a_keyhole
	79	370	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

Sensor Type: Blob Soft Sensors, Blob Selectors
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.

Sensor Type: Blob Soft Sensors, Blob Selectors,

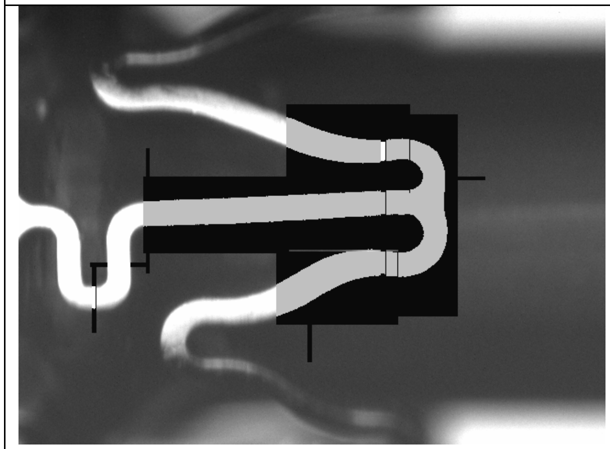


Sensor Shape	X	Y	Position Reference
blo_7_w_1			blo_7_a_m2
Point 0	363	265	
Point 1	413	265	
Point 2	413	277	
Point 3	363	277	
Point 4	363	265	
blo_7_w_2			blo_7_a_m2
Point 0	366	204	
Point 1	413	204	
Point 2	413	213	
Point 3	366	213	
Point 4	366	204	
blo_7_w_3			blo_7_a_m2
Point 0	366	215	
Point 1	265	215	
Point 2	265	185	
Point 3	315	201	
Point 4	366	201	
Point 5	366	215	
blo_7_w_4			blo_7_a_m2
Point 0	363	263	
Point 1	264	268	
Point 2	264	302	
Point 3	335	278	
Point 4	363	278	
Point 5	363	263	
blo_7_w_5			blo_7_a_m2
Point 0	471	90	
Point 1	529	90	
Point 2	529	236	
Point 3	481	236	
Point 4	471	152	
Point 5	471	90	
blo_7_w_6			blo_7_a_m2
Point 0	271	90	
Point 1	271	124	
Point 2	354	152	
Point 3	471	152	

Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 4	471	90			
Point 5	271	90			
blo_7_w_7			blo_7_a_key2		
Point 0	48	85			
Point 1	32	118			
Point 2	103	175			
Point 3	118	175			
Point 4	78	136			
Point 5	48	85			
blo_7_w_8			blo_7_a_keyhole		
Point 0	27	185			
Point 1	150	185			
Point 2	150	219			
Point 3	27	219			
Point 4	27	185			
blo_7_w_9			blo_7_a_keyhole		
Point 0	60	220			
Point 1	72	275			
Point 2	81	274			
Point 3	92	220			
Point 4	60	220			
blo_7_w_10			blo_7_a_keyhole		
Point 0	72	275			
Point 1	80	275			
Point 2	80	305			
Point 3	72	305			
Point 4	72	275			
blo_7_w_11			blo_7_a_key2		
Point 0	132	332			
Point 1	145	337			
Point 2	121	366			
Point 3	38	366			
Point 4	38	353			
Point 5	114	353			
Point 6	132	332			
blo_7_w_12			blo_7_a_m2		
Point 0	291	383			
Point 1	291	372			
Point 2	359	352			
Point 3	364	341			
Point 4	465	341			
Point 5	465	384			
Point 6	291	383			
int_7_t_dark			blo_7_a_m2		
Point 0	523	186			
Point 1	620	186			
Point 2	620	295			
Point 3	523	295			
Point 4	523	186			
int_7_t_light			blo_7_a_m		
Point 0	443	200			
Point 1	453	200			
Point 2	453	210			

Point 3	443	210	
Point 4	443	200	

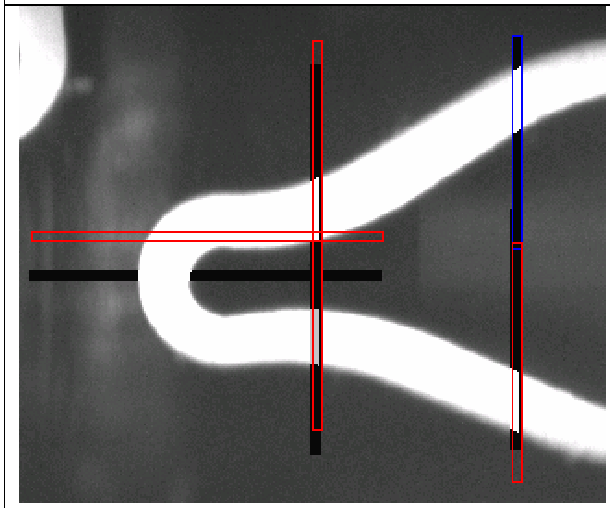


Sensor Shape	X	Y	Position Reference
blo_7_c_ts	287	121	blo_7_a_keyhole
	421	207	
blo_7_c_bs	276	282	blo_7_a_keyhole
	408	361	
blo_7_c_ms	283	208	blo_7_a_keyhole
	414	275	
blo_7_c_ms2	131	200	blo_7_a_keyhole
	289	283	
blo_7_c_crown	409	132	blo_7_a_keyhole
	473	352	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

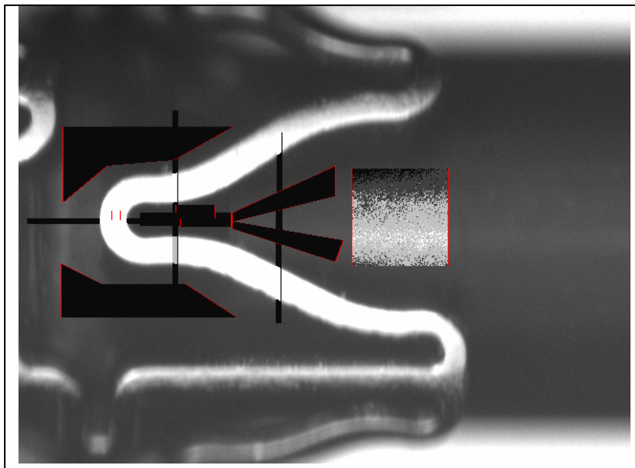
Feature 10

Feature 10 occurs three times within the end ring. It uses the same reference and inspection sensors as Features 2, 4, and 8.



Sensor Shape	X	Y	Position Reference
blo_2_a_topstr	270	100	None
	275	240	
blo_2_a_botstr	270	245	blo_2_a_topstr
	275	374	
blo_2_a_neck	162	136	blo_2_a_botstr
	167	346	
blo_2_a_crown	10	239	blo_2_a_neck
	200	244	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

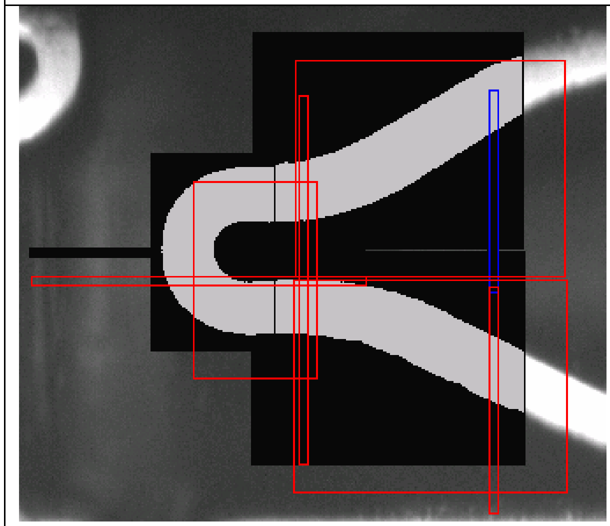
Sensor Type: Blob Soft Sensors, Blob Selectors
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Shape	X	Y	Position Reference
blo_2_w_1			blo_2_a_crown
Point 0	156	233	
Point 1	192	233	
Point 2	192	246	
Point 3	156	246	
Point 4	156	233	
blo_2_w_2			blo_2_a_crown
Point 0	193	233	
Point 1	246	233	
Point 2	246	247	
Point 3	193	247	
Point 4	193	233	
blo_2_w_3			blo_2_a_crown
Point 0	247	249	
Point 1	355	284	
Point 2	363	262	
Point 3	247	242	
Point 4	247	249	
blo_2_w_4			blo_2_a_crown
Point 0	247	233	
Point 1	355	184	
Point 2	355	215	
Point 3	247	242	
Point 4	247	233	
blo_2_w_5			blo_2_a_crown
Point 0	90	160	
Point 1	70	222	
Point 2	117	183	
Point 3	187	177	
Point 4	247	143	
Point 5	90	160	
blo_2_w_6			blo_2_a_crown
Point 0	68	286	
Point 1	111	308	
Point 2	199	308	
Point 3	251	342	
Point 4	68	342	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 5	68	286	
int_2_t_dark			blo_2_a_crown
Point 0	372	187	
Point 1	473	187	
Point 2	473	288	
Point 3	372	288	
Point 4	372	187	
int_2_t_light			blo_2_a_crown
Point 0	121	231	
Point 1	130	231	
Point 2	130	240	
Point 3	121	240	
Point 4	121	231	

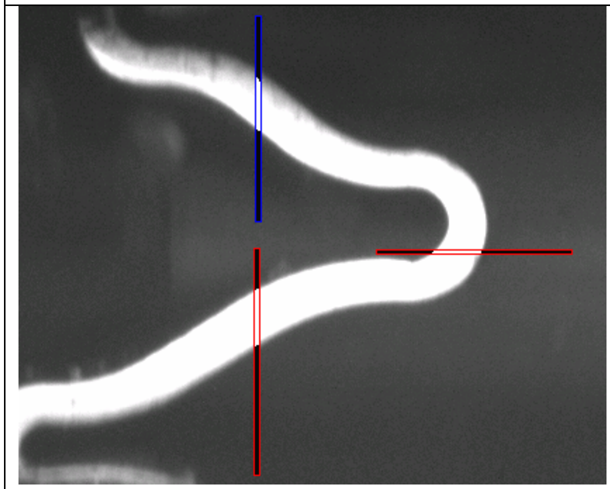


Sensor Shape	X	Y	Position Reference
Top Strut	160	116	blo_2_a_crown
	313	239	
Bottom Strut	159	251	blo_2_a_crown
	314	357	
Crown	102	185	blo_2_a_crown
	172	297	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

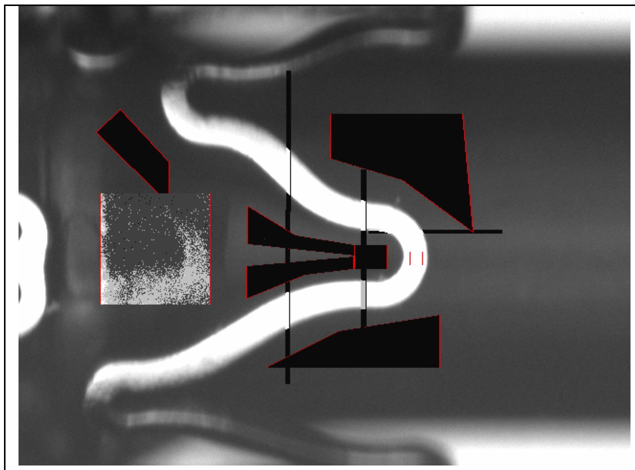
Feature 11

Feature 11 occurs three times within the end ring. It uses the same reference sensors as Feature 3.



