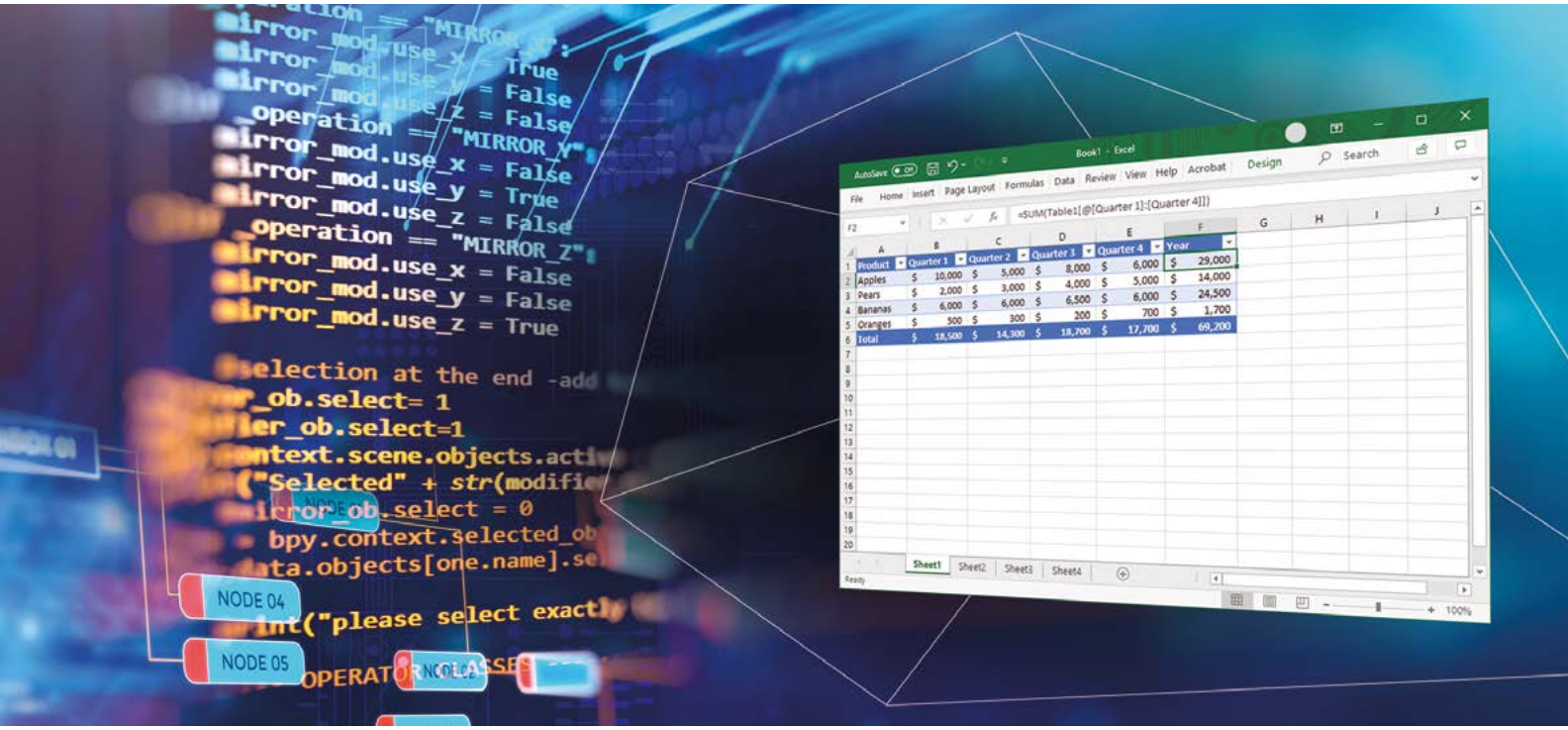
PROGRAMMING A VISION APPLICATION

Why the spreadsheet interface is a valuable approach to machine vision applications

COGNEX

PROGRAMMING A VISION APPLICATION

Why the spreadsheet interface is a valuable approach to machine vision applications



Machine vision systems are becoming ever more critical in automated manufacturing. These systems are also a means for obtaining significant amounts of data that can be used to continuously improve product quality. Before the full benefits of machine vision can be realized, the inspection application itself must be developed. This task can feel daunting, at first.

Historically, there have been two primary ways that machine vision system applications are developed: coding them using programmable machine vision libraries or using a graphical user interface (GUI). Each approach has both its pros and cons.

On the one hand, conventional programming methods provide the most flexibility; however, hiring programmers can be difficult and costly and ongoing maintenance of the application is also expensive. On the other hand, point-and-click GUIs are easier to use, but their limitations often times means they cannot solve more complex applications.

Fortunately, there is a third approach to creating vision applications, one that offers the ease of use of a GUI environment with the flexibility and control of a programming language: the spreadsheet.

If you have ever used spreadsheet, you have programmed a computer. While most people tend to think of Microsoft Excel or Google Sheets as computer applications, a spreadsheet is actually a powerful computer programming environment. Spreadsheets have also demonstrated over the years to be the best environment for not only storing and organizing data, but also for analyzing data.

The Benefits of Spreadsheets

Cognex made the decision to use a spreadsheet as the programming methodology for its line of In-Sight vision systems based on three key benefits:

1. Vision systems generate large amounts of data. Solving a vision application requires sequencing the correct functions and manipulating the resulting data to make a decision. Spreadsheets are recognized as the most efficient, accessible method of manipulating lots of data.

2. Spreadsheets are already widely used. Given that the number of people who can use a spreadsheet is far greater than the number of formally trained software developers, the In-Sight spreadsheet enables large numbers of users to create and maintain their vision application programs in a familiar, flexible environment.

3. Hiring qualified software programmers is expensive and point-and-click GUIs have inherent limitations to solving complex applications. The spreadsheet provides the right balance of ease of use and flexibility so that manufacturers and machine builders have an application development platform that easily adapts as processes and products change.

In short, leveraging the spreadsheet offers advantages over both a GUI interface and programmable vision libraries: a programming environment for non-programmers, offering more flexibility and control, lower development and support costs.

The In-Sight Spreadsheet

The In-Sight spreadsheet, although designed for machine vision, behaves similar to a conventional spreadsheet. It comes with a powerful vision tools library to address guidance, inspection, gauging, and identification applications.

“Cognex added numerous vision-specific functions such as pattern matching, finding an edge, and blobs to turbocharge the spreadsheet. This is in addition to all the standard math functions, formulas, and cell referencing one would expect,” says Rick Roszkowski, one of the early engineers on Cognex’s machine vision team.

The Efficiency of Cell Referencing

The spreadsheet is also efficient thanks to the concept of cell referencing. This is a method for referring to a cell or series of cells in a spreadsheet formula.

An individual cell within a spreadsheet contains data and that data could be different types, such as integers, floating point numbers, strings, or even functions that generate new data. The spreadsheet can pass these numbers and functions from cell to cell. That’s what we mean by “cell referencing.” Finally, the cell name, denoted by its column/row location, becomes what traditional computer programmers would call a variable.

In a factory automation case, take a formula for measuring the distance between two edges. When an application engineer creates that formula, it allows them to measure the distance between two edges only once. But if they want to measure the distance between edges 50 times across a part doing that in a GUI or programming language gets tedious, says Roszkowski.

“Cell referencing in spreadsheets allows a user to simply copy the formulas in one row of cells and paste them into rows of cells below in order to measure multiple distances at the same time,” he notes.

A second benefit of cell referencing happens during I/O and factory communications. Cell referencing makes it easy to aggregate application results from different sources in the spreadsheet to create a "message" that is then sent to the programmable logic controller (PLC).

Finally, a very practical example of cell referencing in the real world is when a manufacturer is building three variants of an assembly. While there might be many shared parts among those assemblies, each assembly has some unique features that need to be differentiated on the production line. Those differences will be sent from the PLC to tell the program which variant is being assembled and which inspection must take place. That input from the PLC to the program can ultimately be reduced to a simple cell referencing problem that nearly everyone already knows how to solve.

“It’s a huge efficiency in terms of time and effort and man hours,” says Roszkowski.