Sensor Shape	X	Y	Position Reference
blo_11_a_topstr	280	68	None
	284	214	
blo_11_a_botstr	279	233	blo_11_a_botstr
	283	394	
blo_11_c_crown	366	234	blo_11_a_botstr
	505	237	
blo_11_c_neck	358	171	blo_11_a_botstr
	363	350	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

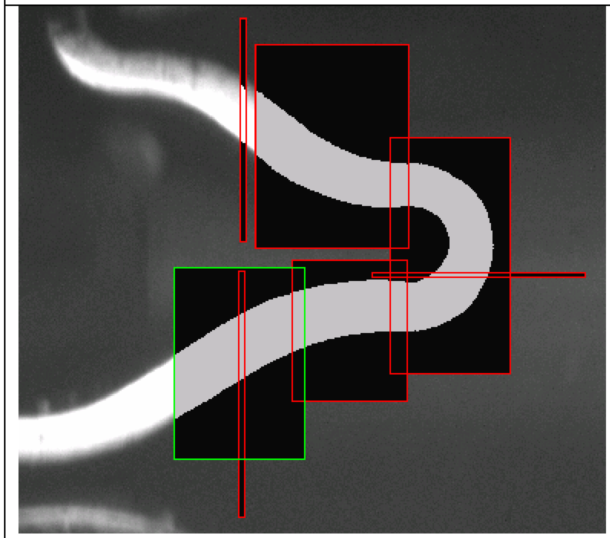
Sensor Type: Blob Soft Sensors, Blob Selectors
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.



Sensor Shape	X	Y	Position Reference
blo_11_w_1			blo_11_a_neck
Point 0	351	250	
Point 1	375	250	
Point 2	375	274	
Point 3	351	274	
Point 4	351	250	
blo_11_w_2			blo_11_a_neck
Point 0	350	262	
Point 1	350	274	
Point 2	300	280	
Point 3	238	305	
Point 4	238	272	
Point 5	350	262	
blo_11_w_3			blo_11_a_neck
Point 0	351	260	
Point 1	239	248	
Point 2	239	209	
Point 3	290	240	
Point 4	352	250	
Point 5	351	260	
blo_11_w_4			blo_11_a_neck
Point 0	326	159	
Point 1	326	113	
Point 2	464	113	
Point 3	475	236	
Point 4	400	181	
Point 5	326	159	
blo_11_w_5			blo_11_a_neck
Point 0	157	207	
Point 1	157	163	
Point 2	106	108	
Point 3	81	132	
Point 4	157	207	
blo_11_w_6			blo_11_a_neck
Point 0	440	323	
Point 1	440	377	
Point 2	260	377	

Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 3	333	340	
Point 4	440	323	
int_11_t_dark			blo_11_a_crown
Point 0	85	196	
Point 1	200	196	
Point 2	200	311	
Point 3	85	311	
Point 4	85	196	
int_11_t_light			blo_11_a_crown
Point 0	409	257	
Point 1	422	257	
Point 2	422	270	
Point 3	409	270	
Point 4	409	275	

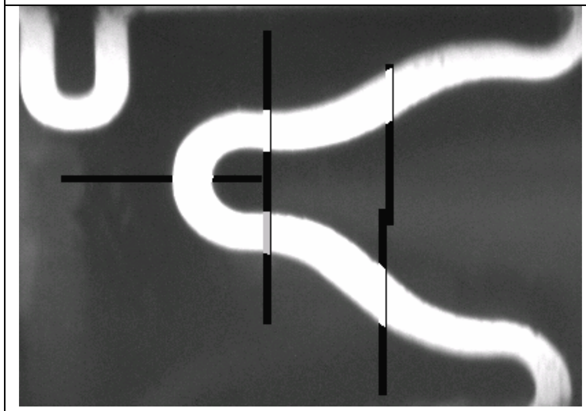


Sensor Shape	X	Y	Position Reference
blo_11_c_ts	290	85	blo_11_a_crown
	390	218	
blo_11_c_bs	314	226	blo_11_a_crown
	389	318	
blo_11_c_bs2	237	231	blo_11_a_crown
	322	356	
blo_11_c_crown	378	146	blo_11_a_crown
	456	300	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

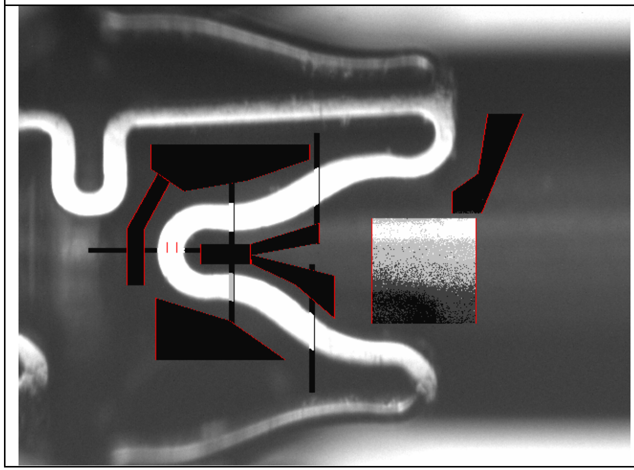
Feature 12

Feature 12 occurs three times within the end ring. It uses the same reference and inspection sensors as Feature 6.



Sensor Shape	X	Y	Position Reference
blo_6_a_topstr	309	133	None
	314	248	
blo_6_a_botstr	304	275	blo_6_a_topstr
	309	408	
blo_6_a_crown	73	251	blo_6_a_neck
	218	256	
blo_6_a_neck	220	151	blo_6_a_botstr
	225	361	
Sensor Thresholds	**Type**	**Value**	
Threshold Level	Fixed Value	65	
Preprocessing	Light Blob		
Parameters	Enable Boundary Blobs	enabled	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Alignment
Purpose: These sensors act as reference points for the inspection sensors. They return a PASS when a light blob is found within the search area. They return a FAIL if no light blobs are found.
Pass: A PASS signifies that one of the stent's struts has been found. The Blob Selector than calculates the center of mass of the blob. The center of mass for each strut is used as a reference for aligning the inspection sensors.
Fail: A FAIL signifies that there is a geometry error within the stent or an error with the sensor placement. This causes an invalid test and unpredictable inspection results.

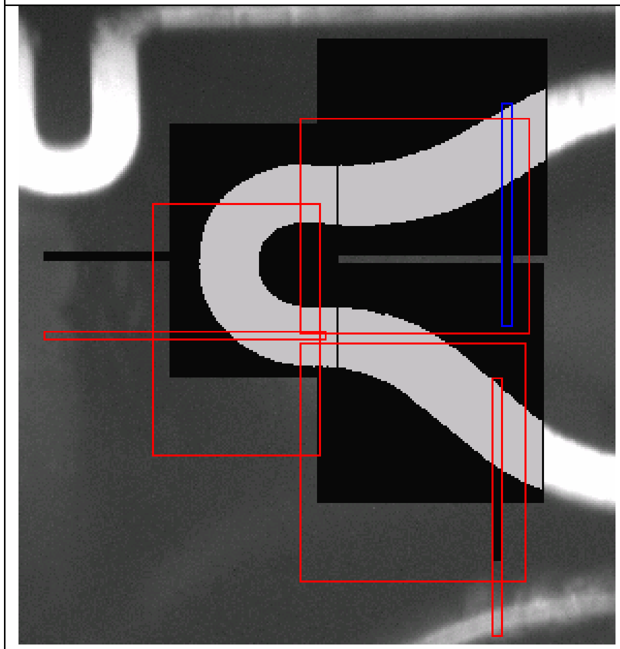


Sensor Type: Blob Soft Sensors, Blob Selectors, Intensity Sensors
Package Type: Black Space Inspection
Purpose: These sensors inspect the area between stent struts. They return a PASS when no light blobs are found or blobs under the threshold area are found. They return a FAIL if multiple blobs whose sum is above a known threshold.
Pass: A PASS signifies that the search area has been found free of defects. The inspection has passed and the stent is rotated to the next

Sensor Shape	X	Y	Position Reference
blo_6_w_1			blo_6_a_crown
Point 0	196	247	
Point 1	243	247	
Point 2	243	267	
Point 3	196	267	
Point 4	196	247	
blo_6_w_2			blo_6_a_crown
Point 0	243	258	
Point 1	244	267	
Point 2	289	288	
Point 3	331	324	
Point 4	331	280	
Point 5	243	258	
blo_6_w_3			blo_6_a_crown
Point 0	243	247	
Point 1	315	225	
Point 2	315	245	
Point 3	243	258	
Point 4	243	247	
blo_6_w_4			blo_6_a_crown
Point 0	491	111	
Point 1	528	111	
Point 2	485	214	
Point 3	454	214	
Point 4	454	192	
Point 5	481	173	
Point 6	491	111	
blo_6_w_5			blo_6_a_crown
Point 0	139	145	
Point 1	139	168	
Point 2	173	191	
Point 3	240	180	
Point 4	305	158	
Point 5	305	145	
Point 6	139	145	
blo_6_w_6			blo_6_a_crown
Point 0	144	303	
Point 1	144	367	
Point 2	279	367	
Point 3	222	327	
Point 4	144	303	
blo_6_w_7			blo_6_a_crown
Point 0	132	289	
Point 1	114	289	
Point 2	114	230	
Point 3	146	173	
Point 4	159	182	
Point 5	132	231	
Point 6	132	289	
int_6_t_dark			blo_6_a_crown

feature.
Fail: A FAIL signifies that there is a defect in the black space between stent struts. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