Spreadsheets are, after all, a way for users to organize and work with large amounts of data. While most vision systems can export individual results during production, the In-Sight spreadsheet can write data to universally compatible, comma delimited text file. That file can then easily be used in any business intelligence application.

Other Benefits

Over time, it is clear there are other value-propositions for the spreadsheet interface aside from efficiency when programming a machine vision application. In particular, the spreadsheet is a very flexible environment for just about every type of machine vision need.

While most machine vision applications focus on 2D inspections with a traditional rules-based approach to solving them, the coming wave of smart factories employing various Industry 4.0 technologies will require companies to leverage a broad range of machine vision capabilities — deep learning and other cognitive technologies, and 3D inspections, for example. These newer inspection approaches will require a more robust user interface and the spreadsheet is designed to seamlessly integrate these new capabilities.

“The potential for this is unlimited for factories and manufacturers,” says Roszkowski.

For example, if a part is measured and the measurement falls between 20 and 22 units, it is deemed a good part. To determine it is indeed a good part, the engineering team conducted a feasibility study to identify that accepted range. It turns out that 95% of warranty claims on that part occur at 20 units and not 22. A company could, therefore, use that data to determine that 20 is not within the acceptable range based on high warranty expenses. They can adjust their warranties and the accepted measurement range of the part, shifting business practices to account for what’s actually happening on the factory floor.

Building a Factory Automation Application in Spreadsheet

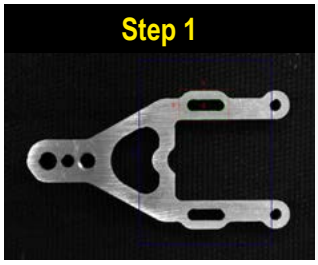
Here is a simple and high-level introduction to building an application in spreadsheet. In this example, we will walk through the steps of training and finding a pattern on a machined metal part, finding the mid-point distance between two patterns, and then finally, detecting the number of holes, or “blobs”, on the part itself.

The intention is to demonstrate how efficient and easy it is for non-programmers to set up an inspection in the spreadsheet interface.

For reference,
the numbers within the spreadsheet are corresponding pixel coordinates on the image, which itself is 1600 pixels wide by 1200 pixels high.

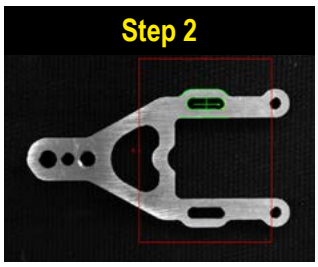
Train/Find Patterns

In the first step of our application, we are using the pattern tool to train and find a pattern within an image taken of the part. This can be accomplished in the spreadsheet using just a handful of cells, and finding a second pattern within the image simply creates a new row of cells. The total output of our efforts is also contained within just a handful or rows and columns (columns D-F).



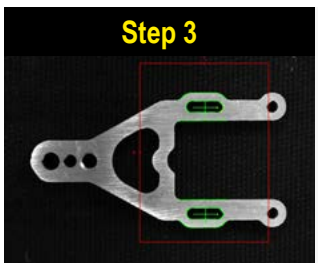
B3 = TrainPatMaxPattern(A0,0,0,0,267.936,895.608,155.624,244.053,0,0,0,2,12,0,0,0,0,0,0,0,5000,0)

	A	B	C	D	E	F	G
0	Image						
1							
2		Train Patterns	Find Patterns	Row	Col	Angle	
3	Top Slot	Patterns	Patterns	347.75	1029.63	0.00	



C3 = FindPatMaxPatterns(A0,0,0,0,125,700,900,650,0,0,0,B3,1,50,10,0,0,-15,15,100,100,0,100,100,

	A	B	C	D	E	F	G
0	Image						
1							
2		Train Patterns	Find Patterns	Row	Col	Angle	
3	Top Slot	Patterns	Patterns	347.75	1029.63	0.00	