Point 1	479	220	
Point 2	479	329	
Point 3	370	329	



Sensor Shape	X	Y	Position Reference
blo_6_c_ts	205	141	blo_6_a_crown
	323	252	
blo_6_c_bs	205	257	blo_6_a_crown
	321	380	
blo_6_c_crown	125	185	blo_6_a_crown
	215	315	

Sensor Type: Blob Soft Sensors, Blob Selectors
Package Type: Strut Inspection
Purpose: These sensors inspect the stent struts and edges. They return a PASS when a single light blob whose area and perimeter are under the threshold values or multiple blobs whose sum of areas is under a threshold value are detected. They return a FAIL if multiple blobs whose sum is above a known threshold are detected.
Pass: A PASS signifies that the strut within the search area has been found and is free of defects. The Blob Selector has calculated the surface area of the blobs and the script has determined the sum of these areas. The sum is under the threshold value. The inspection has passed and the stent is rotated to the next feature.
Fail: A FAIL signifies that there is a defect on the surface of the stent. One of the Blob Sensors has detected multiple blobs above the threshold area or a blob with too large of an area or perimeter. The inspection has failed and the user is prompted.

12. Appendix F: Inspection Results

The detailed results of the test inspections are shown below. 15 stents were inspected and 2 were aborted mid-inspection due to geometry defects. The stents are classified as “D” for True Defect, “A” for Acceptable Defect, “N” for Mandrel Noise, or “M” for Missed Defect and are named using the following convention: Batch#_Stent#.

Stent ID	Inspection Result	Sensor Fail Reason	Count
B1_S1	A	>1 Blob - > Max Error - Blackspace	1
	D	> Max Error - Blackspace	4
	N	> Max Error - Blackspace	2
B1_S10	A	> Max Error - Blackspace	4
		1 Blob - Area Error	1
		1 Blob - Perimeter Error	1
		Alignment	4
	D	> Max Error - Blackspace	28
		>1 Blob - > Max Error - Blackspace	4
		>1 Blob - Area Error	1
		1 Blob - Area Error	1
		Alignment	1
B1_S13	A	> Max Error - Blackspace	4
		>1 Blob - Area Error	1
		1 Blob - Area Error	1
	D	> Max Error - Blackspace	67
		>1 Blob - > Max Error - Blackspace	1
		>1 Blob - Area Error	4
		1 Blob - Perimeter Error	1
		Alignment	3
	N	> Max Error - Blackspace	2
B1_S15	A	> Max Error - Blackspace	7
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Area Error	2
		Alignment	1
	D	> Max Error - Blackspace	38
		>1 Blob - > Max Error - Blackspace	2
		>1 Blob - Area Error	3
		Alignment	8
B1_S17	A	> Max Error - Blackspace	4
		1 Blob - Area Error	2
		Alignment	1
	D	> Max Error - Blackspace	25
		Alignment	1

B1_S19	A	> Max Error - Blackspace	6
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Perimeter Error	1
		Alignment	1
	D	> Max Error - Blackspace	27
		>1 Blob - > Max Error - Blackspace	1
B1_S19	D	Alignment	1
B1_S2	A	> Max Error - Blackspace	15
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Area Error	6
		Alignment	17
	D	> Max Error - Blackspace	7
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Area Error	1
		1 Blob - Perimeter Error	1
	M	Unknown	1
	N	> Max Error - Blackspace	2
B1_S20	A	> Max Error - Blackspace	6
		1 Blob - Area Error	1
		1 Blob - Perimeter Error	1
	D	> Max Error - Blackspace	15
		>1 Blob - Area Error	1
B1_S3	A	> Max Error - Blackspace	9
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Area Error	10
		Alignment	1
	D	> Max Error - Blackspace	5
		>1 Blob - > Max Error - Blackspace	1
	N	> Max Error - Blackspace	7
B1_S4	A	> Max Error - Blackspace	6
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Perimeter Error	5
		Alignment	1
	D	> Max Error - Blackspace	11
		>1 Blob - > Max Error - Blackspace	1
		1 Blob - Area Error	1
	N	> Max Error - Blackspace	5
B1_S5	A	> Max Error - Blackspace	9
		>1 Blob - > Max Error - Blackspace	3
		Alignment	1
	D	> Max Error - Blackspace	5
		>1 Blob - > Max Error - Blackspace	1
	N	> Max Error - Blackspace	6
B1_S6	A	1 Blob - Perimeter Error	1

	D	> Max Error - Blackspace	11
		1 Blob - Perimeter Error	1
	N	> Max Error - Blackspace	1
B1_S7	A	> Max Error - Blackspace	3
		1 Blob - Area Error	6
		1 Blob - Perimeter Error	2
B1_S7	A	Alignment	1
	D	> Max Error - Blackspace	20
		>1 Blob - Area Error	1
	M	> Max Error - Blackspace	1
	N	> Max Error - Blackspace	1
B1_S8	A	> Max Error - Blackspace	5
		Alignment	1
	D	> Max Error - Blackspace	15
B1_S9	A	> Max Error - Blackspace	1
		>1 Blob - > Max Error - Blackspace	1
		Alignment	3
	D	> Max Error - Blackspace	12
		>1 Blob - > Max Error - Blackspace	1
		Alignment	1
	N	> Max Error - Blackspace	2

13. Appendix G: Hardware Description

For the console to adapt to external triggering, there were a few hardware changes that need to be made to it. The first step was to identify the binary inputs and outputs that were available on the CNC controller breakout board. The main available output was Binary Output 14. Binary Output 14 was used to trigger the DVT system to perform an inspection. Using isolated input and output modules, Binary Output 14 was connected from the CNC controller output to a DVT hardware input on the inspection camera break out box. This connection is shown in Figure 40.

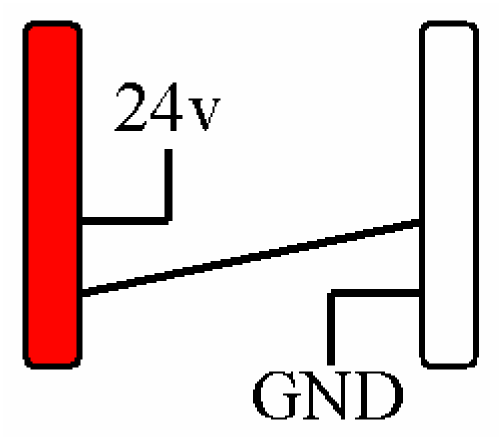


Figure 40: Hardware IO Description

The Red block is the output module. Its primary function is to connect the 24 volts to the input module thereby, transmitting a pulse. This pulse is the signal to run an inspection. The trigger waveform must have an on time of 5 ms and the inspection triggers on the rising edge of the pulse. Setting this output from 0 to 1 can also trigger a DVT inspection.

Wiring another pair of input/output modules as shown above allowed DVT to signal back to the automation controller. One of the positions on the inspection camera break out box was configured to the IO of Inspection Toggle. This IO changes state each time an inspection is finished and was used to signal back to the CNC controller that an inspection had been completed. This way the inspection could take a variable length of time without being cut short by the incremented rotation of the CNC fixture.

All necessary hardware connections are detailed below with their respective uses noted. There are a total of 4 important connections to allow efficient hardware communication between the vision system and motion control application.

DVT Position	Type	DVT Function	Motion Position	Type	Comment

1	In	Product Select	31	Out	Starts a product swap
2	In	Trigger	30	Out	Triggers inspection
3	In	Product bit0	23	Out	Controls which product is loaded
4	Out	Inspection Toggle	3	In	Changes state after inspection completed
5	-	None	-	-	
6	-	None	-	-	
7	Out	Strobe2	N/A	N/A	Connects to lighting system
8	Out	Strobe	N/A	N/A	Connects to lighting system

With the hardware connected the DVT software, FrameWork, required configuration to make use of this setup. The I/O Configure tab located within the I/O Parameters window can be found under the I/O menu in FrameWork and should be configured so that the functions outlined above are met.

Another change to the defaults is to set the debounce times to 2ms. This will allow for increased frequency between toggles, since the system is opto-electrical and there is no mechanical bouncing to regulate.

The last change necessary in Framework is to set digital id numbers for the two different products. This is done in the Product Management window located under the Product menu. By checking the “Enable Product for Digital Selection” and setting the ID to 0 for the edge find product and 1 for the stent scan product it is possible for an outside source to change the product on demand, without the need for background commands or VB interfaces.

Using this hardware setup there were zero wasted inspections or incomplete inspections. Using the previous software handshaking method required at least 2 inspection cycles per position. One cycle was used to perform the actual inspection and the other one to clear the variables. If the timing didn’t line up perfectly, at least one additional inspection cycle would be wasted. This hardware setup avoided this wasted cycle and resulted in a more efficient inspection process, and enabled additional automated usage.