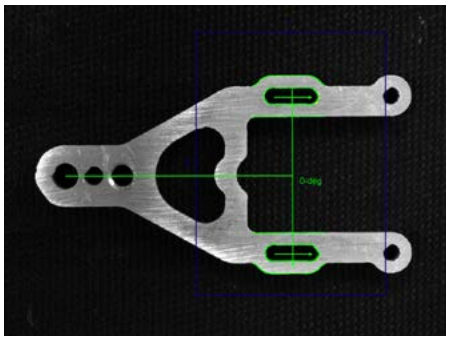
C4 = FindPatMaxPatterns(A0,0,0,0,125,700,900,650,0,0,0,B4,1,50,10,0,0,-15,15,100,100,0,100,100,

	A	B	C	D	E	F	G
0	Image						
1							
2		Train Patterns	Find Patterns	Row	Col	Angle	
3	Top Slot	Patterns	Patterns	347.75	1029.63	0.00	
4	Bottom Slot	Patterns	Patterns	883.81	1031.03	0.00	

As indicated in the previous three images, we have found two patterns, identified their location within the image (the pattern is located at the image coordinate that is 1029.63 pixels over from the left and 347.75 pixels down from the top) and recorded the angle of the two patterns within the image.

Part Orientation

Now that we've located the part in the image, we can determine whether the part is oriented correctly. After finding the patterns, we can use simple math in the spreadsheet to calculate the midpoint between the patterns and calculate the rotation of the part. This is cell referencing in action.



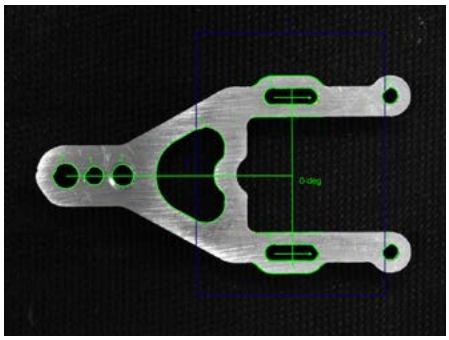
F7 = PlotString(D7,D6,E6+25,8,1)

	A	B	C	D	E	F	G
0	Image						
1							
2		Train Patterns	Find Patterns	Row	Col	Angle	
3	Top Slot	Patterns	Patterns	347.75	1029.63	0.00	
4	Bottom Slot	Patterns	Patterns	883.81	1031.03	0.00	
5							
6			MidPoint:	615.78	1030.33	Plot	Plot
7			Rotation:	0-deg		Plot	

The midpoint between the two patterns is at image coordinate 615.78, 1030.33, which is calculated from cells D3 and D4.

Detect Blobs

Now that we found two patterns, calculated the midpoint between those two patterns, and confirmed the orientation of the part, we can now find and confirm that all the holes within the part are present.

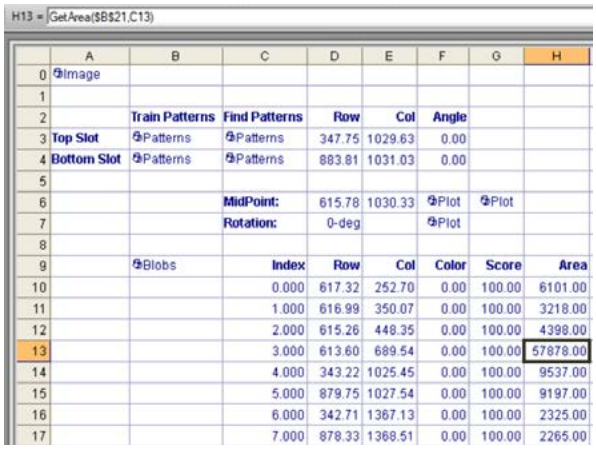


4	Bottom Slot	Patterns	Patterns	883.81	1031.03	0.00			
5									
6			MidPoint:	615.78	1030.33	Plot	Plot		
7			Rotation:	0-deg		Plot			
8									
9		Blobs	Index	Row	Col	Color	Score	Area	Elon
10			0.000	617.32	252.70	0.00	100.00	6101.00	
11			1.000	616.99	350.07	0.00	100.00	3218.00	
12			2.000	615.26	448.35	0.00	100.00	4398.00	
13			3.000	613.60	689.54	0.00	100.00	57878.00	
14			4.000	343.22	1025.45	0.00	100.00	9537.00	
15			5.000	879.75	1027.54	0.00	100.00	9197.00	
16			6.000	342.71	1367.13	0.00	100.00	2325.00	
17			7.000	878.33	1368.51	0.00	100.00	2265.00	

Holes are most often referred to as "blobs" within vision applications, so we use the Blob tool to find and count the holes. The Blob tool counts 8 holes in total and outputs the data in the Index (C9) column starting with 0.000 on the spreadsheet.

Get Blob Statistics

Users can quickly identify the largest hole in the part just by looking at the spreadsheet. The Area (H9) column outputs the size of the holes. Scanning down the column, users identify the largest hole in the part as hole number 3.000 based on having the largest area amount.



H13 = GetArea(B21,C13)

	A	B	C	D	E	F	G	H
0	Image							
1								
2		Train Patterns	Find Patterns	Row	Col	Angle		
3	Top Slot	Patterns	Patterns	347.75	1029.63	0.00		
4	Bottom Slot	Patterns	Patterns	883.81	1031.03	0.00		
5								
6			MidPoint:	615.78	1030.33	Plot	Plot	
7			Rotation:	0-deg		Plot		
8								
9		Blobs	Index	Row	Col	Color	Score	Area
10			0.000	617.32	252.70	0.00	100.00	6101.00
11			1.000	616.99	350.07	0.00	100.00	3218.00
12			2.000	615.26	448.35	0.00	100.00	4398.00
13			3.000	613.60	689.54	0.00	100.00	57878.00
14			4.000	343.22	1025.45	0.00	100.00	9537.00
15			5.000	879.75	1027.54	0.00	100.00	9197.00
16			6.000	342.71	1367.13	0.00	100.00	2325.00
17			7.000	878.33	1368.51	0.00	100.00	2265.00

Building the Same Inspection Application Via Programming

As a comparison, here is how the same pattern finding and blob detection inspections would be accomplished via a more traditional programmatic approach. It is evident how efficient the spreadsheet interface is in comparison to using programmable machine vision libraries.

Find a Pattern

```
// train patquick pattern
ccPMAlignPattern pat;
win.window(40,40,70,70);
pat.train(win);

// set origin to upper left of black square
pat.origin(cc2Vect(50,50));

// display trained pattern
ccDisplayConsole trainDisp(ccIPair(300, 300),
                           cmT("Trained Patquick Pattern"));
trainDisp.image(pat.trainImage());

ccUITablet uiTablet;

// display the trained fine-grain features
pat.displayFeatures(uiTablet, true);
trainDisp.drawSketch(uiTablet.sketch(), ccDisplayConsole::eImageCoords);

// run patquick
ccPMAlignRunParams params;
params.numToFind(3);
params.zoneEnable(ccPMAlignDefs::kUniformScale);
params.zone(ccPMAlignDefs::kUniformScale, 0.8, 1.2);
ccPMAlignResultSet set;
pat.run(image, params, set);

// display results
cogOut << cmT("time (ms) = ") << set.time() * 1000 << cmStd endl;
cogOut << cmT("number found = ") << set.numFound() << cmStd endl;

ccDisplayConsole runDisp(ccIPair(300, 300), cmT("Found Patquick Pattern"));
runDisp.image(image, false);
runDisp.fit();

for (int i = 0; i < set.numFound(); i++)
{
  ccPMAlignResult r;
  set.getResult(i, r);

  cogOut << cmT("result ") << i << cmT(" =  (x,y):(") << r.location().x()
         << cmT(",") << r.location().y() << cmT(")") << cmT("  ")
         << (r.accepted() ? cmT("accepted") : cmT("not accepted"))
         << cmStd endl;

  const ccColor* c = r.accepted() ? &ccColor::green : &ccColor::red;
  uiTablet.drawPointIcon(ccPoint(r.location()), *c);
}

runDisp.drawSketch(uiTablet.sketch(), ccDisplayConsole::eClientCoords);

return 0;
```