14. Appendix H: Inspection G-Code Program

14.1. Vision_Inspection_System.PGM

```
' Vision_Inspection_System.PGM

' This program will control the motion control for the inspection of
' the VISION stents.


' -------------------------------------------------------------------
' Variable Declarations
' -------------------------------------------------------------------
DVAR $TimeOut                                 ' Amount to wait in milliseconds
                                              '  before a timeout error occurs
DVAR $Velocity                        ' Speed at which to move in inches/minute
DVAR $ScaleFactorX, $ScaleFactorY     ' Scale factors - this is the distance in
                                              '  inches that is the equivilent to 1
                                              '  pixel on screen
DVAR $YIncrement, $XIncrement                 ' The amount to increment in the Y/X to
                                              '  get to the next feature or ring.
DVAR $NPass, $NFail                   ' Counts of number fails and passes
DVAR $FeaturePosition, $Ring, $RingCount      ' These track the current position
                                                      '  and how many rings to go
                                                      '  used to image the whole stent
DVAR $XCenter, $YCenter                       ' Center positions that are used
                                              '  to perform updates
DVAR $LastInsp                                ' Holds the last inspection state
DVAR $resp                                    ' Response from UI Button
DVAR $NBridge, $NCobweb, $NPoolweb            ' Counting vars for the type of
DVAR $NClumping, $NGeometry                   ' failures recorded.
DVAR $NAccept, $NNondefect, $NContam
' -------------------------------------------------------------------


:Startup
' -------------------------------------------------------------------
' Motion Parameters
' -------------------------------------------------------------------
G68             ' Set rate-based accel/decel
G64             ' Set linear accel/decel
G109            ' Deceleration between moves
G82 X Y Z       ' Reset the position reference to the hardware home

ENABLE X        ' Enable X Axis
ENABLE Y        ' Enable Y Axis
ENABLE Z        ' Enable Z Axis
HOME Z          ' Home Z Axis
HOME X          ' Home X Axis
HOME Y          ' Home Y Axis
' -------------------------------------------------------------------
```

```
:StartInspection
' -------------------------------------------------------------------
' Variable Initialization
' -------------------------------------------------------------------
ExecuteNumLines = 10            ' Reserve CPU time for loops to execute
$TimeOut = 1000                         ' Wait 1 second
$XIncrement = 0.05286           ' Set the increment values for X
$YIncrement = 0.01963           '  and Y to move to the next Feature/ring
$Velocity = 20                  ' 20 Inches/minute velocity
$ScaleFactorX = .000121                 ' Engineering equivlent of 1 pixel
$ScaleFactorY = -.000124        '  in both the X and Y directions
$RingCount = 14                         ' Number of rings to scan total
$RO5 = $RingCount                       ' Provide this for DVT
$FeaturePosition = 0            ' Starting position
$Ring = 1                               ' Starting ring
$RO4 = 11                               ' Set TaskAck flag (will clear task complete)
$BO14 = 0                       ' Set the Vision Trigger to 0
$LastInsp = $BI3                        ' Set the last state to the current camera setting
$NPass = 0                              ' Number of passes
$NFail = 0                              ' Number of fails
$NBridge = 0                    ' Clear all counts
$NCobweb = 0
$NPoolweb = 0
$NClumping = 0
$NGeometry = 0
$NNondefect = 0
$NAccept = 0
$NContam = 0
' -------------------------------------------------------------------


' -------------------------------------------------------------------
' Message Window Setup & Display
' -------------------------------------------------------------------
MSGCLEAR -1  ' Clear all messages
MSGSHOW      ' Toggle to the message window (not the code view)
MSGDISPLAY 99, "Vision Stent Inspection System"
' -------------------------------------------------------------------


' -------------------------------------------------------------------
' Stent Load Operation
' -------------------------------------------------------------------
G90          ' Set to absolute coordinates
G0 Z0 X0 Y0  ' Move to the home position 0,0,0
$BO6 = 1     ' Open the tailstock
$RO6 = 002   ' Blink the "In Progress" light



             ' This section will ask if a stent is loaded and will
             ' react accordingly with the Yes, No, Cancel buttons.
             ' Yes - Inspection continues
             ' No - A prompt will tell the user to put a stent in place
             ' Cancel - Disable all drives, and provide a way to loop to
             '                  the top (startup) initialization (M0 instead of M2)
```

```
:WaitForLoad
$resp = MSGBOX DF_MSGBOX_YESNOCANCEL + DF_ICON_QUESTION, "Stent Loaded?"
IF($resp == YES_BUTTON) THEN
        GOTO DoneLoad
ELSE IF($resp == NO_BUTTON) THEN
        MSGDISPLAY 1, "Please Load a Stent"
        GOTO WaitForLoad
ELSE IF($resp == CANCEL_BUTTON) THEN
        MSGDISPLAY 1, "Operation Canceled"
        Call EndMotion
        M0
        GOTO Startup
END IF

:DoneLoad
MSGCLEAR 1   ' Clear the stent loading messages only
$RO6 = 001   ' Disable blinking - steady on
$BO6 = 0     ' Close the tailstock

' Move to start location for scan
G0 F50 Z5.900 X0 Y0   ' Rapid Traversal to Z=5.9
G1      F20 Z5.94            ' Slowly slide under the camera
        F10 Z5.946 X-.16     '  and back to just before the hard limit
                             '  to give the greatest possibility of
                             '  finding the stent edge, without an error
' -------------------------------------------------------------------


' -------------------------------------------------------------------
' Setup for Stent Edge and Fiducial Location Operation
' -------------------------------------------------------------------
MSGDISPLAY 2, "Stent Edge Product Change Initiated"
$BO7 = 0     ' Set bit 0 to Edge align
$BO15 = 1    ' Trigger Product Selection
G4 F1        ' Wait 1 second
$BO15 = 0    ' clear Product Trigger
G4 F3        ' Wait 5 seconds to guarentee a successful change
MSGDISPLAY 2, "Stent Edge Product Change Complete"


MSGDISPLAY 2, "Stent Edge Location In Progress"

G91          ' Incremental Positioning
G1 F$Velocity ' Linear traversal at $velocity
$RO4 = 10    ' Clear TaskAck flag

$LastInsp = $BI3        ' Set last trigger state
call TaskAck' Clear the flags
G4 F0.100

' -------------------------------------------------------------------


' -------------------------------------------------------------------
'  Find the Edge and Locate Fiducial Point
' -------------------------------------------------------------------
```

```
'M0
while $RI0 LT 11 do                                ' While the task is not complete

        call Inspect
        $XCenter = ($RI52 - $RI1) * $ScaleFactorX       ' Calculate new center position
        $YCenter = ($RI53 - $RI2) * $ScaleFactorY       '  by using the center-offset
        M1                                              ' Optional Stop for Debugging
        X$XCenter                                       ' Move to the designated Center
        Y$YCenter

endwhile                                                ' Loop

call TaskAck                    ' TaskComplete so it is neccessary to ACK it
MSGDISPLAY 2, "Fiducial Found, Starting Inspection"
' -------------------------------------------------------------------


' -------------------------------------------------------------------
' Swap to Stent Scan product
' -------------------------------------------------------------------
MSGDISPLAY 2, "Starting Product Change to Stent Scan"
$BO7 = 1        ' Set bit 0 to Edge align
$BO15 = 1       ' Trigger Product Selection
G4 F1
$BO15 = 0       ' clear Product Trigger
G4 F3
MSGDISPLAY 2, "Finished Product Change"
MSGCLEAR 2
MSGDISPLAY 2, "Inspection Scan Running"


' -------------------------------------------------------------------


' -------------------------------------------------------------------
'  Scan the stent and inspect
' -------------------------------------------------------------------
$RO2 = $Ring                ' Provide DVT with the current ring
$RO3 = $FeaturePosition             '  and FeaturePosition
                                    '  (ensuring good data is in the register)

while $Ring LT $RingCount + 1                      ' While rings are left to scan
MSGDISPLAY 3, "Inspecting Ring " $Ring
        while $FeaturePosition LT 12               ' While there are features to scan

                ' This is where the DVT will do the inspection
                call Inspect

                ' The result of the inspection will be stored in $RI5
                ' and the error code reported by the sensor will be stored
                ' in $RI6.

                if($RI5 == 10)                     ' Failure detected
                        $RO6 = 011                 ' Activate red fail light
                        $NFail = $NFail + 1        ' Increment failure count
                        CALL Classify              ' classify the defect
                else if ($RI5 == 11)                ' Pass detected
                        $RO6 = 101                  ' Activate green pass light
                        $NPass = $NPass + 1         ' Increment pass count
                endif

                M1                                          ' optional pause for
                                                            ' debugging

                $FeaturePosition = $FeaturePosition + 1' Increment feature
                $RO3 = $FeaturePosition                     ' Update DVT position register
                Y$YIncrement                        ' Increment the Y Position
        endwhile                                            ' Loop until out of feature positions

        $Ring = $Ring+1                                     ' Increment Ring
        $RO2 = $Ring                                ' Update DVT Ring register
        $FeaturePosition = 0                        ' Reset FeaturePosition
        $RO3 = $FeaturePosition                             ' Update DVT FeaturePosition Register
        X$XIncrement                                ' Increment the X Position
        if ($Ring < $RingCount)                     ' If not the End Ring
                Y-(2*$YIncrement)                           ' Increment the Y Position
                call AlignRing                      ' Call the ring alignment function
        end if                                      ' If it's past the end (done)
                                                            '  then do not call Align
        MSGCLEAR 3
endwhile                                                    ' Loop until all rings are scanned

' ---------------------------------------------------------------------


' ---------------------------------------------------------------------
' Inspection Complete or Aborted
' ---------------------------------------------------------------------
Call UnloadStent
Call DisplayStats

$resp = MSGBOX DF_MSGBOX_YESNO + DF_ICON_QUESTION, "Inspect Another?"
IF($resp == YES_BUTTON) THEN
        GOTO StartInspection
ELSE IF($resp == NO_BUTTON) THEN
        Call EndMotion
END IF

M2                          ' End the program
GOTO Startup
' ---------------------------------------------------------------------


' ---------------------------------------------------------------------
' Align Ring
' ---------------------------------------------------------------------
'  Function to align the stent to a known location to prevent
'    small errors from building over the length of a stent
DFS AlignRing                               ' Function AlignRing

M1                                          ' Optional stop for debugging
```

```
call TaskAck                        ' Trigger an ACK to clear step
G4 P0.07                                    ' Wait 70ms before starting align

while $RI0 LT 11 do                 ' While TaskComplete is not asserted
          $LastInsp = $BI3                  ' Record current inspection toggle
          call Inspect              ' Trigger an inspection
          G4 P0.150                         ' Wait 150ms for processing
          $XCenter = ($RI52-$RI1) * $ScaleFactorX         ' Calculate new Incremental
          $YCenter = ($RI53-$RI2) * $ScaleFactorY         '  position based on offsets
                                                                    '  and center location then
                                                                    '  convert to engineering
                                                                    '  units with the scale
                                                                    '  factors
          M1                                ' Optional stop for debugging
          X$XCenter                         ' Move based on the incremental position
          Y$YCenter                         '  calculated previously
endwhile                                    ' Loop until TaskComplete

ENDDFS                                      ' End of Function
' -----------------------------------------------------------------


' -----------------------------------------------------------------
' Inspect
' -----------------------------------------------------------------
'  This fuction will trigger one inspection, and wait for the updated
'    vision flag
DFS Inspect

$BO14 = 1                                   ' Trigger Inspection

wait($BI3 != $LastInsp) $TimeOut    ' Wait for the state change that
                                            ' indicates that the inspection
                                            ' has been completed

$LastInsp = $BI3                            ' Store the new state
$BO14 = 0                                   ' Clear the inspection trigger

ENDDFS
' -----------------------------------------------------------------


' -----------------------------------------------------------------
' TaskAck
' -----------------------------------------------------------------
DFS TaskAck
$RO4 = 11                ' Set the TaskAck flag
call Inspect     ' Trigger an inspection so DVT can set/clear
                         ' based on it's taskack instructions
$RO4 = 10                ' Clear the taskack flag
ENDDFS
' -----------------------------------------------------------------


' -----------------------------------------------------------------
' Classify
' ------------------------------------------------------------------
' This is used to classify the defects and record the statistics
DFS Classify
MSGMENU (DF_MENU_REMOVE), -1 ""
MSGMENU (DF_MENU_ADD), 1, "Bridge"
MSGMENU (DF_MENU_ADD), 2, "Cob web"
MSGMENU (DF_MENU_ADD), 3, "Pool Web"
MSGMENU (DF_MENU_ADD), 4, "Clumping"
MSGMENU (DF_MENU_ADD), 5, "Geometry"
MSGMENU (DF_MENU_ADD), 6, "Contaimination"
MSGMENU (DF_MENU_ADD), 7, "Acceptable Defect"
MSGMENU (DF_MENU_ADD), 8, "Not A Defect"


if ($RI6 > 2700) THEN
        $resp = 4
else
        $resp = 2
end if

$resp = MSGMENU (DF_MENU_SHOW),(DF_MSGBOX_OKONLY), "Select defect type; Select
Defect Type:;",$resp

if($resp == 1) then
        $NBridge = $NBridge+1
else if($resp == 2) then
        $NCobweb = $NCobweb+1
else if($resp == 3) then
        $NPoolweb = $NPoolweb+1
else if($resp == 4) then
        $NClumping = $NClumping+1
else if($resp == 5) then
        $NGeometry = $NGeometry+1
else if($resp == 6) then
        $NContam = $NContam+1
else if($resp == 7) then
        $NAccept = $NAccept+1
else if($resp == 8) then
        $NNondefect = $NNondefect+1
end if

ENDDFS
' ------------------------------------------------------------------


' ------------------------------------------------------------------
' DisplayStats
' ------------------------------------------------------------------
' This will display statistics about the last inspection and
' a recommended result.
DFS DisplayStats

MSGDISPLAY 5, "--------------------------------------"
MSGDISPLAY 5, "Total Inspections  : " ($NFail+$NPass)
MSGDISPLAY 5, "Passing Inspections: " $NPass
MSGDISPLAY 5, "Failing Inspections: " $NFail
MSGDISPLAY 5, "-----------------------------------"
MSGDISPLAY 5, "Bridge      : " $NBridge
MSGDISPLAY 5, "Cobweb      : " $NCobweb
MSGDISPLAY 5, "Pool Web    : " $NPoolweb
MSGDISPLAY 5, "Clumping    : " $NClumping
MSGDISPLAY 5, "Geometry    : " $NGeometry
MSGDISPLAY 5, "Contaminat  : " $NContam
MSGDISPLAY 5, "Acceptable  : " $NAccept
MSGDISPLAY 5, "Not A Defect: " $NNondefect
MSGDISPLAY 5, "-----------------------------------"

ENDDFS
' -----------------------------------------------------------------


' -----------------------------------------------------------------
' UnloadStent
' -----------------------------------------------------------------
' This will open the tailstock and move to home position.
DFS UnloadStent
G90                     ' Absolute coordinate mode
G0 Z0 X0 Y0             ' Move to 0,0,0
$BO6 = 1                ' Open the tailstock
$RO6 = 002              ' Flash Operator Light
ENDDFS
' -----------------------------------------------------------------


' -----------------------------------------------------------------
' EndMotion
' -----------------------------------------------------------------
'  This function will bring the fixture to the zero location
'    and then open the tailstock before stopping the program
DFS EndMotion ' Function EndMotion

$RO6 = 000
DISABLE X               ' Disable all Axis servos
DISABLE Y
DISABLE Z

ENDDFS                  ' End of Function
' -----------------------------------------------------------------
```