Blob Detection

```
ccBlobParams blobParams;
blobParams.setSegmentationHardThresh(50, false);

ccBlobEnhancedParams blobEnhParams;

ccBlobEnhancedResultSet results;

cfBlobEnhancedAnalysis(image, blobParams, blobEnhParams, results,
                       &diagObject, diagFlags);

for (unsigned int i = 0 ; i < results.results().size() ; i++)
{
  for(int j=0;j < results.results()[i].featureletChainSet().numChains();j++)
  {
    ccPolylineModelPtrh polyline =
      cfPolylineShapeModel(results.results()[i].featureletChainSet(),j);

    cc2Vect com = polyline->centerArea();
    cogOut
      << cmT("Feature: ") << cmStd setw(3) << i+1 << cmT(" ")
      << cmStd setw(3) << j+1
      << cmT(" Area: ") << cmStd setw(3) << polyline->area()
      << cmT(" Center of mass: (") << com.x() << cmT(",") << com.y()
      << cmT(")") << cmStd endl;
  }
}

// display results
ccDiagServer::init();
ccDiagServer::showDiagObject(0,0,diagObject);
cfWaitForContinue();
ccDiagServer::exit();

return 0;
```

In-Sight Spreadsheet – Now with Deep Learning

Cognex has been a leader in the machine vision industry since 1981, introducing many of the key technologies that the manufacturing industry have come to depend on. When the In-Sight spreadsheet was introduced in 2000, it made machine vision technology accessible to large numbers of users for the very first time. Thousands of manufacturers all over the world now use the In-Sight spreadsheet to quickly and easily solve their guidance, inspection, gauging, and identification applications.

In addition to providing access to the industry leading vision tool library, the In-Sight spreadsheet provides access to powerful example-based deep learning technology for optical character recognition (OCR), assembly verification, and defect detection inspection applications that previously could not be solved with traditional rule-based machine vision approaches.

The In-Sight spreadsheet is a platform for lowering development costs, shorten the time to deploy applications, lower support costs, and more easily on-board new automation engineers to develop vision applications. Also, with the trend to move to Industry 4.0 and the smart factory, vision application developers can count on the In-Sight spreadsheet to help meet these challenges with industry leading vision and deep learning technologies.

BUILD YOUR VISION

2D VISION SYSTEMS

Cognex machine vision systems are unmatched in their ability to inspect, identify and guide parts. They are easy to deploy and provide reliable, repeatable performance for the most challenging applications.

www.cognex.com/machine-vision

3D VISION SYSTEMS

Cognex In-Sight laser profilers and 3D vision systems provide ultimate ease of use, power and flexibility to achieve reliable and accurate measurement results for the most challenging 3D applications.

www.cognex.com/3D-vision-systems

VISION SOFTWARE

Cognex vision software provides industry leading vision technologies, from traditional machine vision to deep learning-based image analysis, to meet any development needs.

www.cognex.com/vision-software

BARCODE READERS

Cognex industrial barcode readers and mobile terminals with patented algorithms provide the highest read rates for 1D, 2D and DPM codes regardless of the barcode symbology, size, quality, printing method or surface.

www.cognex.com/barcodereaders

COGNEX

Companies around the world rely on Cognex vision and barcode reading solutions to optimize quality, drive down costs and control traceability.

Corporate Headquarters **One Vision Drive Natick, MA 01760 USA**

Regional Sales Offices

Americas

North America	+1 844-999-2469
Brazil	+55 (11) 2626 7301
Mexico	+01 800 733 4116

Europe

Austria	+49 721 958 8052
Belgium	+32 289 370 75
France	+33 1 7654 9318
Germany	+49 721 958 8052
Hungary	+36 800 80291
Ireland	+44 121 29 65 163
Italy	+39 02 3057 8196
Netherlands	+31 207 941 398
Poland	+48 717 121 086
Spain	+34 93 299 28 14
Sweden	+46 21 14 55 88
Switzerland	+41 445 788 877
Turkey	+90 216 900 1696
United Kingdom	+44 121 29 65 163

Asia

China	+86 21 6208 1133
India	+9120 4014 7840
Japan	+81 3 5977 5400
Korea	+82 2 530 9047
Malaysia	+6019 916 5532
Singapore	+65 632 55 700
Taiwan	+886 3 578 0060
Thailand	+66 88 7978924
Vietnam	+84 2444 583358

Lit. No. SpreadsheetWP-02-2020

www.cognex.com