14.2. Light_Control.PGM

```
'Light_Control.PGM

' This program watches a register and will
' activate the lights on the console as appropriate

' The system used is by setting a 3 digit code, the order being
' RGY with a 0 meaning off, 1 meaning on, and 2 is flashing.

DVAR $Yellow, $Red, $Green              ' Variables to receive value
DVAR $YellowOut, $RedOut, $GreenOut             ' Variables to toggle light
DVAR $Counter                       ' Variable as a timer

while 1 do                                  ' Infinite looping

        $Red = INT ($RO6 / 100)                                 ' Extract the red value
        $Green = INT (($RO6 / 10) - $Red*10)            ' Extract the green
        $Yellow = INT ($RO6 - $Green * 10 - $Red * 100)         ' Extract the yellow

        if ($Yellow < 2)
                $YellowOut = $Yellow
        else if($Yellow == 2 )
                if ($Counter == 0)
                        $YellowOut = 1 - $YellowOut
                endif
        endif

        if ($Red < 2)
                $RedOut = $Red
        else if($Red == 2)
                if ($Counter == 0)
                        $RedOut = 1 - $RedOut
                endif
        endif

        if ($Green < 2)
                $GreenOut = $Green
        else if($Green == 2)
                if ($Counter == 0)
                        $GreenOut = 1 - $GreenOut
                endif
        endif

        $Counter = $Counter + 1
        if($Counter == 30)
                $Counter = 0
        endif

        $BO11 = $YellowOut
        $BO10 = $GreenOut
        $BO12 = $RedOut


endwhile
```

15. Appendix I: Inspection DVT Script

15.1. scr_Edge_Align.dvtscr

```
class scr_Edge_Align
{

//*******************************************************************
//*                 Standard Register Methods                  *
//*******************************************************************
public void setStatus(short stat)
{
        RegisterWriteShort(1032, stat);
}
public short getStatus()
{
        return RegisterReadShort(1032);
}
public short getInPositionFlag()
{
        return RegisterReadShort(14);
}
public short getVisionUpdatedFlag()
{
        return RegisterReadShort(1132);
}
public void setVisionUpdatedFlag()
{
        RegisterWriteShort(1132, 11);
}
public void clearVisionUpdatedFlag()
{
        RegisterWriteShort(1132, 10);
}
public short getTaskAckFlag()
{
        return RegisterReadShort(20);
}
public short getTaskCompetedFlag()
{
        return RegisterReadShort(1024);
}
public void setTaskCompletedFlag()
{
        RegisterWriteShort(1024,11);
}
```

```
public void clearTaskCompletedFlag()
{
	RegisterWriteShort(1024,10);
}
public short getTaskStep()
{
	return RegisterReadShort(1030);
}
public void setTaskStep(short step)
{
	RegisterWriteShort(1030,step);
}
public int[] getOffset()
{
	int[] off = new int[2];
	off[0] = RegisterReadShort(1026);
	off[1] = RegisterReadShort(1028);
	return off;
}
public void setOffset(short x, short y)
{
	RegisterWriteShort(1026,x);
	RegisterWriteShort(1028,y);
	getOffset();
}
public void setAbsolutePosition(short x, short y)
{
	RegisterWriteShort(1128, x);
	RegisterWriteShort(1130, y);
}
public void setAbsolutePosition(int[] pos)
{
	RegisterWriteShort(1128, pos[0]);
	RegisterWriteShort(1130, pos[1]);
}
public void setRelativePosition(short x, short y)
{
	int[] off = getOffset();
	RegisterWriteShort(1128, x+off[0]);
	RegisterWriteShort(1130, y+off[1]);
}
public void setInspectionPass()
{
	RegisterWriteShort(1034, 11);
}
public void setInspectionFail()
```

```
{
	RegisterWriteShort(1034, 10);
}
//*********************************************************************



//*********************************************************************
//	Alignment Functions
//*********************************************************************
// This function will check for the center point of the template seek
// so that the first move can get the blob sensors to the point that
// they are able to be used for better adjustment.

// Since the template will catch off TDC it is only semi-accurate, to about
// 6 pixels which is why the coarse and fine adjustment are still needed.
public int[] getTemplateAdjust()
{
	int[] pos = new int[2];
	//set up variables for the pos
	pos = getOffset();
	//get the current offset
	DebugPrint("Starting Template Adjust");
	if(tmp_1_a_fidseek.Result != 0)
	{
		setStatus(18);
		DebugPrint("Template Adjust Failure");
	}
	else
	{
		pos[0] = tmp_1_a_fidseek.Position.X - 90;
		pos[1] = tmp_1_a_fidseek.Position.Y;
	}

	return pos;
}

// The coarse adjustment is a rotational (Y) axis adjustment
// and is required to be performed before an X adjust on the
// keyhole can be reliably used.

// The averages of all the struts should balance to TDC in
// the Y axis so this is just a simple average calcuation
public int[] getCoarseAdjustment()
{
	int[] pos = new int[2];
```

```
        pos = getOffset();

        DebugPrint("Starting Coarse Adjust");
        if(bls_1_a_topstr.NumBlobs != 1)                    //verify that only 1
blob is
        {
        //found by the alignment blobs
                setStatus(11);
        //otherwise alignment will be
        }
        //wrong when it repositions
        else if(bls_1_a_botstr.NumBlobs != 1)
        {
                setStatus(12);
        }
        else if(bls_1_a_fid.NumBlobs != 1)
        {
                setStatus(13);
        }
        else
        {
                pos[1] = (((bls_1_a_topstr.BlobPosition.Y[0] +
bls_1_a_botstr.BlobPosition.Y[0])/2)
                                + bls_1_a_fid.BlobPosition.Y[0])/2;
                DebugPrint("Success");
        }

        return pos;
}

// Fine adjustment is used to calculate the total movement in both the X
// and Y direction. The Y movement is again done
// via the blobs set on the struts to perform rotational
// alignment.  The blobs that fall across the keyhole are
// used for the horizontal adjustment based on their
// average position and a given offset.
public int[] getFineAdjustment()
{
        int[] pos = new int[2];
        pos = getOffset();

        DebugPrint("Fine Adjust");
        // beginning ring
        if(bls_1_a_topstr.NumBlobs != 1)                    //verify that only 1
blob is
```

```
	{
	//found by the alignment blobs
		setStatus(14);
	//otherwise alignment will be
	}
	//wrong when it repositions
	else if(bls_1_a_botstr.NumBlobs != 1)
	{
		setStatus(15);
	}
	else if(bls_1_a_fid.NumBlobs != 1)
	{
		setStatus(16);
	}
	else if(bls_1_a_key.NumBlobs != 2)
	{
		setStatus(17);
	}
	else
	{
		DebugPrint("Success");
		pos[1] = (((bls_1_a_topstr.BlobPosition.Y[0] +
bls_1_a_botstr.BlobPosition.Y[0])/2)
					+ bls_1_a_fid.BlobPosition.Y[0])/2;
		pos[0] =
(bls_1_a_key.BlobPosition.X[0]+bls_1_a_key.BlobPosition.X[1])/2 - 190;
	}

	return pos;
}
//********************************************************************

//********************************************************************
//	Inspection sensor testing methods
//********************************************************************
public void inspect()
{
	clearVisionUpdatedFlag();

	short shtTaskStep = getTaskStep();	// Get the current task step

	setOffset(320, 240);	// Center of the view - offset that should be used
					// from the 0 position when calculating the
actual
					// incremental move
```

```
        setStatus(10);          // Set the status to 10 or normal

        DebugPrint("");DebugPrint("");
        DebugPrint("Vision Triggered.  Current Task: "+shtTaskStep);
        if(shtTaskStep != 99)
        {
                int[] pos = new int[2];

                if(shtTaskStep == 10)
                {
                        if(int_1_a_edge_seek.Result == 0)               //search for
beginning of strut
                        {
                                setTaskStep(11);                                //if
found go to template
                                setVisionUpdatedFlag();
                        }
                        else
                        {
                                setRelativePosition(10,0);              //if not move
in x direction
                                setVisionUpdatedFlag();
                        }
                }
                else if(shtTaskStep == 11)
                {
                        if(tmp_1_a_fidseek.Result == 0)                 //search for
template
                        {
                                setAbsolutePosition(getTemplateAdjust());
                                setTaskStep(12);                                //if
found go to adjustment
                                setVisionUpdatedFlag();
                        }
                        else
                        {
                                setRelativePosition(0,5);               //if not move
in y direction
                                setVisionUpdatedFlag();
                        }
                }
                else if(shtTaskStep == 12)
                {
                        pos = getCoarseAdjustment();                    //perform
coarse adjustment
```

```
                        if(getStatus() == 10)                                   //and if
nominal move
                        {
                                setAbsolutePosition(pos);
                                setTaskStep(13);
                                setVisionUpdatedFlag();
                        }
                        else
                        {
                                setRelativePosition(0,0);
                                setTaskStep(11);
                                setVisionUpdatedFlag();
                                DebugPrint("Error condition of: "+getStatus());
                        }
                }
                else if(shtTaskStep == 13)
                {
                        pos = getFineAdjustment();
        //perform fine adjustment
                        if(getStatus() == 10)
                        {
                                setAbsolutePosition(pos);
                                setTaskStep(99);
        //signal the end of the realign
                                setTaskCompletedFlag();
                                setVisionUpdatedFlag();
                        }
                        else
                        {
                                setRelativePosition(0,0);
                                setTaskStep(11);
                                setVisionUpdatedFlag();
                        }
                }
        }

        if(getTaskAckFlag() == 11)          // If a task has been acknowledged
        {
                DebugPrint("Task Ack'd clearing flags");
                setTaskStep(10);                        // reset the task counter for next
new ring
                clearTaskCompletedFlag();
                RegisterWriteShort(1050,0);
                RegisterWriteShort(1052,0);
                RegisterWriteShort(1054,0);
                RegisterWriteShort(1056,0);
```

```
			RegisterWriteShort(1058,0);
			RegisterWriteShort(1060,0);
			RegisterWriteShort(1062,0);
			RegisterWriteShort(1064,0);
			RegisterWriteShort(1066,0);
			RegisterWriteShort(1068,0);
			RegisterWriteShort(1070,0);
		}


} // end Inspect()



} // end Class
```

15.2. scr_Stent_Scan.dvtscr

```
class scr_Stent_Scan
{

//*********************************************************************
//*			Standard Register Methods			*
//*********************************************************************
public int getMaxError()	// This holds the maximum area allowed
{					// for a defect on the strut
	return 75;			// 50 pixels
}
public void setStatus(short stat)
{
	RegisterWriteShort(1032, stat);
}
public short getStatus()
{
	return RegisterReadShort(1032);
}
public short getInPositionFlag()
{
	return RegisterReadShort(14);
}
public short getVisionUpdatedFlag()
{
	return RegisterReadShort(1132);
}
public void setVisionUpdatedFlag()
{
	RegisterWriteShort(1132, 11);
```

```
}
public void clearVisionUpdatedFlag()
{
        RegisterWriteShort(1132, 10);
}
public short getTaskAckFlag()
{
        return RegisterReadShort(20);
}
public short getTaskCompetedFlag()
{
        return RegisterReadShort(1024);
}
public void setTaskCompletedFlag()
{
        RegisterWriteShort(1024,11);
}
public void clearTaskCompletedFlag()
{
        RegisterWriteShort(1024,10);
}
public short getTaskStep()
{
        return RegisterReadShort(1030);
}
public void setTaskStep(short step)
{
        RegisterWriteShort(1030,step);
}
public int[] getOffset()
{
        int[] off = new int[2];
        off[0] = RegisterReadShort(1026);
        off[1] = RegisterReadShort(1028);
        return off;
}
public void setOffset(short x, short y)
{
        RegisterWriteShort(1026,x);
        RegisterWriteShort(1028,y);
        getOffset();
}
public void setAbsolutePosition(short x, short y)
{
        RegisterWriteShort(1128, x);
        RegisterWriteShort(1130, y);
```

```
}
public void setAbsolutePosition(int[] pos)
{
        RegisterWriteShort(1128, pos[0]);
        RegisterWriteShort(1130, pos[1]);
}
public void setRelativePosition(short x, short y)
{
        int[] off = getOffset();
        RegisterWriteShort(1128, x+off[0]);
        RegisterWriteShort(1130, y+off[1]);
}
public void setInspectionPass()
{
        RegisterWriteShort(1034, 11);
}
public void setInspectionFail()
{
        RegisterWriteShort(1034, 10);
}
public static int getTotalArea(Sensor b)   // Given a blob sensor this will
{                                                   // determine the total area of all
        int sum = 0;                        // blobs
        for(int i=0; i<b.NumBlobs; i++)
        {
                sum += b.BlobArea[i];
        }
        return sum;
}
public int getErrorArea(Sensor b)          // This will report total area of
{                                                   // all blobs excluding the largest
        int sum = 0;                        // blob found
        int max = 0;

        for(int i=0; i<b.NumBlobs; i++)
        {
                sum += b.BlobArea[i];
                if(b.BlobArea[i] > max)
                {
                        max = b.BlobArea[i];
                }
        }

        sum -= max;

        return sum;
```

```
}
public int getMaxAreaIndex(Sensor b)    // returns the index of the largest
{                                                   // blob by area
        int ind = 0;
        int max = 0;

        for(int i=0; i<b.NumBlobs; i++)
        {

                if(b.BlobArea[i] > max)
                {
                        max = b.BlobArea[i];
                        ind = i;
                }
        }

        return ind;

}
public void setSensorStatus(int sid, int err)
{
        int stat = sid*100+err;
        RegisterWriteShort(1036, stat);
}
public int getSensorStatus()
{
        return RegisterReadShort(1036);
}
public void incrementRegister(int reg)
{
        int tmp = RegisterReadShort(reg);
        tmp = tmp + 1;
        RegisterWriteShort(reg, tmp);
}
//*******************************************************************


//*******************************************************************
//      Adjustment & Realign sensors
//*******************************************************************
public int[] getCoarseAdjustment()
{
        DebugPrint("Starting Coarse Adjustment");
        int[] pos = new int[2];                         // Set up variables for the
pos
        pos = getOffset();                      // get the current offset
        if(RegisterReadShort(16) == RegisterReadShort(22))  //test for end ring
        {       // End ring alignment required.

                DebugPrint("CA: End Ring");


        }
        else
        {       // Middle ring alignment required.

                DebugPrint("CA: Middle Ring");

                if(bls_1_a_topstr.NumBlobs != 1) // Verify that only 1 blob is
                {                                       // found by the alignment
blobs
                        setStatus(11);                  // otherwise alignment will
be
                }                                       // wrong when it
repositions
                else if(bls_1_a_botstr.NumBlobs != 1)
                {
                        setStatus(12);
                }
                else if(bls_1_a_fid.NumBlobs != 1)
                {
                        setStatus(13);
                }
                else
                {
                        DebugPrint("CA: Calculating movement");
                        // The coarse adjustment is a rotational (Y) axis adjustment
                        // and is required to be performed before an X adjust on the
                        // keyhole can be reliably used.

                        // The averages of all the struts should balance to TDC in
                        // the Y axis so this is just a simple average calcuation
                        pos[1] = (((bls_1_a_topstr.BlobPosition.Y[0]
                                        + bls_1_a_botstr.BlobPosition.Y[0])/2)
                                        + bls_1_a_fid.BlobPosition.Y[0])/2;
                }
        }
        DebugPrint("CA: Final Coords X:"+pos[0]+" Y:"+pos[1]);
        return pos;
}
```

```
public int[] getFineAdjustment()
{
        DebugPrint("Starting Fine Adjustment");
        int[] pos = new int[2];
        pos = getOffset();

        if(RegisterReadShort(16) == RegisterReadShort(22))
        {
                DebugPrint("FA: End Ring");
        // end ring

        }
        else
        {
                DebugPrint("FA: Middle Ring");
                // middle ring
                if(bls_1_a_topstr.NumBlobs != 1)  // Verify that only 1 blob is
                {                                         // found by the alignment
blobs
                        setStatus(14);                    // otherwise alignment will
be
                }                                         // wrong when it
repositions
                else if(bls_1_a_botstr.NumBlobs != 1)
                {
                        setStatus(15);
                }
                else if(bls_1_a_fid.NumBlobs != 1)
                {
                        setStatus(16);
                }
                else if(bls_1_a_key.NumBlobs != 2)
                {
                        setStatus(17);
                }
                else
                {
                        DebugPrint("FA: Calculating Movement");
                        // calculate the total movement.  The Y movement is again
done
                        // via the blobs set on the struts to perform rotational
                        // alignment.  The blobs that fall across the keyhole are
                        // used for the horizontal adjustment based on their
                        // average position and a given offset.
                        pos[1] = (((bls_1_a_topstr.BlobPosition.Y[0] +
bls_1_a_botstr.BlobPosition.Y[0])/2)
```

```
                                        + bls_1_a_fid.BlobPosition.Y[0])/2;
                        pos[0] = 
(bls_1_a_key.BlobPosition.X[0]+bls_1_a_key.BlobPosition.X[1])/2 - 190;
                }
        }
        DebugPrint("FA: Final Coords X:"+pos[0]+" Y:"+pos[1]);
        return pos;
}
//********************************************************************

//********************************************************************
//      Inspection sensor testing methods
//********************************************************************
public boolean checkStrutBlob(Sensor  b, int maxerror, int maxperimeter, int 
maxarea, int sid)
{
                setSensorStatus(sid, 00);
                float pmin = 0.50; // percent of max the min value is

                // This will test the strut sensors for multiple conditions
                // The first is if there is 1 blob - does it fit within the defined
                // specifications for the sensor in question.
                // The test is done on both area and perimeter, with separate
                // fail codes for statistical and debugging reasons.

                if(b.NumBlobs == 1)     // check condition of 1 blob first
                {
                        if(     (b.BlobArea[0] <= maxarea) &&   // if blob (strut) is 
smaller than the max
                                (b.BlobArea[0] >= maxarea*pmin))        // and larger 
than the min area
                        {
                                if(     (b.BlobPerimeter[0] <= maxperimeter) && 
        // and smaller than the max
                                        (b.BlobPerimeter[0] >= maxperimeter*pmin) 
        // and larger than the min perimeter
                                )
                                {
                                        setSensorStatus(sid, 01);
                                        incrementRegister(1050);
                                        return true;    // pass this strut.
                                }
                                else
                                {
                                        setSensorStatus(sid, 03);
                                        incrementRegister(1054);
                                        return false;
                                }
                        }
                        else
                        {
                                setSensorStatus(sid,02);
                                incrementRegister(1052);
                                return false;
                        }
                }

                // If there is not 1 blob detected the previous block is skipped
                // and the test is then to see if there are 0 blobs found.
                // If there are no blobs located the sensor has misaligned, the
                // stent has been moved, or there is a major geometry defect
                // in which case this is a failing condition.

                if(b.NumBlobs == 0)
                {
                        setSensorStatus(sid, 08);
                        incrementRegister(1064);
                        return false;   // fail if no blobs found
                }

                // After the tests for 0 and 1 blob are completed the only
                // situation left is >1 blob.
                // When there is more than 1 blob the first step is to test
                // the total of the "error" blobs, which are blobs that are not
                // the strut being inspected.  It is assumed that the strut will
                // be the largest blob found, so that is the differentiation criteria

                int toterr = getErrorArea(b.BlobArea);
                if(toterr > maxerror)
                {
                        setSensorStatus(sid, 04);
                        incrementRegister(1056);
                        return false;   // if there's too much error
                }

                // If the area of the error blobs is within the tolerance set then
                // the test will continue by testing the total area of the error
                // and the strut itself to verify that it is within tolerance

                int totArea = getTotalArea(b);
                int ind = getMaxAreaIndex(b);
                int maxPeri = b.BlobPerimeter[ind];
```

```
		if(	(totArea > maxarea) ||		// if total is larger than the max
			(totArea < maxarea*pmin) )		// or smaller than the min
area
		{
			setSensorStatus(sid, 05);
			incrementRegister(1058);
			return false;
		}

		// If this is the case then the perimeter of the strut (largest blob)
		// is tested to verify that it doesn't buldge greatly, and that there
		// are no defects that would indicate the strut has been bent or
		// otherwise damaged.
	/*	if(	(b.BlobPerimeter[0] > maxperimeter + 25) ||	// and
smaller than the max
			(b.BlobPerimeter[0] < maxperimeter*pmin) )	// and larger
than the min perimeter
		{
			setSensorStatus(sid, 06);
			incrementRegister(1060);
			return false;
		}*/

		// When the test makes it to this stage all possible defective
		// conditions have been tested so the only possible condition
		// is a passing test.

		setSensorStatus(sid, 07);
		incrementRegister(1062);
		return true;	// if it hasn't been disqualified yet, it's valid.
}

public boolean checkBlackSpaceBlob(Sensor b, int sid)
{
	// This method will test the black space blob by first
	// seeing if there are blobs detected at all.  This is the ideal
	// and most common case for a non-defect.  If there are no blobs
	// found here then there are no cob-webs, bridges, or contamination
	// fibers running through this region

	if(b.NumBlobs == 0)
	{
		setSensorStatus(sid, 10);
		incrementRegister(1068);
		return true;
```

```
        }

        // If there are blobs tested the total area is computed for all blobs
        // that have a minimum of 2 pixels to them.  This will screen out
        // most of the 'random' noise that shows when looking in the blackspace
        // region of the inspection.

        if(getTotalArea(b) > 10)
        {
                setSensorStatus(sid, 09);
                incrementRegister(1066);
                return false;
        }

        // If the total area is under the error threshold above then this is
        // considered to be noise, and will be passed.

        setSensorStatus(sid, 11);
        incrementRegister(1070);
        return true;
}
//*********************************************************************

//*********************************************************************
//      Product inspection methods
//*********************************************************************
// These methods will be used to control what sensors are used
// to determine the pass or fail condition.  The main inspect method
// will call one of these based upon the current feature under test.

public boolean InspectProduct1()
{
        setSensorStatus(00, 00);
        int MAX_ERROR = getMaxError();
        // Inspection criteria for features 1 & 5
        DebugPrint("InspectionProduct1, Features 1,5");

        // Test webbing:
        if(checkBlackSpaceBlob(bls_1_w_1, 27)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_1_w_2, 28)==false)
        {
                return false;
        }
```

```
if(checkBlackSpaceBlob(bls_1_w_3, 29)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_1_w_4, 30)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_1_w_5, 31)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_1_w_6, 32)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_1_w_7, 33)==false)
{
        return false;
}

DebugPrint("Blackspace Passed");

if(checkStrutBlob(bls_1_c_ms2s, MAX_ERROR, 355, 3825, 5)==false)
{
        return false;
}
if(checkStrutBlob(bls_1_c_bss, MAX_ERROR, 245, 2705, 2)==false)
{
        return false;
}
if(checkStrutBlob(bls_1_c_ts2s, MAX_ERROR, 325, 4095, 7)==false)
{
        return false;
}
if(checkStrutBlob(bls_1_c_tss, MAX_ERROR, 280, 3165, 8)==false)
{
        return false;
}
if(checkStrutBlob(bls_1_c_bs2s, MAX_ERROR, 295, 3630, 1)==false)
{
        return false;
}
if(checkStrutBlob(bls_1_c_khs, MAX_ERROR, 495, 5200, 4)==false)
{
        return false;
        }
        if(checkStrutBlob(bls_1_c_mss, MAX_ERROR, 285, 3000, 6)==false)
        {
                return false;
        }
        if(checkStrutBlob(bls_1_c_cs, MAX_ERROR, 425, 5265, 3)==false)
        {
                return false;
        }


        DebugPrint("Strut Passed");

        return true;
}

public boolean InspectProduct2()
{
        setSensorStatus(00, 00);
        int MAX_ERROR = getMaxError();

        // Inspection criteria for features 2,4,8 & 10
        DebugPrint("InspectionProduct2, Features 2,4,8,10");
        if(checkBlackSpaceBlob(bls_2_w_1, 34)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_2_w_2, 35)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_2_w_3, 36)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_2_w_4, 37)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_2_w_5, 38)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_2_w_6, 39)==false)
        {
                return false;
        }

        DebugPrint("Blackspace Passed");

        if(checkStrutBlob(bls_2_c_bss, MAX_ERROR, 480, 5695, 9)==false)
        {
                DebugPrint("Damn");
                return false;
        }
        DebugPrint("mmkay");
        if(checkStrutBlob(bls_2_c_tss, MAX_ERROR, 450, 6250, 11)==false)
        {
                return false;
        }
        if(checkStrutBlob(bls_2_c_cs, MAX_ERROR, 370, 4255, 10)==false)
        {
                return false;
        }

        DebugPrint("Strut Passed");

        return true;
}

public boolean InspectProduct3()
{
        setSensorStatus(00, 00);
        int MAX_ERROR = getMaxError();

        // Inspection criteria for feature 3
        DebugPrint("InspectionProduct3, Features 3");

        if(checkBlackSpaceBlob(bls_3_w_1, 40)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_3_w_2, 41)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_3_w_3, 42)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_3_w_4, 43)==false)
        {
```

```
			return false;
		}
		if(checkBlackSpaceBlob(bls_3_w_5, 44)==false)
		{
			return false;
		}
		if(checkBlackSpaceBlob(bls_3_w_6, 45)==false)
		{
			return false;
		}

		DebugPrint("Blackspace Passed");

		if(checkStrutBlob(bls_3_c_bss, MAX_ERROR, 385, 4625, 12)==false)
		{
			return false;
		}
		if(checkStrutBlob(bls_3_c_tss, MAX_ERROR, 400, 5225, 14)==false)
		{
			return false;
		}
		if(checkStrutBlob(bls_3_c_ms, MAX_ERROR, 570, 6955, 13)==false)
		{
			return false;
		}


		DebugPrint("Strut Passed");

		return true;
	}

	public boolean InspectProduct4()
	{
		setSensorStatus(00, 00);
		int MAX_ERROR = getMaxError();

		// Inspection criteria for features 6 & 12
		DebugPrint("InspectionProduct4, Features 6,12");
		if(checkBlackSpaceBlob(bls_6_w_1, 46)==false)
		{
			return false;
		}
		if(checkBlackSpaceBlob(bls_6_w_2, 47)==false)
		{
			return false;
```

```
        }
        if(checkBlackSpaceBlob(bls_6_w_3, 48)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_6_w_4, 49)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_6_w_5, 50)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_6_w_6, 51)==false)
        {
                return false;
        }
        if(checkBlackSpaceBlob(bls_6_w_7, 52)==false)
        {
                return false;
        }

        DebugPrint("Blackspace Passed");

        if(checkStrutBlob(bls_6_c_bss, MAX_ERROR, 375, 4905, 15)==false)
        {
                return false;
        }
        if(checkStrutBlob(bls_6_c_tss, MAX_ERROR, 360, 4600, 17)==false)
        {
                return false;
        }
        if(checkStrutBlob(bls_6_c_cs, MAX_ERROR, 430, 5150, 16)==false)
        {
                return false;
        }

        DebugPrint("Strut Passed");

        return true;
}

public boolean InspectProduct5()
{
        setSensorStatus(00, 00);
        int MAX_ERROR = getMaxError();
```

```
// Inspection criteria for features 7 & 9
DebugPrint("InspectionProduct5, Features 7,9");
if(checkBlackSpaceBlob(bls_7_w_1, 53)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_2, 54)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_3, 55)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_4, 56)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_5, 57)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_6, 58)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_7, 59)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_8, 60)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_9, 61)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_10, 62)==false)
{
        return false;
}
if(checkBlackSpaceBlob(bls_7_w_11, 63)==false)
{
        return false;
```

```
	}
	if(checkBlackSpaceBlob(bls_7_w_12, 64)==false)
	{
		return false;
	}

	DebugPrint("Blackspace Passed");

	if(checkStrutBlob(bls_7_c_bss, MAX_ERROR, 385, 4865, 18)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_7_c_tss, MAX_ERROR, 375, 4640, 22)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_7_c_ms2s, MAX_ERROR, 420, 5105, 20)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_7_c_mss, MAX_ERROR, 360, 4325, 21)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_7_c_cs, MAX_ERROR, 540, 6815, 19)==false)
	{
		return false;
	}

	DebugPrint("Strut Passed");

	return true;
}

public boolean InspectProduct6()
{
	setSensorStatus(00, 00);
	int MAX_ERROR = getMaxError();

	// Inspection criteria for feature 11
	DebugPrint("InspectionProduct6, Feature 11");
	if(checkBlackSpaceBlob(bls_11_w_1, 65)==false)
	{
		return false;
	}
	if(checkBlackSpaceBlob(bls_11_w_2, 66)==false)
	{
		return false;
	}
	if(checkBlackSpaceBlob(bls_11_w_3, 67)==false)
	{
		return false;
	}
	if(checkBlackSpaceBlob(bls_11_w_4, 68)==false)
	{
		return false;
	}
	if(checkBlackSpaceBlob(bls_11_w_5, 69)==false)
	{
		return false;
	}
	if(checkBlackSpaceBlob(bls_11_w_6, 70)==false)
	{
		return false;
	}

	DebugPrint("Blackspace Passed");

	if(checkStrutBlob(bls_11_c_bss, MAX_ERROR, 240, 2700, 24)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_11_c_bs2s, MAX_ERROR, 300, 3600, 23)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_11_c_tss, MAX_ERROR, 330, 3825, 26)==false)
	{
		return false;
	}
	if(checkStrutBlob(bls_11_c_cs, MAX_ERROR, 395, 4501, 25)==false)
	{
		return false;
	}

	DebugPrint("Strut Passed");

	return true;
}
//*******************************************************************
```

```
//********************************************************************
//      Main inspection method, this is called for each trigger of the camera
//********************************************************************
public void inspect()
{
        short shtTaskStep = getTaskStep();      // Get the current task step

        setOffset(320, 240); // Center of the view - offset that should be used
                                        // from the 0 position when calculating the
actual
                                        // incremental move.  This is used since
different
                                        // products could have different offsets


        int intCurrentFeature;          // The current feature under the camera

        setStatus(10);          // Set the status to 10 or normal

        intCurrentFeature = RegisterReadShort(18);      // Get the value of the
        short shtCurrentRing = RegisterReadShort(16);// current feature and
                                                                // ring so that the
proper
                                                                // product can be
used

        DebugPrint("");DebugPrint("");                  // Spaces on the debug
        DebugPrint("Inspection Started. Current Position:"+intCurrentFeature
                        +" Current Ring: "+shtCurrentRing);     // This is debug
information

        // If on the first ring, and the alignment isn't complete then run
        // the following code.  This will perform the 2 step realignment procedure
        // that is used after each ring advancement.
        // The other part of the if block is to make sure a realign is not
        // started on the beginning ring right after a stent edge align has been
        // executed.  Otherwise the script and G-code will desyncronize.
        if(intCurrentFeature == 0 && shtTaskStep != 99 && shtCurrentRing != 1)
        {
                int[] pos = new int[2];         // Positional array to use for movement

                DebugPrint("New ring found, current task: "+shtTaskStep);
                if(shtTaskStep == 10)           // Task 10 is the coarse adjustment
                {
                        pos = getCoarseAdjustment();
                        if(getStatus() == 10)// If the adjustment returned normal
                        {                               // then it is safe to adjust stent
                                setAbsolutePosition(pos);   // Set the new center
                                setTaskStep(11);        // and then increment the task
step
                        }
                        else
                        {
                                DebugPrint("Error condition of: "+getStatus());
                                setRelativePosition(0,0);
                        }
                }
                else if(shtTaskStep == 11)
                {
                        pos = getFineAdjustment();              // perform the fine
adjustment
                        if(getStatus() == 10)           // and if nominal move
                        {
                                setAbsolutePosition(pos);
                                setTaskCompletedFlag();
                                setTaskStep(99);                // signal the end of the
realign
                        }
                        else
                        {
                                DebugPrint("Error Condition of: "+getStatus());
                                setRelativePosition(0,0);
                                setTaskStep(10);
                                //setVisionUpdatedFlag();
                        }
                }
        }
        else    // If there is not during a ring alignment then perform an inspection
        {       // of the given feature.

                // Since there are 4 repeating features for each ring it is
                // possible to divide by 4 and use the remainder (modulo) to select
                // which product to use
                short shtInspectionFeature = intCurrentFeature % 4;


                if(shtCurrentRing == 1)
                {
                        // beginning ring
                        if(shtInspectionFeature == 0)
                        {
                                if(InspectProduct1())
```

```
				{
					setInspectionPass();
					DebugPrint("Feature 1 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 1 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else if(shtInspectionFeature == 1)
			{
				if(InspectProduct2())
				{
					setInspectionPass();
					DebugPrint("Feature 2 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 2 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else if(shtInspectionFeature == 2)
			{
				if(InspectProduct3())
				{
					setInspectionPass();
					DebugPrint("Feature 3 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 3 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else if(shtInspectionFeature == 3)
			{
				if(InspectProduct2())
				{
					setInspectionPass();
					DebugPrint("Feature 4 : Inspection Passed");
				}
```

```
                        else
                        {
                                setInspectionFail();
                                DebugPrint("Feature 4 : Inspection Failed:
Status Code: " + getSensorStatus());
                        }
                }
                else
                {
                DebugPrint("Error Condition - Beginning Ring Else Clause");
                }
        }
        else if(shtCurrentRing == RegisterReadShort(22))
        {
                // ending ring
                if(shtInspectionFeature == 0)
                {
                        if(InspectProduct5())
                        {
                                setInspectionPass();
                                DebugPrint("Feature 9 : Inspection Passed");
                        }
                        else
                        {
                                setInspectionFail();
                                DebugPrint("Feature 9 : Inspection Failed:
Status Code: " + getSensorStatus());
                        }
                }
                else if(shtInspectionFeature == 1)
                {
                        if(InspectProduct2())
                        {
                                setInspectionPass();
                                DebugPrint("Feature 10 : Inspection Passed");
                        }
                        else
                        {
                                setInspectionFail();
                                DebugPrint("Feature 10 : Inspection Failed:
Status Code: " + getSensorStatus());
                        }
                }
                else if(shtInspectionFeature == 2)
                {
                        if(InspectProduct6())
```

```
                                        {
                                                setInspectionPass();
                                                DebugPrint("Feature 11 : Inspection Passed");
                                        }
                                        else
                                        {
                                                setInspectionFail();
                                                DebugPrint("Feature 11 : Inspection Failed:
Status Code: " + getSensorStatus());
                                        }
                                }
                                else if(shtInspectionFeature == 3)
                                {
                                        if(InspectProduct4())
                                        {
                                                setInspectionPass();
                                                DebugPrint("Feature 12 : Inspection Passed");
                                        }
                                        else
                                        {
                                                setInspectionFail();
                                                DebugPrint("Feature 12 : Inspection Failed:
Status Code: " + getSensorStatus());
                                        }
                                }
                                else
                                {
                                DebugPrint("Error Condition - Ending Ring Else Clause");
                                }
                        }
                        else
                        {
                                // middle rings
                                if(shtInspectionFeature == 0)
                                {

                                        if(InspectProduct1())
                                        {
                                                setInspectionPass();
                                                DebugPrint("Feature 5 : Inspection Passed");
                                        }
                                        else
                                        {
                                                setInspectionFail();
                                                DebugPrint("Feature 5 : Inspection Failed:
Status Code: " + getSensorStatus());
```

```
				}
			}
			else if(shtInspectionFeature == 1)
			{

				if(InspectProduct4())
				{
					setInspectionPass();
					DebugPrint("Feature 6 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 6 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else if(shtInspectionFeature == 2)
			{
				if(InspectProduct5())
				{
					setInspectionPass();
					DebugPrint("Feature 7 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 7 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else if(shtInspectionFeature == 3)
			{
				if(InspectProduct2())
				{
					setInspectionPass();
					DebugPrint("Feature 8 : Inspection Passed");
				}
				else
				{
					setInspectionFail();
					DebugPrint("Feature 8 : Inspection Failed:
Status Code: " + getSensorStatus());
				}
			}
			else
```

```
            {
            DebugPrint("Error Condition - Middle Ring Else Clause");
            }
        }
    }

    if(getTaskAckFlag() == 11)          // If a task has been acknowledged
    {
        DebugPrint("Task Ack'd clearing flags");
        setTaskStep(10);                // reset the task counter for next new
ring
        clearTaskCompletedFlag();
    }


} // end Inspect()



} // end Class
```